Combined Complexity of Probabilistic Query Evaluation

Mikaël Monet

université PARIS-SACLAY

BNNT : 2018SACLT003

Thèse de doctorat

TELECOM ParisTech

Une école de l'IMT

Combined Complexity of Probabilistic Query Evaluation

Thèse de doctorat de l'Université Paris-Saclay
préparée à Télécom ParisTech

École doctorale n° 580 STIC
Spécialité de doctorat : informatique

Thèse présentée et soutenue à Télécom ParisTech, le 12 octobre 2018, par

Mikaël Monet

Composition du jury :

Antoine Amarilli Maître de Conférences, Télécom ParisTech	Directeur de thèse
Florent Capelli Maître de Conférences, Université de Lille	Examinateur
Georg Gottlob Professor, University of Oxford	Rapporteur
Benny Kimelfeld Associate Professor, Technion	Rapporteur
Dan Olteanu Professor, University of Oxford	Examinateur
Pierre Senellart Professeur, ENS, Université PSL	Directeur de thèse
Cristina Sirangelo Professeure, Université Paris-Diderot	Présidente
Emmanuel Waller Maître de Conférences, Université Paris Sud 11	Examinateur

Abstract

Query evaluation over probabilistic databases (*probabilistic query evaluation* or PQE) is known to be intractable in many cases, even in data complexity, i.e., when the query is fixed. Although some restrictions of the queries and instances have been proposed to lower the complexity, these known tractable cases usually do not apply to combined complexity, i.e., when the query is not fixed. My thesis investigates the question of which queries and instances ensure the tractability of PQE in combined complexity.

My first contribution is to study PQE of conjunctive queries on binary signatures, which we rephrase as a probabilistic graph homomorphism problem. We restrict the query and instance graphs to be trees and show the impact on the combined complexity of diverse features such as edge labels, branching, or connectedness. While the restrictions imposed in this setting are quite severe, my second contribution shows that, if we are ready to increase the complexity in the query, then we can evaluate a much more expressive language on more general instances. Specifically, we show that PQE for a particular class of Datalog queries on instances of bounded *treewidth* can be solved with linear complexity in the instance and doubly exponential complexity in the query. To prove this result, we use techniques from tree automata and knowledge compilation. The third contribution is to show the limits of some of these techniques by proving general lower bounds on knowledge compilation and tree automata formalisms.

L'évaluation de requêtes sur des données probabilistes (*probabilistic query evaluation* ou PQE) est généralement très coûteuse en ressources et ce même à requête fixée. Bien que certaines restrictions sur les requêtes et les données aient été proposées pour en diminuer la complexité, les résultats existants ne s'appliquent pas à la complexité *combinée*, c'est-à-dire quand la requête n'est pas fixe. Ma thèse s'intéresse à la question de déterminer pour quelles requêtes et données l'évaluation probabiliste est faisable en complexité combinée.

La première contribution de cette thèse est d'étudier PQE pour des requêtes conjonctives sur des schémas d'arité 2. Nous imposons que les requêtes et les données aient la forme d'arbres et montrons l'importance de diverses caractéristiques telles que la présence d'étiquettes sur les arêtes, les bifurcations ou la connectivité. Les restrictions imposées dans ce cadre sont assez sévères, mais la deuxième contribution de cette thèse montre que si l'on est prêts à augmenter la complexité en la requête, alors il devient possible d'évaluer un langage de requête plus expressif sur des données plus générales. Plus précisément, nous montrons que l'évaluation probabiliste d'un fragment particulier de Datalog sur des données de *largeur d'arbre* bornée peut s'effectuer en temps linéaire en les données et doublement exponentiel en la requête. Ce résultat est prouvé en utilisant des techniques d'automates d'arbres et de compilation de connaissances. La troisième contribution de ce travail est de montrer les limites de certaines de ces techniques, en prouvant des bornes inférieures générales sur la taille de formalismes de représentation utilisés en compilation de connaissances et en théorie des automates.

Remerciements

Je souhaite remercier tous ceux qui, de près ou de loin, ont contribué à faire de ma thèse une expérience intéressante et enrichissante.

Commençons par les deux personnes que je tiens le plus pour responsables du succès de cette aventure : mes deux directeurs de thèse, Pierre et Antoine. Lorsque je commençais mon stage de Master avec Pierre il y a un peu plus de trois ans, je n'avais aucune idée d'où je mettais les pieds. La principale raison pour laquelle j'avais choisi ce stage était qu'il se déroulait en partie à Singapour. Le lieu m'avait l'air exotique et le sujet vaguement intriguant, mais surtout je n'avais toujours pas purgé les deux mois à l'étranger que mon cursus aux Mines requérait. À part ça, je ne connaissais essentiellement rien au monde de la recherche, et rien à la théorie des bases de données. Pierre et Antoine m'ont introduit au domaine et à la communauté, et de manière générale ont su m'encadrer à la perfection. Ils ont toujours été disponibles lorsque j'avais besoin de conseils, que ce soit sur des questions de recherche, d'enseignement, d'informatique pratique (Linux, LaTeX...), d'administratif, d'anglais, de carrière, de cuisine, de tourisme... J'ai énormément appris à leur côté et ai beaucoup apprécié travailler avec eux. Je resterai toujours impressionné par leur efficacité visiblement sans limites, leur culture générale dense et variée, leur professionnalisme, et leur bienveillance ; Pierre, Antoine, vous êtes pour moi de véritables modèles.

Je remercie également mes (autres) co-auteurs : Silviu Maniu, avec qui nous avons beaucoup discuté tentacules, Pierre Bourhis, qui m'a souvent donné de précieux avis sur la recherche, et Dan Olteanu, qui m'a invité faire un stage de trois mois avec lui à Oxford qui m'a beaucoup plu.

Cette thèse n'aurait pas pu être soutenue sans l'accord des membres de mon jury, qui ont bénévolement accepté la responsabilité de juger ces trois ans de travail. Je les remercie tous pour leur temps, et (un peu) plus particulièrement mes deux rapporteurs, Benny et Georg, qui ont méticuleusement relu le manuscrit et m'ont donné l'occasion de l'améliorer. Je sais en effet à quel point relire et commenter un texte d'informatique théorique peut être long et laborieux.

Quand ce n'était pas la science qui me poussait hors de mon lit pour aller à Télécom le matin, c'était souvent la perspective de discussions stimulantes et parfois improbables avec les membres de l'équipe. Je pense évidemment à Jean-Benoît (sans qui cette thèse serait sans doute trop grosse), Oana, Luis, Marie, Marc, Miyoung, Pierre-Alexandre, Julien, Fabian, Nicoleta, Jacob, Ziad, Atef, les deux Thomas, Mostafa, Katerina, Quentin, Maroua, Camille, Mauro, Jean-Louis, Jonathan, et à tous ceux qui sont passés plus brièvement à DBWeb. L'ambiance de cette équipe était exceptionnelle.

Je remercie aussi mes (autres) amis, ceux que j'ai vu régulièrement pendant ma thèse tout comme ceux que je vois plus rarement. Yvon, Denis, Thibaut, Emmanuel, Adrien, Gaël, Julien, Damien, Éléonore, Antoine, Alexandre, Myriam, Maxence, Camille, Claire, Jean-Charles, Charles, Laurent, Mélanie, Nicolas, Pierre, Pablo et Solange (Oxford c'était cool en grande partie grâce à vous !), Quentin, Loïc, Benoît, Emmanuel, les deux Tom, Caroline, Maëlig, Pierre, Olivier, et Théo : il y aurait beaucoup à raconter et j'oublie certainement des noms, et même si cela n'a pas

forcément de rapport avec cette thèse je tiens à vous dire que suis content de vous avoir rencontrés et de côtoyer chacun de vous.

Un peu moins personnellement, j'accorde une pensée à Télécom et Inria pour avoir financé ma thèse et les nombreux voyages que j'ai eu la chance de faire dans ce cadre : Singapour, San Francisco, Nice, Chicago, Oxford, Venise, Stockholm, Londres, Vienne, Cali, et Bucarest.

Je suis reconnaissant envers toute ma famille pour leur soutient inébranlable depuis ma naissance. Je pense en particulier à mes parents, qui ont toujours fait en sorte que je ne manque de rien, qui m'ont donné des racines et des ailes, et qui m'ont aussi donné une merveilleuse petite sœur, Marlène. Contrairement aux amis, pas besoin de lister le reste de la famille ici, normalement ils savent qui ils sont :) Merci à vous d'être présents dans ma vie. Avoir eu l'occasion lors de ma soutenance de vous raconter un peu mieux ce que j'ai fait pendant ces trois ans m'a rempli de joie.

Enfin, je m'adresserai à celle qui partage ma vie depuis maintenant quatre ans. Nelly, je suis heureux d'avoir traversé cette étape à tes côtés, et c'est en grande partie grâce à toi que j'en suis arrivé au bout en un seul morceau. Tous ces voyages ont plus de sens lorsque tu es avec moi, et j'attends avec impatience le début de notre nouvelle aventure au Chili !

Table of Contents

General Introduction

The field of database research in computer science emerged from the need to store, access and query information. The foundations of this domain arguably lie in the relational model, introduced by Edgar Codd in the late sixties [Codd 1970] and based on the well-established formalism of first-order logic. The relational model owes its success to its genericity: it provides a way of thinking about data that is abstract enough to be used in many different contexts. This success can be observed through the everyday usage of Relational Database Management Systems (RDBMSs) all around the world, such as Oracle, MySQL, PostgreSQL, and many others.

Traditional database research and RDBMSs tend to assume that the data is reliable and complete. Yet, data uncertainty naturally appears in many real-life situations. This uncertainty in data can come in various forms. For instance, when information is automatically extracted from arbitrary web pages, uncertainty can be introduced due to the inherent ambiguity of natural language processing. In road monitoring systems, uncertainty may come from the lack of recent information about traffic [Hua and Pei 2010]: is the road congested? is there a diversion? Due to hardware limitations, in the field of experimental sciences, measurement errors also occur in many databases [Asthana, King, Gibbons, and Roth 2004]. Even when crisp data can be obtained, it can still be the case that we do not trust who retrieved it or how it came to us. Querying these databases without considering this uncertainty can lead to incorrect answers. For some of these scenarios, specific algorithms have been developed to deal with the uncertainty, but these greatly depend on the domain of application and thus do not form a unified framework. One first attempt to generically capture data uncertainty in RDBMSs is the notion of NULLs. However, NULLs can only represent missing or unknown values, and certainly cannot represent *quantitative uncertainty* about tuples.

To address this challenge, probabilistic databases [Suciu, Olteanu, Ré, and Koch 2011] have been introduced. Their goal is to generically capture data uncertainty and reason about it, much like the relational model was introduced to generically work with non-probabilistic data. A natural way to capture database uncertainty is to explicitly represent all possible states of the data, i.e., all possible databases, called the *possible worlds*, and associate to each world a probability value. The probability that a probabilistic database satisfies a given Boolean query is then the sum of the probabilities of the possible worlds that satisfy this query. This is the *probabilistic query evaluation problem*, or PQE: given a Boolean query Q and a probabilistic database D, compute the probability that D satisfies Q. The problem with this approach is that there can be exponentially many such possible worlds, so that it is not feasible in practice to represent the data and to query it in this way. Nevertheless, we can efficiently *represent* uncertain data if we make some independence assumptions: for instance, each tuple could be annotated with a probability to be present or absent, independently of the other tuples. This is the framework of *tuple-independent databases* [Lakshmanan, Leone, Ross, and Subrahmanian 1997; Dalvi and Suciu 2007], abbreviated as TID. More elaborate probabilistic representation systems also exist [Barbará, Garcia-Molina, and Porter

1992; Ré and Suciu 2007; Green and Tannen 2006; Huang, Antova, Koch, and Olteanu 2009; Suciu, Olteanu, Ré, and Koch 2011], but in this thesis we will only work with the TID model, because it will already be quite challenging.

Unfortunately, even when we can efficiently *represent* probabilistic data (using independence assumptions), we may not be able to efficiently *query* it. For instance, consider the following conjunctive query $q_{\text{hard}} : \exists xy\, R(x) \wedge S(x, y) \wedge T(y)$. It is intractable to compute its probability on arbitrary TID instances [Dalvi and Suciu 2007]. Specifically, this problem is #P-hard, where #P is the complexity class of counting problems whose answer can be expressed as the number of accepting paths of a nondeterministic polynomial-time Turing machine. For instance, a typical #P-complete problem is #SAT, that of counting the number of satisfying assignments of a propositional formula [Valiant 1979].

To address the intractability of PQE on TID instances, three general approaches have been proposed. The first one is to slightly relax the problem by asking to compute *approximations* of the resulting probabilities. To do this, it is always possible to resort to Monte-Carlo sampling [Fishman 1986; Ré, Dalvi, and Suciu 2007; Jampani et al. 2008], which provides an *additive approximation* of the query probability. This is of limited use, however, when probabilities are very low: the running time of Monte-Carlo sampling is quadratic in the desired precision, so we cannot use it in cases where we want a high precision. In this thesis, we will not be interested in approximate probability computation, so we focus on the other two approaches. Our point of view is that one first has to understand the difficulties of exact probability computation in order to determine if approximating methods are needed.

The two other approaches follow a general idea in computer science: when faced with the hardness of a problem, imposing restrictions on the input might recover tractability. Beyond their theoretical interest, such restrictions are also relevant in practice. Indeed, in real-life applications the input is generally not completely arbitrary but instead has some kind of structure or respects some properties that an algorithm could take advantage of. In our case, the first approach exploits the structure of the *query*, while the second exploits that of the *data*:

- **Restricting the queries.** Dalvi and Suciu [Dalvi and Suciu 2012] have shown a complete characterization of the unions of conjunctive queries (UCQs) for which PQE is tractable to evaluate on TID instances: a UCQ is either *safe* and PQE is PTIME, or it is not safe and PQE is #P-hard. However, as it turns out, many simple queries are already hard to evaluate, such as q_{hard} above.

- **Restricting the instances.** Recent work from our group [Amarilli, Bourhis, and Senellart 2015] has shown that, when the TID instances are taken to be of *bounded treewidth*, then we can evaluate in linear time the probability of any fixed Monadic Second-Order (MSO) query Q. MSO is an extension of first-order logic where quantification over sets of elements is allowed. In particular, it can express any UCQ.

Treewidth is a graph measure that describes how far an instance is to being a tree. Bounding the treewidth of instances is a well-known natural criterion to ensure the tractability of many problems that are NP-hard on arbitrary instances.

These two approaches focus on what is called the *data complexity*, i.e., the complexity of the problem as a function of the size of the input database, considering the query to be fixed. Yet, in practice, the queries are not fixed but are also given as input by the user, so the complexity also needs to be reasonable in the query. A better measure is thus the *combined complexity*, that is, the complexity of the problem as a function of the sizes of both the data and the query. For this measure, the algorithm of [Dalvi and Suciu 2012] is superexponential [Suciu, Olteanu, Ré, and Koch 2011], and that of [Amarilli, Bourhis, and Senellart 2015] is not even elementary [Thatcher and Wright 1968; Meyer 1975]. Hence, even when PQE is tractable in data complexity, the task may still be infeasible because of unrealistically large constants that depend on the query.

At first glance, it seems unreasonable to want a tractable combined complexity for PQE, because already in the non-probabilistic setting the combined complexity is usually not tractable. For instance, evaluating an arbitrary Boolean conjunctive query (CQ) on an arbitrary database is an NP-complete problem. However, in the non-probabilistic case, some research has been done to try to isolate cases where query evaluation is tractable in combined complexity. For example, Yannakakis's algorithm can evaluate α-acyclic queries on non-probabilistic instances with tractable combined complexity [Yannakakis 1981]. For these reasons, I believe that it is also important to achieve a good understanding of the combined complexity of PQE, and to isolate cases where PQE is tractable in combined complexity. This motivates the main question studied in this thesis: *for which classes of queries and instances does PQE enjoy reasonable combined complexity?*

To achieve reasonable combined complexity, the idea of this thesis is to impose restrictions on *both* the queries and the instances considered. We briefly outline here the main contributions of this thesis, before presenting them in more detail in the rest of the introduction:

1. I have studied PQE of conjunctive queries on binary signatures, which we can rephrase as a probabilistic graph homomorphism problem. We restrict the query and instance graphs to be trees and show the impact on the combined complexity of diverse features such as edge labels, branching, or connectedness. This yields a surprisingly rich complexity landscape, but where tractable cases are sadly very limited.

2. I showed that, in the non-probabilistic setting, the evaluation of a particular class of Datalog queries on instances of bounded treewidth can be solved in time linear in the product of the instance and the query. To show this result we used techniques from tree automata and provenance computation, and we introduced a new provenance representation formalism as cyclic Boolean circuits. Our result captures the tractability of such query classes as two-way regular path queries and α-acyclic conjunctive queries, and has connections to PQE, as we explain in the next point.

3. The third contribution is to show how we can apply the results of the second contribution to PQE by transforming these cyclic circuits into a class of Boolean circuits with strong properties from the domain of *knowledge compilation*, namely, *deterministic decomposable negation normal forms* (d-DNNFs), on which we can efficiently perform probabilistic evaluation. This allows us to

solve PQE for our class of Datalog queries (from the last point) on instances of bounded treewidth with a complexity linear in the data and doubly exponential in the query, in contrast with the nonelementary bound in combined complexity from [Amarilli, Bourhis, and Senellart 2015] for MSO queries. More generally, we study the connections between different circuit classes from knowledge compilation, and we also point to the limits of some of these techniques by proving lower bounds on knowledge compilation formalisms.

4. Unrelated to the combined complexity of PQE, I have also worked on the compilation of *safe* queries from [Dalvi and Suciu 2012] to d-DNNFs and conducted experiments that suggest that a certain class of safe queries can indeed be compiled efficiently to such circuits.

Although we decided to present our PhD research in the context of probabilistic databases, the scope of our contributions is not restricted to PQE. For instance, our work on the evaluation of Datalog on treelike instances implies new results on the combined complexity of non-probabilistic query evaluation and on the computation of *provenance information*. Our third contribution has applications in the field of knowledge compilation, and our first contribution has links with *constraint satisfaction problems*.

We now present our contributions in more details in the last four sections of this introduction.

Limits of Combined Tractability of PQE

In Chapter 2 of the thesis we study the combined complexity of PQE for conjunctive queries on TID instances. Conjunctive queries form one of the simplest query language for relational databases. They correspond to the basic SQL queries build from the SELECT, FROM, and WHERE directives. Our goal is to isolate cases with very strong tractability guaranties, namely, polynomial-time combined complexity. Remember that this is not possible in general because already in data complexity PQE can be #P-hard on arbitrary instances even for some simple conjunctive queries (such as the query q_{hard} mentioned above). Hence we will need to impose restrictions on the problem.

The first general restriction that we will make is to restrict our attention to binary schemas, where every relation has arity two. This is a natural restriction that is common in the world of *graph databases* or *knowledge bases*. For example a knowledge base stores information in the form of *triples* such as (Elvis, Likes, donuts). This triple can be seen as a fact Likes(Elvis, donuts) on the arity-two predicate Likes. This way of storing information can also be seen as a graph where edges carry labels; for instance here we would have two nodes, "Elvis" and "donuts", and one directed edge between the first node and the second, labeled by "Likes". Hence for simplicity of exposition, we will phrase the problem in terms of graphs. Given a directed *query graph* G and a (directed) *instance graph* H, where each edge is annotated by a label, we say that H *satisfies* G if there exists a *homomorphism* from G to H (intuitively, a mapping from the nodes of G to the nodes of H that respects the structure of G). Now, given a directed *query graph* and a probabilistic *instance graph*, where each edge is annotated by a probability (and by a label), we must determine the

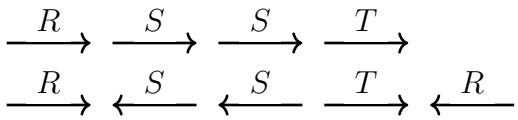

Figure 1 – Examples of labeled one-way path (top) and two-way path (bottom)

Figure 2 – Examples of unlabeled downward tree (left) and polytree (right)

probability that the instance graph satisfies the query graph. This is defined as the total probability mass of the subgraphs of H which satisfy G, assuming independence between edges.

Our second general restriction is to impose the query and instance graphs to have a tree shape. That is, we will restrict them to be *polytrees*, i.e., directed graphs whose underlying undirected graphs are trees. As we will see, however, even this restriction does not suffice to ensure tractability, so we study the impact of several other features:

- *Labels*, i.e., whether edges of the query and instance can be labeled by a finite alphabet, as would be the case on a relational signature with more than one binary predicate.
- *Disconnectedness*, i.e., allowing the graphs to be disconnected or not.
- *Branching*, i.e., allowing graphs to branch out, instead of requiring them to be a path.
- *Two-wayness*, i.e., allowing edges with arbitrary orientation, instead of requiring all edges to have the same orientation.

We accordingly study the combined complexity of PQE for *labeled graphs* and *unlabeled graphs*, and when the query and instance graphs are in the following classes, that cover the possible combinations of the characteristics above: one-way and two-way paths, downward trees and polytrees, and disjoint unions thereof. Figures 1 and 2 show some examples of a one-way path and a two-way path with labels, and of a downward tree and a polytree without labels.

Results. Our results completely classify the combined complexity of PQE for all combinations of instance and query restrictions that we consider, both in the labeled and in the unlabeled setting. In particular, we have identified four incomparable maximal tractable cases, reflecting various trade-offs between the expressiveness that we can allow in the queries and in the instances:

- in the unlabeled case, arbitrary queries on downward tree instances;

- in the labeled case, one-way path queries on downward tree instances;
- in the labeled case, connected queries on two-way path instances;
- in the unlabeled case, downward tree queries on polytree instances;

These tractability results all extend to disconnected instances, and other cases not captured by these restrictions are shown to be #P-hard. The (somewhat sinuous) tractability border is described in Tables 2.1, 2.2, and 2.3 on pages 39, 43, and 51. The proofs use a variety of technical tools: automata-based compilation to d-DNNF lineages as in [Amarilli, Bourhis, and Senellart 2015], β-acyclic lineages using [Brault-Baron, Capelli, and Mengel 2015], the $\underline{X}$-property for tractable CSP from [Gutjahr, Welzl, and Woeginger 1992], graded DAGs [Odagiri and Goto 2014], as well as various coding techniques for hardness proofs.

Unfortunately, these results show that the combined complexity of PQE quickly becomes intractable, for the strong notion of tractability that we had considered. However, this does not rule out more general tractability results for PQE if we impose a less restrictive notion of tractability.

Fixed-Parameter Tractability of Provenance Computation

We now lower our expectations regarding the complexity in the query, since we realized that cases where PQE is tractable in combined complexity are very restricted. We would like to find a more expressive query language and classes of instances for which the combined complexity is tractable in the data and "reasonable" in the query, this being motivated by the fact that the data is usually much bigger than the queries. To reach this goal, we will first focus on the combined complexity of *provenance computation* on non-probabilistic relational databases. Indeed, a common technique to evaluate queries on probabilistic databases is the *intensional approach*: first compute a representation of the *lineage* (or *provenance*) of the query on the database, which intuitively describes how the query depends on the possible database facts; then use this lineage to compute probabilities efficiently. Hence in Chapter 3 of the thesis we will focus on the computation of provenance information, with the hope that the lineages that we will obtain can be used in the probabilistic case.

Our starting point is the work of Courcelle, as initially presented in [Courcelle 1990] and summarized in [Flum, Frick, and Grohe 2002], which shows that for any fixed Boolean MSO query Q and constant bound k, we can evaluate Q in linear time on instances whose *treewidth* is bounded by k. Intuitively, instances of bounded treewidth can be decomposed into small "parts" that are themselves connected following the structure of a tree (we talk of a *tree decomposition*). As we already mentioned, bounding the treewidth of instances is a natural criterion to ensure that many problems that are hard on arbitrary instances become tractable. The proof of Courcelle's result is as follows: the query Q is translated into a *tree automaton* A, independently from the data but depending on the treewidth parameter k and the query. The instance I of treewidth $\leqslant k$ is then transformed into what is called a *tree encoding* T, which is just a tree decomposition encoded with a finite alphabet so as to be processable by a tree automaton. The automaton A is designed to ensure

that T is accepted by A iff Q holds on I (written $I \models Q$), so that we can evaluate Q on I in linear-time data complexity. Moreover, [Amarilli, Bourhis, and Senellart 2015] shows how to use these automata to compute *provenance circuits*, again in linear-time data complexity. However, this tells little about the combined complexity. Indeed the complexity of computing the automaton is nonelementary in the MSO query, making it the bottleneck of the whole process. Hence the questions that we investigate in Chapter 3: *Which queries can be efficiently translated into automata? For these queries, can we efficiently compute provenance information?*

Results. We will use the framework of *parameterized complexity theory* [Flum and Grohe 2006] in order to better understand what makes the translation to automata intractable. Parameterized complexity theory is a branch of complexity theory introduced in the nineties by Downey and Fellows (see, e.g., [Downey and Fellows 1992]) to study the complexity of a problem depending on some input parameter. In our case, rather than restricting to a fixed class of "efficient" queries, we study *parameterized* query classes, i.e., we define an efficient class of queries for each value of the parameter. We further make the standard assumption that the signature is fixed; in particular, its arity is constant. This allows us to aim for low combined complexity for query evaluation, namely, fixed parameter tractability with linear-time complexity in the product of the input query and instance, which we call *FPT-bilinear* complexity.

The translation of restricted query fragments to tree automata on treelike instances has already been used in the context of *guarded logics* and other fragments, to decide *satisfiability* [Benedikt, Bourhis, and Vanden Boom 2016] and *containment* [Barceló, Romero, and Vardi 2014]. To do this, one usually establishes a *treelike model property* to restrict the search to models of low treewidth (but dependent on the formula), and then translate the formula to an automaton, so that the problems reduce to emptiness testing. Inspired by these fragments, we consider the language of *clique-frontier-guarded Datalog* (CFG-Datalog), and show an efficient FPT-linear (in the query) translation procedure for this language, parameterized by the body size of rules. This implies FPT-bilinear combined complexity of query evaluation on treelike instances (parameterized by the body size and the treewidth bound). We show how the tractability of this language captures the tractability of such query classes as two-way regular path queries [Barceló 2013] and α-acyclic conjunctive queries. Regular path queries are an important building block of query languages for graph databases, while the class of α-acyclic queries [Yannakakis 1981] is the main known class of conjunctive queries to enjoy tractable combined complexity on arbitrary instances.

Instead of the *bottom-up tree automata* used in [Courcelle 1990; Flum, Frick, and Grohe 2002], and later in [Amarilli, Bourhis, and Senellart 2015] (which extended the result of Courcelle to probabilistic evaluation, but still in data complexity), we use the formalism of *alternating two-way automata*. These automata are more succinct than bottom-up automata, hence the translation to them is more efficient. In fact, we prove that bottom-up tree automata are not succinct enough even to translate α-acyclic conjunctive queries efficiently.

We can use our CFG-Datalog language to understand what make the translation to automata efficient for conjunctive queries. For conjunctive queries, we show that the treewidth of queries is not the right parameter to ensure efficient translation: we

prove that bounded-treewidth conjunctive queries cannot be efficiently translated to automata at all, so we cannot hope to show combined tractability for them via automata methods. By contrast, CFG-Datalog implies the combined tractability of bounded-treewidth queries with an additional requirement (interfaces between bags must be clique-guarded), which is the notion of *simplicial decompositions* previously studied by Tarjan in [Tarjan 1985]. CFG-Datalog can be understood as an extension of this fragment to disjunction, clique-guardedness, stratified negation, and inflationary fixpoints, that preserves tractability.

We then focus on the computation of provenance information for our Datalog fragment on bounded-treewidth instances. As we mentioned, [Amarilli, Bourhis, and Senellart 2015] showed how to compute provenance Boolean circuits from the *bottom-up tree automata* used by Courcelle. However, for our alternating two-way automata, the computation of provenance as Boolean circuits is not straightforward anymore. For this reason, we introduce a notion of *cyclic provenance circuits*, that we call *cycluits*. These cycluits are well-suited as a provenance representation for alternating two-way automata that encode CFG-Datalog queries, as they naturally deal with both recursion and two-way traversal of a treelike instance. While we believe that this natural generalization of Boolean circuits may be of independent interest, it does not seem to have been studied in detail. We show that cycluits can be evaluated in linear time. We further show that the provenance of an alternating two-way automata on a tree can be represented as a cycluit in FPT-bilinear time, generalizing results on bottom-up automata and circuits from [Amarilli, Bourhis, and Senellart 2015].

This implies we can compute in FPT-bilinear time the provenance cycluit of a CFG-Datalog query on a treelike instance (parameterized by the query body size and instance treewidth). We will then be able to use these cycluits for probabilistic evaluation, following the intensional approach to PQE.

From Cycluits to d-DNNFs and Lower Bounds

In Chapter 4 of the thesis we will investigate the connections between various provenance representations that can be used by the intensional approach for PQE, and outline the consequences for Chapter 3. We recall that the intensional approach consists of two steps. First, compute a representation of the lineage (or provenance) of the query on the database. The provenance of a Boolean query on a database is a Boolean function with facts of the database as variables that describes how the query depends on these facts. It can be represented with any formalism that is used to represent Boolean functions: Boolean formulas, Boolean circuits, binary decision diagrams, etc. Second, compute the probability of this Boolean function, which is precisely the probability that the probabilistic database satisfies the query. This can be seen as a weighted version of *model counting* (here, counting the number of satisfying assignments of a Boolean function). In order for this second step to be tractable, the representation formalism cannot be arbitrary. The field of *knowledge compilation* studies (among others) which formalisms allow tractable weighted model counting, the properties of these formalisms and the links between them. Thus, to evaluate queries on probabilistic databases, we can use knowledge compilation algorithms to translate circuits (or even the cycluits that we computed in Chapter 3) to tractable formalisms; conversely, lower bounds in knowledge compilation can

identify the limits of the intensional approach.

In this part of the thesis, we study the relationship between two kinds of tractable circuit classes in knowledge compilation: *width-based* classes, specifically, bounded-treewidth and bounded-pathwidth circuits; and *structure-based* classes, specifically, OBDDs (ordered binary decision diagrams [Bryant 1992], following a *variable order*) and d-SDNNFs (structured deterministic decomposable negation normal forms [Pipatsrisawat and Darwiche 2008], following a *v-tree*). Circuits of bounded treewidth can be obtained as a result of practical query evaluation [Jha, Olteanu, and Suciu 2010; Amarilli, Bourhis, and Senellart 2015], whereas OBDDs and d-DNNFs have been studied to show theoretical characterizations of the query lineages they can represent [Jha and Suciu 2011]. Both classes enjoy tractable probabilistic computation: for width-based classes, using *message passing* [Lauritzen and Spiegelhalter 1988], in time linear in the circuit and exponential in the treewidth; for OBDDs and d-SDNNFs, in linear time by definition of the class [Darwiche 2001]. Hence the question that we study in Chapter 4: *Can we compile width-based classes efficiently into structure-based classes?*

Results. We first study how to perform this transformation, and show corresponding *upper bounds*. Existing work has already studied the compilation of bounded-pathwidth circuits to OBDDs [Jha and Suciu 2012; Amarilli, Bourhis, and Senellart 2016]. Accordingly, we focus on compiling *bounded-treewidth circuits* to *d-SDNNF circuits*. We show that we can transform a Boolean circuit C of treewidth k into a d-SDNNF equivalent to C in time $O(|C| \times f(k))$ where f is singly exponential. This allows us to be competitive with message passing (also singly exponential in k), while being more modular. Beyond probabilistic query evaluation, our result implies that all tractable tasks on d-SDNNFs (e.g., enumeration [Amarilli, Bourhis, Jachiet, and Mengel 2017] and MAP inference [Fierens et al. 2015]) are also tractable on bounded-treewidth circuits.

We can then use this result to lift the combined tractability of provenance computation for CFG-Datalog on bounded-treewidth databases to probabilistic evaluation. Indeed, the treewidth of the constructed provenance cycluit is linear in the size of the Datalog query. However, we cannot directly apply our transformation from bounded-treewidth Boolean circuits to d-SDNNFs, because here we have a cycluit. Hence the first step is to transform this cycluit into a circuit, while preserving a bound on the treewidth of the result. We show that a cycluit of treewidth k can be converted into an equivalent circuit whose treewidth is singly exponential in k, in FPT-linear time (parameterized by k). We can then apply our transformation to this circuit, which in the end shows that we can compute a d-SDNNF representation of the provenance of a CFG-Datalog query Q of bounded rule-size on an instance I of bounded treewidth with a complexity linear in the data and doubly exponential in the query. This d-SDNNF then allows us to solve PQE for Q on I with the same complexity. While the complexity in the query is not polynomial time, we consider that it is a reasonable one, given that our language of CFG-Datalog is quite expressive (at least, much more so than the very restricted classes of CQs that we study in Chapter 2).

Second, we study *lower bounds* on how efficiently we can convert from width-based to structure-based classes. Our bounds already apply to a weaker formalism of width-based circuits, namely monotone *conjunctive normal form* (CNF) and *disjunctive*

normal form (DNF) formulas of bounded width. We connect the pathwidth of CNFs/DNF formulas with the minimal size of their OBDD representations by showing that any OBDD for a monotone CNF or DNF must be of width exponential in the pathwidth of the formula, up to factors depending in the formula arity (maximal size of clauses) and degree (maximal number of variable occurrences). Because it applies to *any* monotone CNF/DNF, this result generalizes several existing lower bounds in knowledge compilation that exponentially separate CNFs from OBDDs, such as [Devadas 1993] and [Bova and Slivovsky 2017, Theorem 19]. We also prove an analogue for treewidth and (d-)SDNNFs: again up to factors in the formula arity and degree, any d-SDNNF (resp., SDNNF) for a monotone DNF (resp., CNF) must have size that is exponential in the treewidth of the formula.

To prove our lower bounds, we rephrase pathwidth and treewidth to new notions of *pathsplitwidth* and *treesplitwidth*, which intuitively measure the performance of a variable ordering or v-tree. We also use the *disjoint non-covering prime implicant sets* (dncpi-sets), a tool introduced in [Amarilli, Bourhis, and Senellart 2016; Amarilli 2016] and that can be used to derive lower bounds on OBDD width. We show how they can also imply lower bounds on d-SDNNF size, using the recent communication complexity approach of [Bova, Capelli, Mengel, and Slivovsky 2016].

We then apply our lower bounds to intensional query evaluation on relational databases. We reuse the notion of *intricate* queries of [Amarilli, Bourhis, and Senellart 2016], and show that d-SDNNF representations of the lineage of these queries have size exponential in the treewidth of *any* input instance. This extends the result of [Amarilli, Bourhis, and Senellart 2016] from OBDDs to d-SDNNFs. As in [Amarilli, Bourhis, and Senellart 2016], this result shows that, on arity-2 signatures and under constructibility assumptions, treewidth is the right parameter on instance families to ensure that all queries (in monadic second-order) have tractable d-SDNNF lineage representations.

d-DNNFs for Safe Queries

As a last contribution, I have studied the connections between the intensional approach and the *safe* UCQs [Dalvi and Suciu 2012]. We make no assumption on the shape of the input data and only care about data complexity. As we already mentioned, when Q is a union of conjunctive queries (UCQ), a dichotomy result is provided by the celebrated result of [Dalvi and Suciu 2012]: either Q is *safe* and PQE is PTIME, or Q is *unsafe* and PQE is #P-hard. The algorithm to compute the probability of a safe UCQ exploits the first-order structure of the query to find a so-called *safe query plan* (using extended relational operators that can manipulate probabilities) and can be implemented within a RDBMS. This approach is referred to as *extensional query evaluation*, or *lifted inference*.

This is different from the *intensional query evaluation* that we have followed so far, where one first computes a representation of the lineage/provenance of the query Q on the database I, and then performs weighted model counting on the lineage to obtain the probability. To ensure that model counting is tractable, we use the structure of the query to represent the lineage in tractable formalisms from the field of knowledge compilation, such as read-once Boolean formulas, free or ordered binary decision diagrams (FBDDs, OBDDs), deterministic decomposable normal forms (d-DNNFs), decision decomposable normal forms (dec-DNNFs), deterministic

decomposable circuits (d-Ds), etc. The main advantage of this approach compared to lifted inference is that the provenance can help explain the query answer (this is also true for non-probabilistic evaluation, as provenance is defined for non-probabilistic data). Moreover, having the lineage in a tractable knowledge compilation formalism can be useful for other applications: we can for instance easily recompute the query result if the tuple probabilities are updated, or we can compute the most probable state of the database if we assume that the query is satisfied.

A natural question is to ask if the extensional and the intensional approaches are equally powerful. What we call the q_9 *conjecture*, formulated in [Jha and Suciu 2013; Dalvi and Suciu 2012], states that, for safe queries, extensional query evaluation is strictly more powerful than the knowledge compilation approach. Or, in other words, that there exists a query which is safe (i.e., that can be handled by the extensional approach) whose lineages on arbitrary databases cannot be computed in PTIME in a knowledge compilation formalism that allows tractable weighted model counting (i.e., cannot be handled by the intensional approach). Note that the conjecture depends on which tractable formalism we consider. The conjecture has recently been shown in [Beame, Li, Roy, and Suciu 2017] to hold for the formalism of dec-DNNFs (including OBDDs and FBDDs), which captures the execution of modern model counting algorithms. Another independent result [Bova and Szeider 2017] shows that the conjecture also holds when we consider the formalism of d-SDNNFs. However the status of the conjecture is still unknown for more expressive formalisms, namely, d-DNNFs and d-Ds. Indeed, it could be the case that the conjecture does not hold for such expressive formalisms, which would imply that the reason why PQE is PTIME for safe queries is because we can build deterministic decomposable circuits in PTIME for them.

Results. I have investigated whether we can disprove the conjecture for the classes of d-DNNFs and d-Ds. More specifically, we focus on a class of queries (the so-called $\mathcal{H}$*-queries*) that was proposed in [Jha and Suciu 2013; Dalvi and Suciu 2012] to separate the two approaches, and that was used to prove the conjecture for dec-DNNFs [Beame, Li, Roy, and Suciu 2017] and d-SDNNFs [Bova and Szeider 2017]. We develop a new technique to build d-DNNFs and d-Ds in polynomial time for the $\mathcal{H}$-queries, based on what we call *nice Boolean functions*. Because we were not able to prove that this technique works for all safe $\mathcal{H}$-queries, we test this technique with the help of the SAT solver Glucose [Audemard and Simon 2009] on all the $\mathcal{H}$ queries up to a certain size parameter, that we generated automatically (amounting to about 20 million nontrivial queries). We found no query on which it does not work, hence we conjecture that our technique can build d-Ds for all safe $\mathcal{H}$-queries. Interestingly, we found a few queries for which we can build d-Ds with a single internal negation at the very top, whereas we do not know if we can build d-DNNFs.

In order to compute all these $\mathcal{H}$-queries, we had to solve a task of independent interest, namely, computing explicitly the list of all monotone Boolean functions on 7 variables, up to equivalence. This task had previously been undertaken by Cazé, Humphries, and Gutkin in [Cazé, Humphries, and Gutkin 2013] and by Stephen and Yusun in [Stephen and Yusun 2014]. We reused parts of the code from [Cazé, Humphries, and Gutkin 2013] and confirmed the number of such functions: 490 013 148.

Structure of the Thesis

We first give technical preliminaries in Chapter 1. We then move on to the content of the thesis, starting with our work on the combined complexity of probabilistic graph homomorphism in Chapter 2. We continue in Chapter 3 with the evaluation of CFG-Datalog on bounded-treewidth non-probabilistic instances. Last, we show in Chapter 4 how to lift these results to probabilistic evaluation and also how to derive lower bounds on knowledge compilation formalisms. We then conclude.

The last contribution of my PhD research is not included in the thesis but was published at AMW'2018 [Monet and Olteanu 2018]. Chapters 2 to 4 of this thesis can be read independently, except for Section 4.5 in Chapter 4, which depends on Chapter 3.

Chapter 1

Background and General Preliminaries

We give in this chapter the definitions of the main concepts used throughout this thesis. Most of the concepts that are specific to some chapter are defined in the corresponding chapter rather than here.

1.1 Relational Databases, Graphs, Hypergraphs

Relational databases. A *relational signature* σ is a finite set of relation names (or *relational symbols*) written R, S, T, ..., each with its associated *arity* written $\text{arity}(R) \in \mathbb{N}$. The *arity* $\text{arity}(\sigma)$ of σ is the maximal arity of a relation in σ. A *σ-instance* I (or *σ-database*, or simply *instance* or *database* when the signature is clear from context) is a finite set of *facts* (or *tuples*) on σ, i.e., $R(a_1, \ldots, a_{\text{arity}(R)})$ with $R \in \sigma$ (sometimes simply noted $R(\mathbf{a})$), and where a_i is what we call an *element*. The *active domain* $\text{dom}(I)$ consists of the elements occurring in I, and the size of I, denoted $|I|$, is the number of tuples that I contains.

Example 1.1.1. Table 1.1 shows an example of relational instance I on signature $\sigma = \{R, S, T\}$ with $\text{arity}(R) = \text{arity}(S) = 2$ and $\text{arity}(T) = 3$. The active domain of I is $\text{dom}(I) = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$ and its size is $|I| = 11$. $\lhd$

A *subinstance* of I is a σ-instance that is included in I (as a set of tuples). An *isomorphism* between two σ-instances I and I' is a bijective function $f : \text{dom}(I) \to \text{dom}(I')$ such that for every relation name R, for each tuple $(a_1, \ldots, a_{\text{arity}(R)}) \in$

Table 1.1 – Example relational database

R	
3	7
3	4
5	4
2	5
9	10
7	8

S	
3	7
7	9
11	9
2	6

T		
1	2	3

$\mathrm{dom}(I)^{\mathrm{arity}(R)}$, we have $R(a_1, \ldots, a_{\mathrm{arity}(R)}) \in I$ if and only if $R(f(a'_1), \ldots, f(a'_{\mathrm{arity}(R)})) \in I'$. When there exists such an isomorphism, we say that I and I' are *isomorphic*: intuitively, isomorphic databases have exactly the same structure and differ only by the name of the elements in their active domains. A *homomorphism* between two σ-instances I and I' is a function $h : \mathrm{dom}(I) \to \mathrm{dom}(I')$ such that for every tuple $R(a_1, \ldots, a_{\mathrm{arity}(R)}) \in I$, we have $R(h(a'_1), \ldots, h(a'_{\mathrm{arity}(R)})) \in I'$. Hence, an isomorphism is simply a homomorphism that is bijective and whose inverse is also a homomorphism.

Graphs. In this thesis we consider various definitions of graphs, depending on what we need:

- Let σ be a finite non-empty set of *labels*. A *directed graph with edge labels from* σ is a triple $G = (V, E, \lambda)$ with V a set of vertices, with $E \subseteq V^2$ a set of edges, and with $\lambda : E \to \sigma$ a labeling function. We often write $a \xrightarrow{R} b$ for an edge $e = (a, b)$ with label $\lambda(e) = R$.
- A *directed unlabeled graph* $G = (V, E)$ consists of a set V of vertices and a set $E \subseteq V^2$ of edges, which we write $a \to b$. We can equivalently see a directed unlabeled graph as a directed labeled graph whose set of labels contains only one element.
- An *undirected labeled (resp., unlabeled) graph* is just like a directed labeled (resp., unlabeled) graph, except that we have $a \to b \in E$ iff $b \to a \in E$. In the unlabeled case, we sometimes write the edges $\{a, b\} \in E$.

Note that we do not allow multi-edges: there is at most one edge $a \to b$ between two elements a and b, and this edge has a unique label $\lambda(e)$ in the case of a labeled graph. We will sometimes only write "graph" when the kind of graph is clear from context. All the graphs that we consider in this thesis are *finite*, meaning that V (hence E) is a finite set.

Observe that a directed labeled graph can be seen as a relational instance in which all relational symbols have arity 2. These databases are in fact called *graph databases* and play an important role in many applications (social media, transportation networks, etc.).

Hypergraphs. Hypergraphs are a generalization of undirected unlabeled graphs where an edge can contain more than two vertices. Formally, a hypergraph $\mathcal{H} = (V, E)$ consists of a set V of vertices and a set $E \subseteq 2^V$ of *hyperedges* (or simply *edges*) which are subsets of V. For a node v of $\mathcal{H}$, we write $E(v)$ for the set of edges of $\mathcal{H}$ that contain v. The *arity* of $\mathcal{H}$, written $\mathrm{arity}(\mathcal{H})$, is the maximal size of an edge of $\mathcal{H}$. The *degree* of $\mathcal{H}$, written $\mathrm{degree}(\mathcal{H})$, is the maximal number of edges to which a vertex belongs, i.e., $\max_{v \in V} |E(v)|$. As with graphs, all hypergraphs that we will consider are finite. Given a σ-instance I on signature σ, we define the *hypergraph associated to* I as the hypergraph $\mathcal{H} = (V, E)$, whose set of vertices V is $\mathrm{dom}(I)$ and such that for every subset e of $\mathrm{dom}(I)$, e is in E if and only if I has a fact containing exactly the elements of e.

Example 1.1.2. Figure 1.1 displays the hypergraph associated to the relational instance I from Example 1.1.1. ◁

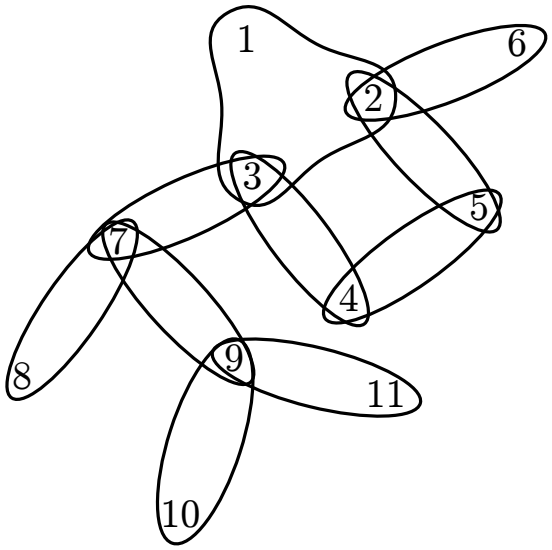

Figure 1.1 – Hypergraph associated to the relational instance from Example 1.1.1.

As we will see later, one often imposes restrictions on the structure of relational databases by restricting the shape of their associated hypergraphs.

Gaifman graph. The *Gaifman graph* (or *primal graph*) of a relational instance I is the undirected unlabeled graph G whose set of vertices is $\mathrm{dom}(I)$ and where two elements are connected by an edge if and only if they co-occur (appear together) in a fact. Observe that each fact of I induces a clique in the primal graph.

1.2 Query Languages

We study query evaluation for several *query languages* that are subsets of first-order logic (e.g., conjunctive queries) or of second-order logic (e.g., Datalog), without built-in relations. In this thesis, we only consider queries that are *constant-free*, and *Boolean*, so that an instance I either *satisfies* a query Q ($I \models Q$) or *violates* it ($I \not\models Q$). The *query evaluation problem* is the problem of determining if a given Boolean query is satisfied by a given relational instance. Its *combined complexity* is the complexity of the problem when both Q and I are given as input, whereas its *data complexity* is the complexity of the problem assuming that Q is fixed, and the only input is I (hence the data complexity may depend on Q). We recall that a constant-free Boolean query Q cannot differentiate between isomorphic instances, i.e., for any two isomorphic relational instances I and I', we have $I \models Q$ if and only if $I' \models Q$.

First-order queries. We refer to [Abiteboul, Hull, and Vianu 1995] for the semantics of first-order queries and for the definitions of basic notions such as existential and universal quantification, free and bound variables, etc. A class of queries that will play an important role in Chapter 2 is the class of *Boolean conjunctive queries*. A Boolean conjunctive query (CQ) Q on a relational signature σ with variables V is an existentially quantified conjunction of *atoms* with no free variables, where an atom is a fact on σ using variables from V as elements. Hence a Boolean CQ

can equivalently be seen as a relational instance, whose domain elements are the variables, and whose facts are the atoms. A σ-instance I satisfies Q when there exists a homomorphism from Q to I. A *Boolean conjunctive query with disequalities* ($\mathrm{CQ}^{\neq}$) on a signature σ is a Boolean conjunctive query on σ that can contain special *disequality atoms* of the form $x \neq y$. A σ-instance I satisfies the $\mathrm{CQ}^{\neq}$ Q when there exists a homomorphism h from the σ-atoms of Q to I such that for every disequality atom $x \neq y$ of Q, we have $h(x) \neq h(y)$.

A *Boolean union of conjunctive queries* (UCQ) is a disjunction of Boolean CQs, and a *Boolean union of conjunctive queries with disequalities* ($\mathrm{UCQ}^{\neq}$) is a disjunction of Boolean $\mathrm{CQ}^{\neq}$s. A σ-instance I satisfies a Boolean UCQ Q (resp., $\mathrm{UCQ}^{\neq}$) if Q contains a Boolean CQ (resp., $\mathrm{CQ}^{\neq}$) that is satisfied by I.

Stratified Datalog. In Chapter 3 we will work with the stratified Datalog query language, whose semantics we briefly recall here. We first recall the definition of (unstratified) Datalog. A *Datalog program* P (without negation) over relational signature σ (called the *extensional signature*) consists of:

- an *intensional signature* σ_{int} disjoint from σ (with the arity of σ_{int} being possibly greater than that of σ);
- a 0-ary *goal predicate* Goal in σ_{int};
- a set of *rules* of the form $R(\mathbf{x}) \leftarrow \psi(\mathbf{x}, \mathbf{y})$, where the *head* $R(\mathbf{x})$ is an atom with $R \in \sigma_{\text{int}}$, and the *body* ψ is a CQ over $\sigma_{\text{int}} \sqcup \sigma$ where each variable of $\mathbf{x}$ must occur.

The *semantics* $P(I)$ of P over an input σ-instance I is defined by a least fixpoint of the interpretation of σ_{int}: we start with $P(I) := I$, and for any rule $R(\mathbf{x}) \leftarrow \psi(\mathbf{x}, \mathbf{y})$ and tuple $\mathbf{a}$ of $\mathrm{dom}(I)$, when $P(I) \models \exists \mathbf{y}\, \psi(\mathbf{a}, \mathbf{y})$, then we *derive* the fact $R(\mathbf{a})$ and add it to $P(I)$, where we can then use it to derive more facts. We have $I \models P$ iff we derive the fact Goal(). The *arity* of P is $\max(\mathrm{arity}(\sigma), \mathrm{arity}(\sigma_{\text{int}}))$, and P is *monadic* if σ_{int} has arity 1.

Datalog with stratified negation [Abiteboul, Hull, and Vianu 1995] allows negated *intensional* atoms in bodies, but requires P to have a *stratification*, i.e., an ordered partition $P_1 \sqcup \cdots \sqcup P_n$ of the rules where:

(i) Each $R \in \sigma_{\text{int}}$ has a *stratum* $\zeta(R) \in \{1, \ldots, n\}$ such that all rules with R in the head are in $P_{\zeta(R)}$;

(ii) For any $1 \leqslant i \leqslant n$ and σ_{int}-atom $R(\mathbf{z})$ in a body of a rule of P_i, we have $\zeta(R) \leqslant i$;

(iii) For any $1 \leqslant i \leqslant n$ and negated σ_{int}-atom $R(\mathbf{z})$ in a body of P_i, we have $\zeta(R) < i$.

The stratification ensures that we can define the semantics of a stratified Datalog program by computing its interpretation for strata $P_1, \ldots, P_n$ in order: atoms in bodies always depend on a lower stratum, and negated atoms depend on strictly lower strata, whose interpretation was already fixed. We point out that, although a Datalog program with stratified negation can have many stratifications, all stratifications give rise to the same semantics [Abiteboul, Hull, and Vianu 1995, Theorem 15.2.10]. Hence, the semantics of P, as well as $I \models P$, are well-defined.

Example 1.2.1. The following stratified Datalog program, with $\sigma = \{R\}$ and $\sigma_{\text{int}} = \{T, \text{Goal}\}$, tests if there are two elements that are not connected by a directed R-path. The program has two strata, P_1 and P_2. The stratum P_1 contains the following two rules:

- $T(x, y) \leftarrow R(x, y)$
- $T(x, y) \leftarrow R(x, z) \wedge T(z, y)$

And P_2 contains the rule:

- $\text{Goal}() \leftarrow \neg T(x, y)$ ◁

Graph query languages. We will also consider in Chapter 3 some query classes that are specific to graph databases. We define here *two-way regular path queries* (2RPQs) and *conjunctions of 2RPQs* (C2RPQ) [Calvanese, De Giacomo, Lenzeniri, and Vardi 2000; Barceló 2013], two well-known query languages in the context of graph databases and knowledge bases.

We assume that the signature σ contains only binary relations. A (non-Boolean) *regular path query* (RPQ) $q_L(x, y)$ is defined by a regular language L on the alphabet Σ of the relation symbols of σ. Its semantics is that q_L has two free variables x and y, and $q_L(a, b)$ holds on an instance I for $a, b \in \text{dom}(I)$ precisely when there is a directed path π of relations of σ from a to b such that the label of π is in L. A *two-way regular path query* (2RPQ) is an RPQ on the alphabet $\Sigma^{\pm} := \Sigma \sqcup \{R^- \mid R \in \Sigma\}$, which holds whenever there is a path from a to b with label in L, with R^- meaning that we traverse an R-fact in the reverse direction. A C2RPQ $q = \bigwedge_{i=1}^{n} q_i(z_i, z_i')$ is a conjunction of 2RPQs, i.e., a conjunctive query made from atoms $q_i(z_i, z_i')$ that are 2RPQs (z_i and z_i' are not necessarily distinct). The *graph* of q is the unlabeled undirected graph having as vertices the variables of q and whose set of edges is $\{\{z_i, z_i'\} \mid 1 \leqslant i \leqslant n,\ z_i \neq z_i'\}$. A C2RPQ is *acyclic* if its graph is acyclic. A *strongly acyclic C2RPQ* (SAC2RPQ) is an acyclic C2RPQ that further satisfies: 1) for $1 \leqslant i \leqslant n$, we have $z_i \neq z_i'$ (no self-loops); and 2) for $1 \leqslant i < j \leqslant n$, we have $\{z_i, z_i'\} \neq \{z_j, z_j'\}$ (no multi-edges).

A *Boolean 2RPQ* (resp., *Boolean C2RPQ*) is a 2RPQ (resp., C2RPQ) which is existentially quantified on all its free variables.

We can express Boolean 2RPQs and C2RPQs as Datalog queries; for instance see Proposition 3.4.16.

1.3 Tuple-Independent Databases

As we already mentioned in the introduction, one way to work with uncertain data would be to explicitly represent all possible states of the data (called the *possible worlds*) and associate to each world a probability value. The probability that a Boolean query satisfies a probabilistic database would then be the sum of the probabilities of the possible worlds that satisfy the query. The problem with this approach is that this is not a very compact way of storing the data, so that it is not feasible in practice to represent the data and query it in this way. Nevertheless, we can efficiently *represent* uncertain data if we make some independence assumptions: each tuple has a probability to be present or absent independently of the other tuples.

Likes		Prob.
Bob	tiramisu	0.9
Mary	tiramisu	0.2
Mary	flapjack	0.5
Tom	meringue	0.6

Table 1.2 – Dessert preferences

Tuple-independent databases. We will consider in this thesis the most commonly used model for probabilistic databases: the *tuple-independent* model, where each tuple is annotated with a probability of being present or absent, assuming independence across tuples. Formally, a *tuple-independent database* (TID) is a pair (I, π) consisting of a relational instance I and a function π mapping each tuple $t \in I$ to a rational probability $\pi(t) \in [0; 1]$. A TID instance (I, π) defines a probability distribution Pr on $I' \subseteq I$, defined by

$$\Pr(I') := \prod_{t \in I'} \pi(t) \times \prod_{t \in I \setminus I'} (1 - \pi(t)).$$

Given a Boolean query Q and a TID instance (I, π), the probability that Q is satisfied by (I, π) is then defined as

$$\Pr((I, \pi) \models Q) := \sum_{I' \subseteq I \text{ s.t. } I' \models Q} \Pr(I').$$

The *probabilistic query evaluation problem* (PQE) asks, given a Boolean query Q and a TID instance (I, π), the probability that Q is satisfied by (I, π). As in the case of query evaluation, the combined complexity of PQE is the complexity of PQE when both Q and (I, π) are given as input, whereas its *data complexity* is the complexity of the problem assuming that Q is fixed, and the only input is (I, π) (hence the data complexity may depend on Q).

Example 1.3.1. An example of TID instance is given in Table 1.2, describing who likes which dessert. A possible world of this instance would be that Mary likes flapjacks, Tom likes meringue and the other facts are absent. This possible world has probability $(1 - 0.9) \times (1 - 0.2) \times 0.5 \times 0.6 = 0.024$. An interesting query on this TID would be: what is the probability that there exist two different people liking the same dessert? This can be expressed as the probability of the Boolean conjunctive query with disequalities $\exists p_1 p_2 d \, \text{Likes}(p_1, d) \wedge \text{Likes}(p_2, d) \wedge p_1 \neq p_2$. The answer is the sum of the probabilities of the possible worlds in which the query is true, which can easily seen to be $0.9 \times 0.2 = 0.18$. ◁

One weakness of the TID model is that is cannot represent arbitrary probability distributions. For instance, there is no TID instance for which there is a possible world where Tom likes meringue and for which in every possible world where Tom likes meringue, Bob also likes meringue. However, as we will soon see in Chapter 2, PQE on TID instances is already quite challenging. More elaborate probabilistic representation systems [Barbará, Garcia-Molina, and Porter 1992; Ré and Suciu 2007; Green and Tannen 2006; Huang, Antova, Koch, and Olteanu 2009; Suciu, Olteanu, Ré, and Koch 2011] exist (*block-independent databases* (BID), *probabilistic c-tables* (pc-tables), etc.).

1.4 Complexity Classes

In this thesis, the upper bounds on running times are given assuming the RAM model [Aho and Hopcroft 1974]. This detail is not important when we simply state that a problem can be solved in polynomial time (PTIME), but it has its importance in many places where we want to obtain linear-time algorithms. We use the RAM model instead of Turing machines as the former is closer to modern computer architecture than the latter. We will also assume that arithmetic operations are performed in unit time. This hypothesis will be important when we work with probabilities in order to obtain linear-time complexity. If we do not make this hypothesis then the complexities become polynomial-time.

Although most problems we are interested in are *function problems*, that is problems that compute a certain output given an input, in contrast with *decision problems* that simply accept or reject an input, we will not insist on the difference and will often simply write that a problem "is in PTIME", instead of writing "is in P" or "is in FP".

Besides "classical" complexity classes (e.g., P, NP, EXPTIME, etc.), two arguably less common complexity classes will be of particular interest to us: the class of counting problems #P and the class of *fixed-parameter tractable* problems. We briefly introduce them here.

Counting complexity classes. A *counting problem* is a problem whose goal is to *count things*. For example, we might want to count the number of satisfying valuations (see Section 1.7) of a Boolean formula φ, which is the problem known as #SAT, or we might want to count the number of subinstances of a database that satisfy a given Boolean conjunctive query. #P (pronounced "sharp P") is the class of *counting problems* that is the counting analog of the decision complexity class NP. It was introduced in 1979 by Valiant to study counting complexity classes [Valiant 1979]. A counting problem is in #P if it can be expressed as the number of accepting paths of a nondeterministic polynomial-time Turing machine. For example, one can easily see that #SAT is in #P: the machine first guesses a valuation and then checks and accepts in PTIME if φ is true under this valuation. In fact, #SAT is even #P-complete (we will talk more about this in Section 1.7). We use here the notion of polynomial-time Turing reduction (with an oracle to the problem we reduce to), which is standard when working with #P.

In a database context, one can also see that counting the number of subinstances of a database that satisfy a fixed Boolean CQ Q is in #P: the machine first guesses a subinstance and then checks and accepts in PTIME if this subinstance satisfies Q. Imagine now that we want to compute the *probability* that a TID (I, π) satisfies the fixed Boolean CQ Q. This is not a counting problem since the answer is generally a rational, not an integer. Hence, technically speaking, we cannot say that this problem is in #P. However we can easily see that it is in $\mathrm{FP}^{\#\mathrm{P}}$ (i.e., solvable by a polynomial-time Turing machine with access to oracles in #P). Informally, the machine first modifies the rational probabilities of the tuples so that they all have the same denominator d. The machine then uses its oracle tape to call in unit time another machine that computes a #P problem. This other machine guesses a subinstance and checks if this subinstance satisfies the query in PTIME. If it is the case, then it nondeterministically creates a number of runs that equals the

integer weight of the subinstance (the product of the nominators of the modified probabilities of the tuples in the subinstance). The first machine then uses the number of accepting paths of the oracle machine to compute the final probability: it simply has to divide this number of accepting paths by d. Although probability computation problems are not counting problems, when proving the hardness of probabilistic query evaluation problems (mainly in Chapter 2), we will always use the notion of #P-hardness, i.e., we reduce from #P-complete problems, such as #SAT.

Parameterized complexity classes. *Parameterized complexity theory* [Flum and Grohe 2006] is a branch of complexity theory introduced in the nineties by Downey and Fellows [Downey and Fellows 1992] to better understand the complexity of a problem depending on some input parameter. The practical motivation is that, sometimes, some natural parameter of the input is small and one would like to use that fact to derive faster algorithms. Indeed, the complexity of the problem can depend in various ways on a given parameter, as illustrated by the following two simple problems:

Example 1.4.1. Our first example comes from the study of the evaluation of Boolean conjunctive queries on relational databases. Here, a natural parameter can be the size of the conjunctive query. A trivial brute-force algorithm can solve this problem in time $O(|I|^{|Q|})$: for each variable of Q test all possible assignments to an element of I and check if the mapping produced is an homomorphism. We see that the role of the query and that of the database are not the same in the complexity of this algorithm. Another example is the following problem: given an undirected unlabeled graph G, find a set C of nodes of G that *covers* G (i.e., for every edge e of G we have $e \cap C \neq \emptyset$). Imagine that we want to decide if there exist such a covering set of size at most k, for some small $k \in \mathbb{N}$. For instance we might want to cover the corridors of a museum with k security cameras, ensuring that each corridor can be seen by at least one camera. One can solve this problem in time $O(2^k \times |G|)$ (using a bounded search tree algorithm), and again we see that the role of the parameter k and that of $|G|$ are different. Moreover in these two examples, even though for some fixed value of the parameter the problem is PTIME, the parameter ($|Q|$ in the first example and k in the second) does not interact in the same way with the input (I and G). For big enough databases and small parameters, we see that a constant factor of 2^k is better than an exponent of $|I|^{|Q|}$. ◁

Thus, parameterized complexity theory can intuitively be understood as the study of how nicely (or badly) parameters influence the complexity of the problem. A typical "nice" case would correspond to the notion of *fixed-parameter tractability* (FPT): a problem on input I parameterized by k is *fixed-parameter tractable* (or just FPT) if its running time is of the form $O(f(k) \times |I|^c)$ for some computable function f and integer $c \in \mathbb{N}$. Observe that calling the problem FPT is more informative than saying that it is in PTIME for fixed k, as we are further imposing that the polynomial degree c does not depend on k. This follows the distinction in parameterized complexity between FPT and the class XP of parameterized problems which are PTIME for each fixed value of k. We will say that a problem is *FPT-linear* when the exponent c is equal to 1. Hence, the second problem from our example is FPT-linear when parameterized by k.

In this thesis, we will simply use parameterized complexity theory as a definitional framework to state asymptotic running times, even though it can be used for more, such as showing that some problems are not likely to be FPT (for instance, the evaluation of a Boolean CQ on a database is unlikely to be FPT when parameterized by the size of the query); see [Flum and Grohe 2006].

FPT-bilinearity. In Chapter 3 we study the query evaluation problem for a query class $\mathcal{Q}$ and instance class $\mathcal{I}$: given an instance $I \in \mathcal{I}$ and query $q \in \mathcal{Q}$, check if $I \models q$. More precisely, we will study cases where $\mathcal{I}$ and $\mathcal{Q}$ are parameterized: given infinite sequences $\mathcal{I}_1, \mathcal{I}_2, \ldots$ and $\mathcal{Q}_1, \mathcal{Q}_2, \ldots$, the *query evaluation problem parameterized by* k_{I}, k_{q} applies to $\mathcal{I}_{k_{\mathrm{I}}}$ and $\mathcal{Q}_{k_{\mathrm{q}}}$. We will call the parameterized problem *FPT-bilinear* if there is a computable function f such that the problem can be solved with combined complexity $O\left(f(k_{\mathrm{I}}, k_{\mathrm{Q}}) \cdot |I| \times |q|\right)$.

1.5 Trees, Treewidth, Pathwidth

Trees. We will consider various forms of trees in the thesis. Most generally, a *tree* T is a finite undirected unlabeled graph that has no cycles and that is connected (i.e., there exists exactly one path between any two different nodes). We will often abuse notation and write $n \in T$ to mean that n is a node of T. A tree T is *rooted* if it has a distinguished node r called the *root* of T. Given two adjacent nodes n_1, n_2 of a rooted tree T with root r, if n_1 lies in the (unique) path from r to n_2, we say that n_1 is the *parent* of n_2 and that n_2 is a *child* of n_2. Hence a node n can have many children but at most one parent (and none if n is the root). A node n' is a *descendant* of a node n in a rooted tree if $n \neq n'$ and n lies on the path from n' to the root. A *leaf* of T is a node that has no children, and an *internal node* of T is a node that is not a leaf. A rooted tree is *binary* if all nodes have at most two children, and it is *full* if all nodes are leaves or have exactly two children. A rooted tree is *ordered* if the children of any node n follow a certain linear order. In the case of a rooted ordered binary full tree, we often refer to the first child of an internal node as its *left child*, and to the second child as its *right child*.

Given a rooted tree T and a node $n \in T$, *the subtree of* T *rooted at* n, denoted T_n, is the rooted tree with root n whose nodes are n and all the descendant of n in T, and where two nodes are adjacent if they are adjacent in T. In particular, if n is the root of T, then we have $T_n = T$. For a subtree U of T, we denote by $\mathsf{Leaves}(U)$ the leaves of U.

We will also work with trees whose nodes are labeled. Letting Γ be a set of *labels*, a Γ-*tree* $\langle T, \lambda \rangle$ consists of a tree T together with a labeling function λ that associates to each node of T a label in Γ.

Treewidth. Treewidth [Robertson and Seymour 1984] is a measure quantifying how far a graph is to being a tree, which we use to restrict instances and conjunctive queries. For example a tree has treewidth 1, a cycle has treewidth 2, and a k-clique has treewidth $k - 1$. Though historically treewidth was first defined for graphs and the definition was later extended to hypergraphs, we choose to directly give the definition for hypergraphs, as it will be more convenient for us. Treewidth can be defined by using the notion of *tree decomposition* of a hypergraph. A tree

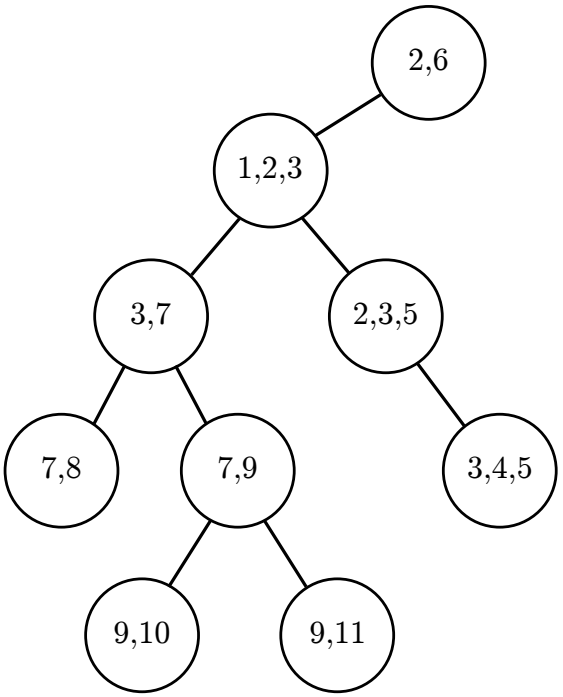

Figure 1.2 – Tree decomposition of the hypergraph from Example 1.1.2.

decomposition of the hypergraph $\mathcal{H} = (V, E)$ is a tree T, whose nodes b (called *bags*) are labeled by a subset $\lambda(b)$ of V (i.e., a 2^V-tree $\langle T, \lambda \rangle$) and which satisfies:

(i) for every hyperedge $e \in E$, there is a bag $b \in T$ with $e \subseteq \lambda(b)$;

(ii) for all $v \in V$, the set of bags $\{b \in T \mid v \in \lambda(b)\}$ is a connected subtree of T.

We will often call $\lambda(b)$ the *domain* of the bag b, and we even sometimes make no difference between b itself and its domain. The *width* of a tree decomposition T is the size of its largest bag minus one, i.e., $\max_{b \in T} |\lambda(b)| - 1$. The *treewidth* of $\mathcal{H}$ is then the smallest k such that I has a tree decomposition of width k.

The treewidth of a graph G is the treewidth of G when seen as a hypergraph, and the treewidth of a relational instance I is the treewidth of its associated hypergraph.

Example 1.5.1. Figure 1.2 shows a tree decomposition of the hypergraph $\mathcal{H}$ from Example 1.1.2 (hence a tree decomposition of the instance in Example 1.1.1). The width of this tree decomposition is 2. Moreover, the width of any tree decomposition of $\mathcal{H}$ is at least 2, since the hyperedge $\{1, 2, 3\}$ must be contained in a bag. Hence, the treewidth of $\mathcal{H}$ is 2. ◁

It is known that the treewidth of an instance is the same as the treewidth of its primal graph.

The definition of treewidth might seem complex when first encountered, but bounding the treewidth it is a well-known natural criterion to ensure the tractability of many problems that are NP-hard on arbitrary instances. An intuitive way to understand tree decompositions is that they decompose the instance in such a way as to be able to use divide and conquer algorithms. When S is a set of relational instances (or hypergraphs, or graphs), we say that S is *treelike* if there exists $k \in \mathbb{N}$ such that the treewidth of each element in S is less than k.

It is NP-hard to determine the treewidth of a hypergraph [Arnborg, Corneil, and Proskurowski 1987], but we can compute a tree decomposition in linear time when parameterizing by the treewidth:

Theorem 1.5.2 ([Bodlaender 1996])**.** *Given a hypergraph $\mathcal{H}$ and an integer $k \in \mathbb{N}$ we can check in FPT-linear time parameterized by k if $\mathcal{H}$ has treewidth $\leqslant k$, and if yes output a tree decomposition of $\mathcal{H}$ of width $\leqslant k$.*

Pathwidth. *Pathwidth* is defined in the same way as treewidth, except that the decomposition is restricted to be a path (i.e., each node has at most one child). It intuitively measures how far an instance is to being a path.

1.6 Tree Automata and Tree Encodings

Bottom-up tree automata. One of the simplest formalisms for tree automata is that of *bottom-up nondeterministic tree automata*. A bottom-up nondeterministic tree automaton running on rooted, full, binary and ordered Γ-trees (or Γ-bNTA) is a tuple $A = (Q, F, \iota, \Delta)$, where:

(i) Q is a finite set of *states*;

(ii) $F \subseteq Q$ is a subset of *accepting states*;

(iii) $\iota : \Gamma \to 2^Q$ is an *initialization function* determining the possible states of a leaf from its label;

(iv) $\Delta : \Gamma \times Q^2 \to 2^Q$ is a *transition function* determining the possible states of an internal node from its label and the states of its two children.

Given a (rooted, full, binary and ordered) Γ-tree $\langle T, \lambda \rangle$, we define a *run* of A on $\langle T, \lambda \rangle$ as a function $\varphi : T \to Q$ such that:

(i) $\varphi(l) \in \iota(\lambda(l))$ for every leaf l of T;

(ii) $\varphi(n) \in \Delta(\lambda(n), \varphi(n_1), \varphi(n_2))$ for every internal node n of T with left child n_1 and right child n_2.

The bNTA A *accepts* $\langle T, \lambda \rangle$ if it has a run on T mapping the root of T to a state of F.

Tree encodings. Throughout this thesis, we will often show that some problem is tractable on treelike instances using the following scheme:

1. Build a tree automaton A running on tree decompositions of width $\leqslant k$ (for k the treewidth bound) such that for every instance I and tree decomposition $\langle T, \lambda \rangle$ of width $\leqslant k$ of I, A accepts $\langle T, \lambda \rangle$ iff I satisfies the problem;
2. Use Theorem 1.5.2 to compute a tree decomposition $\langle T, \lambda \rangle$ of width $\leqslant k$ of I;
3. Check if $\langle T, \lambda \rangle$ is accepted by A.

However, this scheme does not work as-is: the labels of a tree decomposition are not from a finite alphabet (because we do not know $\mathrm{dom}(I)$ in advance), whereas tree automata run on trees labeled by a finite alphabet. The notion of *tree encoding* is here to address this issue. Intuitively, a tree encoding is just a tree decomposition encoded with a finite alphabet so as to be processable by a tree automaton. There are many ways to design such an encoding. We present here the tree encodings used in [Amarilli, Bourhis, and Senellart 2015].

Informally, having fixed the signature σ, for a fixed treewidth $k \in \mathbb{N}$, we define a finite tree alphabet Γ_σ^k such that σ-instances of treewidth $\leqslant k$ can be translated in FPT-linear time (parameterized by k), following the structure of a tree decomposition, to a (rooted full ordered binary) Γ_σ^k*-tree*, which we call a *tree encoding*. Formally:

Definition 1.6.1. Let σ be a signature, and let $k \in \mathbb{N}$. We define the domain $\mathcal{D}_k = \{a_1, \ldots, a_{2k+2}\}$ and the finite alphabet Γ_σ^k whose elements are pairs (d, s), with d being a subset of up to $k+1$ elements of $\mathcal{D}_k$, and s being a σ-instance consisting at most one σ-fact over some subset of d (i.e., $\mathrm{dom}(s) \subseteq d$): in the latter case, we will abuse notation and identify s with the one fact that it contains. A (σ, k)*-tree encoding* is simply a rooted, binary, ordered, full Γ_σ^k-tree $\langle E, \lambda \rangle$. ◁

The fact that $\langle E, \lambda \rangle$ is rooted and ordered is merely for technical convenience when running bNTAs, but it is otherwise inessential.

Example 1.6.2. The tree depicted in black in Figure 1.3 is a $(\{R, S, T\}, 2)$-tree encoding. For now, ignore the annotations in red and green; the link with Example 1.1.1 will be explained later. The domain $\mathcal{D}_2$ is $\{a, b, c, d, e, f\}$, but we only use $\{a, b, c, d\}$. ◁

A tree encoding $\langle E, \lambda \rangle$ can be decoded to an instance I with the elements of $\mathcal{D}_k$ being decoded to new instance elements. Informally, we create a fresh instance element for each occurrence of an element $a_i \in \mathcal{D}_k$ in an a_i*-connected subtree of* E, i.e., a maximal connected subtree where a_i appears in the first component of the label of each node. In other words, reusing the same a_i in adjacent nodes in $\langle E, \lambda \rangle$ means that they stand for the same element, and using a_i elsewhere in the tree creates a new element. Formally:

Definition 1.6.3. Let $\langle E, \lambda \rangle$ be a (σ, k)-tree encoding, where, for each node n of E, we write $\lambda(n) = (d_n, s_n)$. A set S of *bag decoding functions for* $\langle E, \lambda \rangle$ consists of one function dec_n with domain d_n for every node n of E. We say that S is *valid* if S satisfies the following condition: for every $a \in \mathcal{D}_k$ and nodes n_1, n_2 of E such that $a \in d_{n_1}$ and $a \in d_{n_2}$, we have $\mathrm{dec}_{n_1}(a) = \mathrm{dec}_{n_2}(a)$ if and only if n_1 and n_2 are in the same a-connected subtree of $\langle E, \lambda \rangle$. ◁

Example 1.6.4. Consider again the tree encoding $\langle E, \lambda \rangle$ in Figure 1.3. For each node $n \in E$, let dec_n be the function that is defined by the green annotations next to n. Then one can check that $S = \{\mathrm{dec}_n \mid n \in E\}$ is a valid set of bag decoding functions for $\langle E, \lambda \rangle$. ◁

We can use a valid set of bag decoding functions S to decode a tree encoding $\langle E, \lambda \rangle$ to a σ-instance:

Definition 1.6.5. Let $\langle E, \lambda \rangle$ be a (σ, k)-tree encoding, where, for each node n of E, we write $\lambda(n) = (d_n, s_n)$, and let S be a valid set of bag decoding functions for $\langle E, \lambda \rangle$. The σ-instance $\mathrm{dec}_S(\langle E, \lambda \rangle)$ is defined as follows. The elements of $\mathrm{dec}_S(\langle E, \lambda \rangle)$ are $\{\mathrm{dec}_n(a) \mid n \in E,\ a \in d_n\}$. The facts of $\mathrm{dec}_S(\langle E, \lambda \rangle)$ are $\{R(\mathrm{dec}_n(\mathbf{a})) \mid n \in E,\ s_n = R(\mathbf{a})\}$. ◁

Example 1.6.6. Continuing Example 1.6.4, consider again the tree encoding $\langle E, \lambda \rangle$ and valid set S of decoding functions for $\langle E, \lambda \rangle$. Then, computing $\mathrm{dec}_S(\langle E, \lambda \rangle)$ yields the instance from Example 1.1.1. ◁

A tree encoding can have multiple valid sets S of bag decoding functions. However, the choice of S does not matter since they all decode to isomorphic instances:

Lemma 1.6.7 ([Amarilli, Bourhis, and Senellart 2015])**.** *Let $\langle E, \lambda \rangle$ be a tree encoding, and S_1, S_2 be two valid sets of bag decoding functions of $\langle E, \lambda \rangle$. Then $\mathrm{dec}_{S_1}(\langle E, \lambda \rangle)$ and $\mathrm{dec}_{S_2}(\langle E, \lambda \rangle)$ are isomorphic.*

Hence, we will now write $\mathrm{dec}(\langle E, \lambda \rangle)$, forgetting the subscript S, since we are not interested in distinguishing isomorphic instances (and since there always exists at least one valid set of bag decoding functions, for every tree encoding). Furthermore, it is easy to see that $\mathrm{dec}(\langle E, \lambda \rangle)$ has treewidth $\leqslant k$, as a tree decomposition for it can be constructed from $\langle E, \lambda \rangle$. Conversely, for any instance I of treewidth $\leqslant k$, we can compute a (σ, k)-encoding $\langle E, \lambda \rangle$ such that $\mathrm{dec}(\langle E, \lambda \rangle)$ is I (up to isomorphism). We say that $\langle E, \lambda \rangle$ *is a tree encoding of* I:

Definition 1.6.8. Let I be a σ-instance, and $\langle E, \lambda \rangle$ be a (σ, k)-tree encoding (for some $k \in \mathbb{N}$). We say that $\langle E, \lambda \rangle$ *is a tree encoding of* I if $\mathrm{dec}(\langle E, \lambda \rangle)$ is I, up to isomorphism. ◁

Informally, given I of treewidth $\leqslant k$, we can construct a (σ, k)-tree encoding $\langle E, \lambda \rangle$ of I from a tree decomposition of I as follows: copy each bag of the decomposition multiple times so that each fact can be coded in a separate node; arrange these copies in a binary tree to make the tree encoding binary; make the tree encoding full by adding empty nodes. We can easily show that this process is FPT-linear for k, so that we will use the following claim (see [Amarilli 2016] for our type of encodings):

Lemma 1.6.9 ([Flum, Frick, and Grohe 2002])**.** *The problem, given an instance I of treewidth $\leqslant k$, of computing a tree encoding of I, is FPT-linear parameterized by k.*

We sum up this discussion about tree encoding with the full example.

Example 1.6.10. Remember that Figure 1.3 presents a tree encoding $\langle E, \lambda \rangle$ for $k = 2$ and the signature σ from Example 1.1.1. Remember we can decode $\langle E, \lambda \rangle$ by the mappings drawn in green, obtaining this way the instance I from Example 1.1.1. Hence $\langle E, \lambda \rangle$ is a tree encoding of I. Moreover, we recall that any valid way of decoding $\langle E, \lambda \rangle$ would yield an instance isomorphic to I. We point out a few details that can help understand how these encodings work. The elements "a" in bags α and β are decoded to distinct instance elements, since α and β are not in a same a-connected subtree of the tree encoding. Bags like γ, that contain elements but no fact, are usually used in order to help making the tree encoding binary. Empty bags like δ can be used in order to make the tree encoding full. ◁

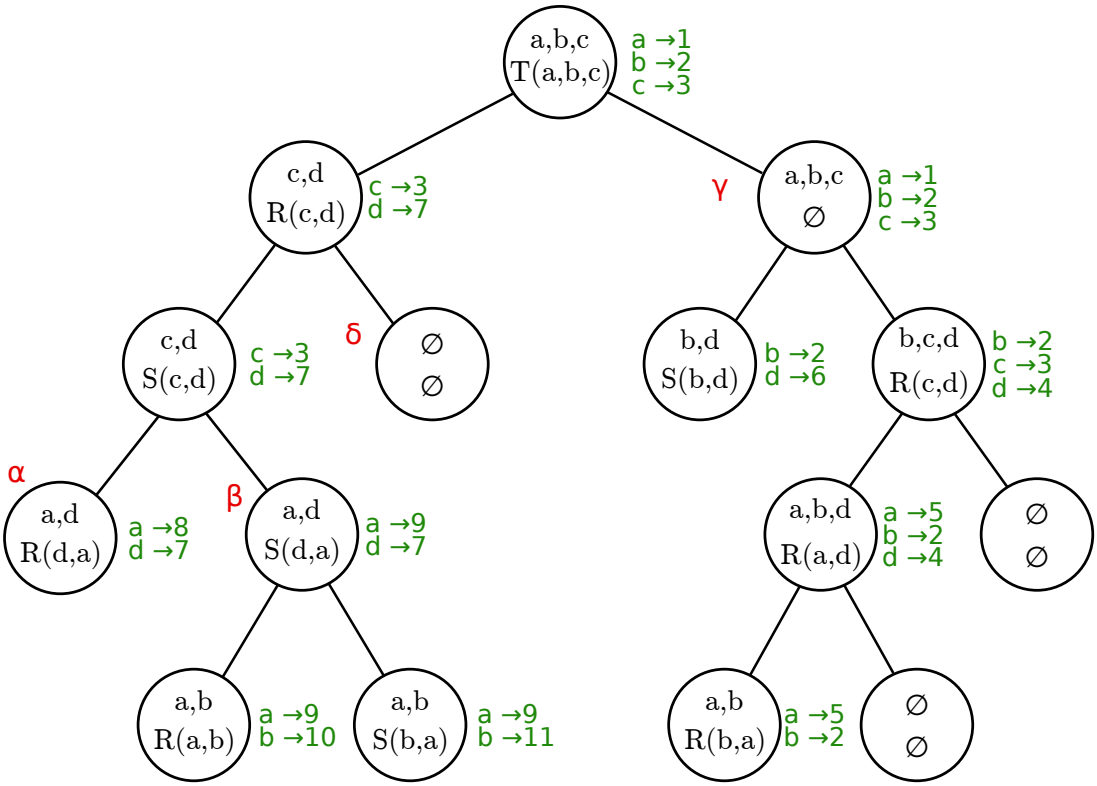

Figure 1.3 – Tree encoding of the relational instance from Example 1.1.1.

1.7 Provenance and Knowledge Compilation Circuit Classes

Boolean circuits and functions. A (Boolean) *valuation* of a set V is a function $\nu : V \to \{0, 1\}$. A *Boolean function* φ on variables V is a mapping that associates to each valuation ν of V a Boolean value in $\{0, 1\}$ called the *evaluation* of φ according to ν.

A *(Boolean) circuit* $C = (G, W, g_{\mathsf{output}}, \mu)$ is a directed acyclic graph (G, W) whose vertices G are called *gates*, whose edges W are called *wires*, where $g_{\mathsf{output}} \in G$ is the *output gate*, and where each gate $g \in G$ has a *type* $\mu(g)$ among var (a *variable gate*), NOT, OR, AND. The *inputs* of a gate $g \in G$ are the gates $g' \in G$ such that $(g', g) \in W$; the *fan-in* of g is its number of inputs. We require NOT-gates to have fan-in 1 and var-gates to have fan-in 0. The *treewidth* of C, and its *size*, are those of the graph (G, W). The set C_{var} of *variable gates* of C are those of type var. Given a valuation ν of C_{var}, we extend it to an *evaluation* of C by mapping each variable $g \in C_{\mathsf{var}}$ to $\nu(g)$, and evaluating the other gates according to their type. We recall the convention that AND-gates (resp., OR-gates) with no input evaluate to 1 (resp., 0). The Boolean function on C_{var} *captured* by the circuit is the one that maps ν to the evaluation of g_{output} under ν. Two circuits are *equivalent* if they capture the same function.

Provenance and lineages. Provenance is a handy tool to solve query evaluation problems where the answer is more complex than *yes* or *no*. Intuitively, the provenance of a query Q on a relational database I is a representation of how the query result

depends on the input data:

The (Boolean) *provenance* of a query Q on an instance I is the Boolean function φ whose variables are the facts of I, which is defined as follows: for any valuation ν of the facts of I, we have $\varphi(\nu) = 1$ iff the subinstance $\{F \in I \mid \nu(F) = 1\}$ satisfies Q.

Hence, provenance can help explain the query results. We can represent Boolean provenance as Boolean formulas [Imielinski and Lipski 1984; Green, Karvounarakis, and Tannen 2007], or (more recently) as Boolean circuits [Deutch, Milo, Roy, and Tannen 2014; Amarilli, Bourhis, and Senellart 2015]. The provenance of Q on I is also sometimes called the *lineage* of Q on I. In this thesis, we will mainly be interested in provenance for its role in *intensional query evaluation*, i.e., an intermediate object that can be constructed during the process of solving PQE to help us compute the final probability.

Probability of a Boolean function. Given a set of variables V and a *probability assignment* π mapping each variable X in V to a rational probability $\pi(X) \in [0, 1]$, we define the *probability* $\pi(\nu)$ of a valuation $\nu : V \to \{0, 1\}$ as

$$\pi(\nu) := \left(\prod_{X \in V,\ \nu(X)=1} \pi(X) \right) \left(\prod_{X \in V,\ \nu(X)=0} (1 - \pi(X)) \right).$$

The *probability* $\mathrm{Pr}(\varphi, \pi)$ *of Boolean function* φ *on variables* V *with probability assignment* π is then the total probability of the valuations that satisfy φ. Formally:

$$\mathrm{Pr}(\varphi, \pi) := \sum_{\nu \text{ satisfies } \varphi} \pi(\nu)$$

Provenance can then be used in PQE to obtain the probability of the query. Indeed, it is clear from the definitions that, given a TID (I, π) and Boolean query Q, if φ is the provenance of Q on I, then the probability of φ with probability assignment π is precisely $\mathrm{Pr}((I, \pi) \models Q)$. This is the *intensional approach* to PQE.

Unfortunately, in general, it is intractable to compute the probability of a Boolean function represented as a formula or circuit. Specifically, we define the *Boolean formula probability computation problem* (resp., *Boolean circuit probability computation problem*) to be the following: the input is a Boolean formula (resp., a Boolean circuit) on a set of variables V together with a probability assignment π, and the output is the probability of the Boolean function represented by the formula (resp., circuit). Then, both these problems are #P-hard, as they generalize the problem #SAT: choosing π so that every variable is mapped to probability $\frac{1}{2}$ we see that the number of satisfying valuations of φ is exactly $2^{|V|} \times \mathrm{Pr}(\varphi, \pi)$. To ensure that computing this probability is tractable, we can sometimes use the structure of the query and data to represent the lineage in tractable formalisms from the field of *knowledge compilation*.

Knowledge compilation circuit classes. We now introduce a few circuit classes from knowledge compilation. For a gate g in a Boolean circuit C, we write $\mathsf{Vars}(g)$ for the set of variable gates of C_{var} that have a directed path to g in C. An AND-gate g of C is *decomposable* if for every two input gates $g_1 \neq g_2$ of g we have $\mathsf{Vars}(g_1) \cap \mathsf{Vars}(g_2) = \emptyset$. We call C *decomposable* if each AND-gate is. An OR-gate g of C is *deterministic* if there is no pair $g_1 \neq g_2$ of input gates of g and valuation

ν of C_{var} such that g_1 and g_2 both evaluate to 1 under ν. A Boolean circuit is *deterministic* if each OR-gate is.

Restricting to *deterministic decomposable Boolean circuits* (d-Ds) is already sufficient to ensure that probability computation can be performed efficiently (specifically, in linear time in the input circuit, as we assumed that arithmetic operations take unit time). Indeed, one can simply transform the d-D into an arithmetic circuit by converting AND gates to $\times$, OR gates to $+$, NOT to $1 - x$ (where x is the value of the only input of the gate), and variable gates by the value mapped by π. One can then evaluate in linear time this arithmetic circuit (by a bottom-up pass), and the determinism and decomposability restrictions ensure that we are indeed computing the probability of the Boolean function that the circuit captures.

In addition to decomposability and determinism, knowledge compilation also studies more restricted versions of Boolean circuits. The most common restriction is that of being a *negation normal form* (NNF). A Boolean circuit C is in negation normal form if the inputs of NOT-gates are always variable gates. A stronger requirement than decomposability is *structuredness*. A *v-tree* [Pipatsrisawat and Darwiche 2008] over a set V is a rooted ordered full binary tree T together with an injective mapping ι from V to $\mathsf{Leaves}(T)$. For simplicity, for every variable $v \in V$, we identify the leaf l of T such that $\iota(v) = l$ with the variable v, and we call a leaf l of T *unlabelled* when l does not correspond to a variable. We say that T *structures* a Boolean circuit C (and call it a *v-tree for* C) if T is over the set C_{var} and if, for every AND-gate g of C with inputs $g_1, \ldots, g_m$ and $m > 0$, there is a node $n \in T$ that *structures* g, i.e., n has m children $n_1, \ldots, n_m$ and we have $\mathsf{Vars}(g_i) \subseteq \mathsf{Leaves}(T_{n_i})$ for all $1 \leqslant i \leqslant m$. We call C *structured* if some v-tree structures it. Note that structured Boolean circuits are always decomposable, and their AND-gates have at most two inputs because T is binary.

The main structured class of circuits that we study in Chapter 4 are *deterministic structured decomposable NNFs*, which we denote d-SDNNF for brevity as in [Pipatsrisawat and Darwiche 2008]. One can use d-SDNNFs to provide efficient enumeration algorithms [Amarilli, Bourhis, Jachiet, and Mengel 2017].

Note that decomposability and structuredness are syntactic conditions, whereas determinism is a semantic condition: in general, checking if an OR gate of a Boolean circuit is deterministic is co-NP complete.

Knowledge compilation also studies a different kind of tractable formalisms for Boolean functions, called *binary decision diagrams*. We define here *ordered binary decision diagrams* (OBDDs). An OBDD on a set of variables $V = \{v_1, \ldots, v_n\}$ is a rooted DAG O whose leaves are labeled by 0 or 1, and whose internal nodes are labeled with a variable of V and have two outgoing edges labeled 0 and 1. We require that there exists a total order $\mathbf{v} = v_{i_1}, \ldots, v_{i_n}$ on the variables such that, for every path from the root to a leaf, the sequence of the variables that label the internal nodes of the path is a subsequence of $\mathbf{v}$ and does not contain duplicate variables. The OBDD O *captures* a Boolean function on V defined by mapping each valuation ν to the value of the leaf reached from the root by following the path given by ν. The *size* $|O|$ of O is its number of nodes, and the *width* w of O is the maximum number of nodes at every *level*, where a level is defined for a prefix of $\mathbf{v}$ as the set of nodes reached by enumerating all possible valuations of this prefix. Note that we clearly have $|O| \leqslant |V| \times w$.

Notice that an OBDD can be seen as a restricted kind of d-SDNNF.

DNFs and CNFs. We also study other representations of Boolean functions, namely, Boolean formulas in *conjunctive normal form* (*CNFs*) and in *disjunctive normal form* (*DNFs*). A DNF (resp., CNF) φ on a set of variables V is a disjunction (resp., conjunction) of *clauses*, each of which is a conjunction (resp., disjunction) of *literals* on V, i.e., variables of V (a *positive* literal) or their negation (a *negative* literal). A *monotone* (or *positive*) DNF (resp., monotone/positive CNF) is one where all literals are positive, in which case we often identify a clause to the set of variables that it contains. Monotone DNFs and CNFs φ are isomorphic to hypergraphs: the vertices are the variables of φ, and the hyperedges are the clauses of φ. We often identify φ to its hypergraph. In particular, the *pathwidth* and *treewidth* of φ, and its *arity* (arity(φ)) and *degree* (degreeφ), are defined as that of its hypergraph.

Chapter 2

Limits of Combined Tractability of PQE

This chapter of my thesis presents my work with Antoine Amarilli and Pierre Senellart on the combined complexity of probabilistic query evaluation (PQE) phrased as a probabilistic graph homomorphism problem. The results presented here were published at PODS'2017 [Amarilli, Monet, and Senellart 2017].

2.1 Introduction

This chapter begins our study of the combined complexity of PQE, focusing on the case of conjunctive queries on tuple-independent databases. As we have already discussed in the introduction, almost all works on PQE so far have focused on data complexity, and have explored the general intractability of PQE in this sense. Indeed, while non-probabilistic query evaluation of fixed queries in first-order logic has polynomial-time data complexity (specifically, AC^0 [Abiteboul, Hull, and Vianu 1995]), the PQE problem is #P-hard already for some fixed conjunctive queries [Dalvi and Suciu 2007]. Specifically, Dalvi and Suciu have shown a celebrated dichotomy on unions of conjunctive queries: some are *safe queries*, enjoying PTIME data complexity (specifically, linear [Ceylan, Darwiche, and Van den Broeck 2016]), and all other queries are #P-hard [Dalvi and Suciu 2012]. In another direction, Amarilli, Bourhis, and Senellart have shown a dichotomy on instance families for fixed monadic second-order queries, with tractable data complexity for bounded-treewidth families [Amarilli, Bourhis, and Senellart 2015], and intractability otherwise under some assumptions [Amarilli, Bourhis, and Senellart 2016].

However, even when PQE is tractable in data complexity, the task may still be infeasible because of unrealistically large constants that depend on the query. For instance, the approach in [Amarilli, Bourhis, and Senellart 2015] is nonelementary in the query, and the algorithm for safe queries in [Dalvi and Suciu 2012] is generally super-exponential in the query [Suciu, Olteanu, Ré, and Koch 2011]. For this reason, we believe that it is also important to achieve a good understanding of the *combined* complexity of PQE, and to isolate cases where PQE is tractable in combined complexity; similarly to how, e.g., Yannakakis's algorithm can evaluate α-acyclic queries on non-probabilistic instances with tractable combined complexity [Yannakakis 1981]. This motivates the question studied in this chapter: *For which classes of queries and instances does PQE enjoy tractable combined complexity?*

Related work. Surprisingly, the question of achieving combined tractability for PQE does not seem to have been studied before. To our knowledge, the only exception is in the setting of probabilistic XML [Kimelfeld and Senellart 2013], where deterministic tree automata queries were shown to enjoy tractable combined complexity [Cohen, Kimelfeld, and Sagiv 2009].

Questions of combined tractability have also been studied in the setting of *constraint satisfaction problems* (CSP), following a well-known connection between CSP and the conjunctive query evaluation problem in database theory, or the study of the graph homomorphism problem (see, e.g., [Grohe 2007]). We can then see the restriction of PQE to conjunctive queries as a probabilistic, or weighted, variant of these problems, but we are not aware of any existing study of this variant. In the graph homomorphism setting, a related but different problem is that of *counting* graph homomorphisms [Bulatov 2013]: but this amounts to counting the number of matches of a query in a database instance, which is different from counting the possible worlds of an instance where the query has some match, as we do. A more related problem is `#SUB` [Curticapean and Marx 2014], which asks, given a query graph G and an instance graph H, for the *number of subgraphs* of H which are *isomorphic* to G. When all facts are labeled with $1/2$, our problem asks instead for the number of subgraphs of H to which G admits a homomorphism. A further difference is that we allow arbitrary probability annotations, amounting to a form of weighted counting; in particular, facts can be given probability 1.

Problem statement. Inspired by the connection to graph homomorphism and CSP, in this chapter we investigate the probabilistic query evaluation problem for conjunctive queries on tuple-independent instances, over arity-two signatures. To our knowledge, we are the first to focus on the combined complexity of conjunctive query evaluation on probabilistic relational data. For simplicity of exposition, we will phrase our problem in terms of graphs: given a *query graph* and a probabilistic *instance graph*, where each edge is annotated by a probability, we must determine the probability that the query graph has a homomorphism to the instance graph, i.e., the total probability mass of the subgraphs which ensure this, assuming independence between edges. We always assume the query and instance graphs to be directed.

As we will see, the problem is generally intractable, so we will have to study restricted settings. We accordingly study this problem under assumptions on the query and input graphs. One general assumption that we will make is to impose *tree-likeness* of the instance. In fact, we will generally restrict it to be a *polytree*, i.e., a directed graph whose underlying undirected graph is a tree. As we will see, however, even this restriction does not suffice to ensure tractability, so we study the impact of several other features:

- *Labels*, i.e., whether edges of the query and instance can be labeled by a finite alphabet, as would be the case on a relational signature with more than one binary predicate.
- *Disconnectedness*, i.e., allowing disconnected queries and instances.
- *Branching*, i.e., allowing graphs to branch out, instead of requiring them to be a path.

- *Two-wayness*, i.e., allowing edges with arbitrary orientation, instead of requiring all edges to have the same orientation (as in a one-way path, or downward tree).

We accordingly study our problem for *labeled graphs* and *unlabeled graphs*, and when query and instance graphs are in the following classes, that cover the possible combinations of the above characteristics: one-way and two-way paths, downward trees and polytrees, and disjoint unions thereof.

Results. This chapter presents our combined complexity results for the probabilistic query evaluation problem in all these settings. After introducing the preliminaries and defining the problem in Section 2.2, we first study the impact of disconnectedness in instances and queries in Section 2.3. While we can easily show that disconnectedness does not matter for instances (Lemma 2.3.7), we show that disconnectedness of queries has an unexpected impact on complexity: in the labeled case, even the simplest disconnected queries on the simplest kinds of instances are intractable (Proposition 2.3.3): this result is shown via the hardness of counting edge covers in bipartite graphs. The picture for disconnected queries is more complex in the unlabeled case (see Table 2.1): indeed, the problem is still hard when allowing two-wayness in the query and instance (as it can be used to simulate labels, see Proposition 2.3.4), but disallowing two-wayness in the instance ensures tractability of all queries. This latter result (Proposition 2.3.6) is established by showing that all queries then essentially collapse to a one-way path: we do so by assigning a *level* to all vertices of the query using a notion of *graded DAGs* [Odagiri and Goto 2014; Schröder 2016].

We then focus on connected queries, and first study the labeled setting in Section 2.4; see Table 2.2 for a summary of results. We show that disallowing instance branching ensures the tractability of all connected queries (Proposition 2.4.10), and that disallowing branching in the query *and* two-wayness in the instance and query also does (Proposition 2.4.9). These two results are shown by computing the Boolean lineage of the query as a DNF, and proving that we can tractably evaluate its probability because it is β-acyclic [Brault-Baron, Capelli, and Mengel 2015], thanks to the restricted instance structure. For the first result, this process further relies on a CSP tool to show the tractability of homomorphism testing in labeled two-way paths, a condition dubbed the $\underline{X}$-property [Gutjahr, Welzl, and Woeginger 1992; Gottlob, Koch, and Schulz 2006]. We show the intractability of all other cases (Propositions 2.4.1, 2.4.3, and 2.4.4), by coding #SAT-reductions.

We last study the unlabeled setting for connected queries in Section 2.5. We show that disallowing query branching and two-wayness suffices to obtain tractability, provided that the instance is a polytree (Proposition 2.5.2): this result is proven by building in PTIME a deterministic tree automaton to test the length of the longest path, and compiling a d-DNNF lineage as in [Amarilli, Bourhis, and Senellart 2015]. This result immediately extends to branching queries, as they are equivalent to paths in this case (Proposition 2.5.3). We complete the picture by showing that, by contrast, allowing two-wayness in the query leads to intractability on polytrees, by a variant of our coding technique (Proposition 2.5.4).

Our results completely classify the complexity of probabilistic conjunctive query evaluation for all combinations of instance and query restrictions, in the labeled and

unlabeled setting. In particular, our tractability results are essentially always about paths, i.e., when the instance or the query is a path, or can be converted to a path.

2.2 Preliminaries on Probabilistic Graph Homomorphism

We first provide some formal definitions of the concepts we use in this chapter, and introduce the *probabilistic graph homomorphism* problem and the different classes of graphs that we consider.

Labels. Througout this chapter, σ will always denote a finite non-empty set of labels. When $|\sigma| > 1$, we say that we are in the *labeled setting*; when $|\sigma| = 1$, in the *unlabeled setting*. We will always consider that σ is fixed, i.e., σ will never be part of the input for problems that we consider.

Graphs and homomorphisms. We consider *directed graphs* with edge labels from σ, as defined in Section 1.1. We recall that we do not allow multi-edges: an edge e has a unique label $\lambda(e)$. When $|\sigma| = 1$, i.e., in the unlabeled setting, we simply write (V, E) for the graph and $a \to b$ for an edge.

A graph $H' = (V', E', \lambda')$ is a *subgraph* of the graph $H = (V, E, \lambda)$, written $H' \subseteq H$, when we have $V' = V$, $E' \subseteq E$, and when λ' is $\lambda_{|E'}$, i.e., the restriction of λ to E'. (Note that, in a slightly non-standard way, we impose that subgraphs have the same set of vertices than the original graph; this will simplify some notation.)

A *graph homomorphism* h from some graph $G = (V_G, E_G, \lambda_G)$ to some graph $H = (V_H, E_H, \lambda_H)$ is a function $h : V_G \to V_H$ such that, for all $(u, v) \in E_G$, we have $(h(u), h(v)) \in E_H$ and further $\lambda_H((h(u), h(v))) = \lambda_G((u, v))$. In other word, it is a homomorphism between G and H, when seen as relational instances. A *match* of G in H is the image in H of such a homomorphism h, i.e., the graph with vertices $h(u)$ for $u \in V_G$ and edges $(h(u), h(v)))$ for $(u, v) \in E_G$. Note that two different homomorphisms may define the same match. Also note that two distinct nodes of G could have the same image by h, so a match of G in H is not necessarily homomorphic to G. We write $G \rightsquigarrow H$ when there exists a homomorphism from G to H. We call two graphs G and G' *equivalent* if, for any graph H, we have $G \rightsquigarrow H$ iff $G' \rightsquigarrow H$. It is easily seen that G and G' are equivalent if and only if $G \rightsquigarrow G'$ and $G' \rightsquigarrow G$.

Probabilistic graphs. A *probability distribution on graphs* is a function Pr from a finite set $\mathcal{W}$ of graphs (called the *possible worlds* of Pr) to values in $[0; 1]$ represented as rational numbers, such that the probabilities of all possible worlds sum to 1, namely, $\sum_{H \in \mathcal{W}} \Pr(H) = 1$.

A *probabilistic graph* is intuitively a concise representation of a probability distribution. Formally, it is a pair (H, π) where H is a graph with edge labels from σ and where π is a probability function $\pi : E \to [0; 1]$ that maps every edge e of H to a probability $\pi(e)$, represented as a rational number. Note that each edge (u, v) in a probabilistic graph (H, π) is annotated both with a label $\lambda((u, v)) \in \sigma$, and a probability $\pi((u, v))$.

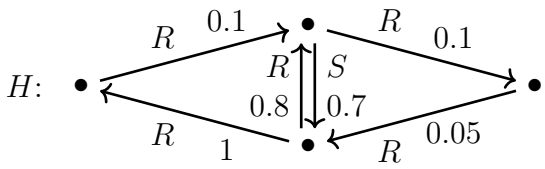

Figure 2.1 – Example probabilistic graph H

The *probability distribution* Pr defined by the probabilistic graph (H, π) is obtained intuitively by considering that edges are kept or deleted independently according to the indicated probability. Formally, the possible worlds $\mathcal{W}$ of Pr are the subgraphs of $H = (V, E, \lambda)$, and for $H' = (V, E', \lambda_{|E'}) \subseteq H$ we define $\Pr(H') := \prod_{e \in E'} \pi(e) \times \prod_{e \in E \setminus E'} (1 - \pi(e))$. Note that, when H has edges labeled with 0 or 1, some possible worlds are given probability 0 by π.

Example 2.2.1. Figure 2.1 represents a probabilistic graph (H, π) on signature $\sigma = \{R, S\}$, where each edge is annotated with its label and probability value. There are 2^6 possible worlds, 2^5 of which have non-zero probability.

The possible world where all R-edges are kept and all S-edges are removed has probability $0.1 \times 1 \times 0.8 \times 0.1 \times 0.05 \times (1 - 0.7)$. ◁

Probabilistic graph homomorphism. The goal of this chapter is to study the *probabilistic homomorphism problem* PHom, for the set of labels σ that we fixed: given a graph G on σ and a probabilistic graph (H, π) on σ, compute the probability that there exists a homomorphism from G to H under Pr, i.e., the sum of the probabilities of all subgraphs H' of H to which G has a homomorphism:

$$\Pr(G \rightsquigarrow H) := \sum_{\substack{H' \subseteq H \\ G \rightsquigarrow H'}} \Pr(H').$$

Example 2.2.2. Continuing the example, consider the PHom problem for the graph $G : \xrightarrow{R} \xrightarrow{S} \xleftarrow{S}$ and the example probabilistic graph (H, π) in Figure 2.1. The graph G intuitively corresponds to the conjunctive query $\exists xyzt\ R(x, y) \wedge S(y, z) \wedge S(t, z)$. Of course, we can compute $\Pr(G \rightsquigarrow H)$ by summing over the possible worlds of H, but this process is generally intractable. Here, by considering the possible matches of G in H, we can see that $\Pr(G \rightsquigarrow H) = 0.7 \times (1 - (1 - 0.1) \times (1 - 0.8))$. ◁

Following database terminology, we call G the *query graph* and (H, π) the *(probabilistic) instance graph*. Indeed, the PHom problem is easily seen to be equivalent (if we allow multi-edges) to conjunctive query evaluation on probabilistic tuple independent relational databases, over binary relational signatures.

Note that we measure the complexity of PHom as a function of *both* the query graph G and of the instance graph (H, π), i.e., in database terminology, we measure the *combined complexity* [Vardi 1995] of probabilistic query evaluation. As we explained, the PHom problem is known to be #P-hard in general [Dalvi and Suciu 2007] (even for some fixed query graphs): by this, we mean that it is hard (under polynomial-time reductions) for the class #P. To achieve tractable complexity for PHom, we will classify the complexity of PHom under various restrictions. We say that the complexity of some variant of the problem is *tractable* if it can be solved in

Figure 2.2 – Inclusions between classes of graphs

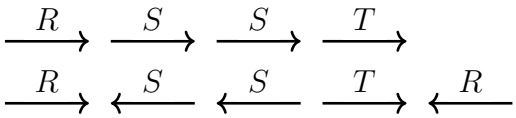

Figure 2.3 – Examples of labeled 1WP (top) and 2WP (bottom) for $\sigma = \{R, S, T\}$

PTIME. All PHom variants that we study will be shown either to be tractable in this sense, or to be #P-hard.

We will study restrictions of PHom first by distinguishing the *labeled* and *unlabeled* settings. We write $\mathsf{PHom}_{\mathsf{L}}$ for the problem when the fixed label set σ is such that $|\sigma| > 1$, and $\mathsf{PHom}_{\not\mathsf{L}}$ when the fixed σ is such that $|\sigma| = 1$.

The second restriction concerns the input query graphs and instance graphs. We will model restrictions on these graphs by requiring them to be taken from specific graph classes, where by *graph class* we simply mean an infinite set of graphs. Inspired by the notation used in CSP, for two classes $\mathcal{G}$ and $\mathcal{H}$ of graphs in the labeled setting, we denote $\mathsf{PHom}_{\mathsf{L}}(\mathcal{G}, \mathcal{H})$ the problem that takes as input a graph G in class $\mathcal{G}$ and a probabilistic graph (H, π) with H in class $\mathcal{H}$, and computes the probability $\Pr(G \rightsquigarrow H)$. We denote the same problem in the unlabeled setting by $\mathsf{PHom}_{\not\mathsf{L}}(\mathcal{G}, \mathcal{H})$.

Graph classes. The graph classes which we study in this chapter are defined as follows, on a graph G with edge labels from σ:

- G is a *one-way path* (1WP) if it is of the form $a_1 \xrightarrow{R_1} \cdots \xrightarrow{R_{m-1}} a_m$ for some m, with all $a_1, \ldots, a_m$ being pairwise distinct, and with $R_i \in \sigma$ for $1 \leqslant i < m$.
- G is a *two-way path* (2WP) if it is of the form $a_1 - \cdots - a_m$, with all $a_1, \ldots, a_m$ being pairwise distinct, and each $-$ being $\xrightarrow{R_i}$ or $\xleftarrow{R_i}$ (but not both) for some label $R_i \in \sigma$.
- G is a *downwards tree* (DWT) if it is a rooted unranked tree (each node can have an arbitrary number of children), with all edges going from parent to child in the tree.
- G is a *polytree* (PT) if its underlying undirected graph is an unranked tree, without restriction on edge directions.

We also consider the class Connected of connected graphs, and write All the class of all graphs. The inclusion diagram between our graph classes is shown in Figure 2.2. We reproduce here (as Figures 2.3 and 2.4) Figures 1 and 2 from the (general) introduction (page 5) for examples of a labeled one-way path and two-way path, and of an unlabeled downwards tree and polytree.

We also introduce the classes $\bigsqcup$1WP (resp., $\bigsqcup$2WP, $\bigsqcup$DWT, $\bigsqcup$PT) of graphs that are *disjoint unions of* 1WP (resp., 2WP, DWT, PT), that is, of possibly disconnected graphs whose connected components are 1WP (resp., 2WP, DWT, PT).

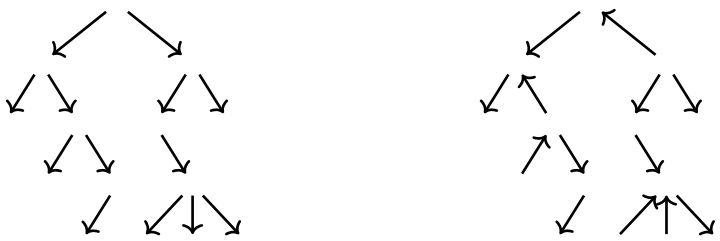

Figure 2.4 – Examples of unlabeled DWT (left) and PT (right)

Our graph classes were chosen to be representative of different features of graphs that will have an impact in the complexity of the PHom problem, namely, *labeling*, *two-wayness*, *branching*, and *disconnectedness*. Indeed, 2WP (resp., PT) adds two-wayness to 1WP (resp., DWT); DWT (resp., PT) adds branching to 1WP (resp., 2WP); and $\bigsqcup$1WP (resp., $\bigsqcup$2WP, $\bigsqcup$DWT, $\bigsqcup$PT) adds disconnectedness to 1WP (resp., 2WP, DWT, PT).

In the following sections, we investigate the complexity of probabilistic graph homomorphism for these various classes of conjunctive queries and instances.

2.3 Disconnected Case

We first consider the case where either the query or probabilistic instance graph is *disconnected*, i.e., not in the Connected class. When the query is disconnected, we show in this section that the probabilistic homomorphism problem is #P-hard in all but the most restricted of cases (in particular in the labeled setting), which justifies that we restrict to connected queries in the rest of the chapter. On the other hand, we will show that disconnectedness in the probabilistic instance graph has essentially no impact on combined complexity.

2.3.1 Labeled Disconnected Queries

We establish our main intractability result on disconnected queries by reduction from the `#Bipartite-Edge-Cover` problem on *undirected* graphs:

Definition 2.3.1. An undirected unlabeled graph is *bipartite* if its vertices can be partitioned into two classes such that no edge connects two vertices of the same class. An *edge cover* of an undirected graph is a subset of its edges such that every vertex is incident to at least one edge of the subset. `#Bipartite-Edge-Cover` is the problem, given a bipartite undirected graph, of counting its number of edge covers. ◁

This problem was shown in [Khanna, Roy, and Tannen 2011] to be intractable.

Theorem 2.3.2 ([Khanna, Roy, and Tannen 2011])**.** *The* `#Bipartite-Edge-Cover` *problem is #P-complete.*

We can then use this result to show intractability for the simplest forms of disconnected query graphs ($\bigsqcup$1WP) on the simplest forms of probabilistic instance graphs (1WP), in the *labeled* case:

Proposition 2.3.3. $\mathsf{PHom}_{\mathsf{L}}(\bigsqcup \mathsf{1WP}, \mathsf{1WP})$ *is #P-hard.*

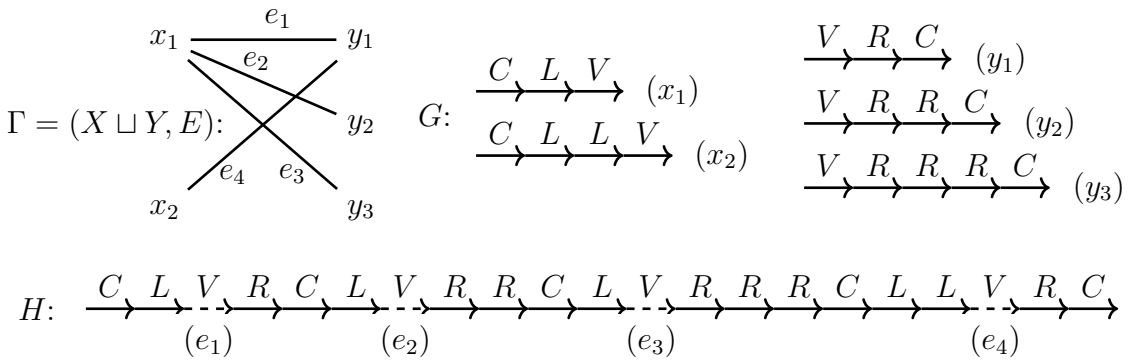

Figure 2.5 – Illustration of the proof of Proposition 2.3.3, for the bipartite graph Γ. Dashed edges have probability $\frac{1}{2}$. We show (within brackets) the edge of Γ coded by each V-labeled edge in the instance graph H, and the vertex of Γ coded by each 1WP component of the query graph G.

Proof. We reduce from #Bipartite-Edge-Cover. Let $\Gamma = (X \sqcup Y, E)$ be an input to #Bipartite-Edge-Cover, i.e., a bipartite undirected graph with parts X and Y; we write $X = (x_1, \ldots, x_{n_\mathrm{l}})$, $Y = (y_1, \ldots, y_{n_\mathrm{r}})$, $E = (e_1, \ldots, e_m)$, and for all $1 \leqslant i \leqslant m$ we write $e_i = (x_{l_i}, y_{r_i})$, with $1 \leqslant l_i \leqslant n_\mathrm{l}$ and $1 \leqslant r_i \leqslant n_\mathrm{r}$.

We first construct in PTIME the 1WP probabilistic graph (H, π): see Figure 2.5 for an illustration of the construction. Specifically, for $1 \leqslant j \leqslant m$, we construct the following 1WP:

$$H_{e_j} := (\xrightarrow{L})^{l_j} \xrightarrow{V} (\xrightarrow{R})^{r_j}.$$

The graph H is then defined as:

$$\xrightarrow{C} H_{e_1} \xrightarrow{C} H_{e_2} \xrightarrow{C} \cdots \xrightarrow{C} H_{e_m} \xrightarrow{C}.$$

We define π as follows: edges labeled by V have probability $\frac{1}{2}$ (intuitively coding whether an edge is part of the candidate cover), all others have probability 1.

We then construct the query graph $G \in \bigsqcup$1WP, coding the edge covering constraints. For every $1 \leqslant i \leqslant n_\mathrm{l}$, the graph G contains the 1WP component $\xrightarrow{C} (\xrightarrow{L})^i \xrightarrow{V}$, and for every $1 \leqslant i \leqslant n_\mathrm{r}$, the graph G contains the 1WP component $\xrightarrow{V} (\xrightarrow{R})^i \xrightarrow{C}$.

It is clear that H is in 1WP, G is in $\bigsqcup$1WP and that both can be constructed in PTIME from Γ. We now show that $\Pr(G \rightsquigarrow H)$ is exactly the number of edge covers of Γ divided by 2^m, so that the computation of the latter reduces in PTIME to the computation of the former, concluding the proof.

To see why, we define a bijection between the subsets of edges of Γ, seen as valuations $\nu : E \to \{0, 1\}$, to the possible worlds H' of H of non-zero probability. We do so in the expected way: keep the one V-edge $\xrightarrow{V}$ of H_{e_i} iff $\nu(e_i) = 1$. We now show that there is a homomorphism from G to H' if and only if ν is an edge cover of Γ. As the number of H''s such that there is a homomorphism from G to H' is exactly $\Pr(G \rightsquigarrow H) \times 2^m$, this will allow us to conclude.

Indeed, if there is a homomorphism h from G to H', then, considering the 1WP component in G that codes the constraint on x_i (resp., on y_i), its image must be of the form $\xrightarrow{C} (\xrightarrow{L})^i \xrightarrow{V}$ (resp., $\xrightarrow{V} (\xrightarrow{R})^i \xrightarrow{C}$), but then by construction of H the V-fact must correspond to an edge e such that x_i (resp., y_i) is adjacent to e, so that we have $\nu(e) = 1$ and so x_i (resp., y_i) is covered. As this is true for each 1WP component, all the vertices are covered and ν is indeed an edge cover of Γ.

Table 2.1 – Tractability of $\mathsf{PHom}_{\not\mathsf{L}}$ for disconnected queries (Section 2.3.2). Results also hold when instances are unions of the indicated classes.

$\downarrow G$ $H\rightarrow$	1WP	2WP	DWT	PT	Connected
$\bigsqcup$1WP					2.5.1
$\bigsqcup$2WP		2.3.4			
$\bigsqcup$DWT				2.5.3	
$\bigsqcup$PT			2.3.6		
All					

PTIME #P-hard Numbers given in cells correspond to the propositions for border cases, the remaining cells can be filled using the inclusions from Figure 2.2.

Conversely, suppose that ν is an edge cover of Γ, then for every vertex x_i (resp., y_i) we know that there exists $1 \leqslant j \leqslant m$ such that $\nu(e_j) = 1$ and $l_j = i$ (resp., $r_j = i$), and we can use the V-fact corresponding to e_j and the surrounding facts to build the homomorphism as above from each component of G to H'. □

The proof of Proposition 2.3.3 crucially requires multiple labels in the signature. Indeed, it is easy to see that, in the unlabeled setting, a query graph in $\bigsqcup$1WP (or even in $\bigsqcup$DWT) is equivalent to the longest path within the graph, and we will show further (Proposition 2.5.3) that $\mathsf{PHom}_{\not\mathsf{L}}(\mathsf{1WP}, \mathsf{1WP})$ (indeed, even $\mathsf{PHom}_{\not\mathsf{L}}(\bigsqcup\mathsf{DWT}, \mathsf{PT})$) is PTIME.

2.3.2 Unlabeled Disconnected Queries

In light of the intractability result of Proposition 2.3.3, let us now consider the unlabeled setting. We show in Table 2.1 where the tractability frontier lies. First, introducing two-wayness in both query and instance graphs is enough to obtain an analogue of the intractability of Proposition 2.3.3:

Proposition 2.3.4. $\mathsf{PHom}_{\not\mathsf{L}}(\bigsqcup\mathsf{2WP}, \mathsf{2WP})$ *is #P-hard.*

Proof. We reduce, again, from the #P-hard problem `#Bipartite-Edge-Cover`. The idea of the reduction is similar to that used in the proof of Proposition 2.3.3, but we face the additional difficulty of not being allowed to use labels. Fortunately, we can use two-wayness to simulate them.

Let $\Gamma = (X \sqcup Y, E)$ be an input of `#Bipartite-Edge-Cover`. Consider the reduction from Γ used in the proof of Proposition 2.3.3 and the 1WP probabilistic graph (H, π) and the $\bigsqcup$1WP query graph G that were constructed. We construct from H and G the unlabeled probabilistic graph H' and unlabeled $\bigsqcup$2WP query graph G' as follows:

- replace each L- or R-labeled edge $a \xrightarrow{L} b$ or $a \xrightarrow{R} b$ in H and G by 3 edges $a \rightarrow\rightarrow\leftarrow b$;
- replace each C-labeled edge $a \xrightarrow{C} b$ of H and G by 3 edges $a \leftarrow\leftarrow\leftarrow b$;
- replace each V-labeled edge $a \xrightarrow{V} b$ of H and G by 6 edges $a \rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\leftarrow b$.

All edges of H' have probability 1, except the first edge of each sequence of 6 edges that replaced a V-labeled edge, which has probability $\frac{1}{2}$.

Consider a 1WP component of G that codes the constraint on a vertex from Y, e.g $\xrightarrow{V} (\xrightarrow{R})^i \xrightarrow{C}$, which was rewritten in G' into $\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\leftarrow (\rightarrow\rightarrow\leftarrow)^i \leftarrow\leftarrow\leftarrow$. A homomorphism from this component into a possible world J' of H' must actually map to a rewriting of a $\xrightarrow{V} (\xrightarrow{R})^i \xrightarrow{C}$ sequence in H': indeed, the key observation is that the first 5 $\rightarrow$ edges can only be matched to 5 consecutive $\rightarrow$ in J', which only exist as the first 5 edges of a sequence of 6 edges that replaced a V-labeled fact in H. There is no choice left to match the subsequent edges without failing. A similar observation holds for components coding the constraints on vertices from X ($\xrightarrow{C} (\xrightarrow{L})^i \xrightarrow{V}$). Hence, we can show correctness of the reduction using the same argument as before. □

Allowing two-wayness in both the query and the instance graphs thus allows us to simulate labels, so that $\mathsf{PHom}_{\not L}$ is intractable. We will study in Section 2.5 what happens for query graph classes without two-wayness (i.e., 1WP, DWT, and unions thereof); so let us now consider the case of instance graph classes where two-wayness is forbidden, i.e., is in $\bigsqcup$DWT. As we will show, $\mathsf{PHom}_{\not L}$ of *arbitrary* query graphs on such $\bigsqcup$DWT instance graphs is tractable. To this end, we need to introduce *level mappings* of acyclic directed graphs (DAGs):

Definition 2.3.5. A *level mapping* of a DAG G is a mapping μ from the vertices of G to $\mathbb{Z}$ such that for each directed edge $u \rightarrow v$ of G we have $\mu(v) = \mu(u) - 1$. We call G a *graded DAG* if it has a level mapping. ◁

An example of graded DAG together with a level mapping is given in Figure 2.6. It is easy to see (and shown in Proposition 1 of [Odagiri and Goto 2014]) that a DAG G is graded iff there are no two vertices u, v and two directed paths χ, χ' in G from u to v such that χ and χ' have different lengths (in the terminology of [Odagiri and Goto 2014], G does not have a *jumping edge*). Graded DAGs are related to the classical notion of graded ordered set [Schröder 2016], and the level mapping function has been called in the literature a *depth function* [Odagiri and Goto 2014], a *grading function* [Schröder 2016], a *set of levels* [Schröder 2016], or a *rank function* [Stanley 1997].

To obtain such a level mapping, we can proceed by picking one vertex in each connected component of G, mapping each of these vertices to level 0, and then exploring G by a breadth-first traversal and assigning the level of each vertex according to the level of the vertex used to reach it, visiting all edges and defining the image of each vertex. It is clear that this process yields a level mapping of G unless it tries to assign two different levels to the same vertex v, which cannot happen if there is no jumping edge [Odagiri and Goto 2014, Proposition 1].

We will now use the notion of graded DAG to show:

Proposition 2.3.6. $\mathsf{PHom}_{\not L}(\mathsf{All}, \bigsqcup\mathsf{DWT})$ *is PTIME.*

Proof. Let G be an arbitrary unlabeled graph and (H, π) a probabilistic graph with $H \in \bigsqcup$DWT. We observe that if G contains a directed cycle, then it cannot have a homomorphism to a subgraph of H (which is necessarily acyclic), so $\Pr(G \rightsquigarrow H) = 0$. Hence, it suffices to study the case where the query graph G is a DAG.

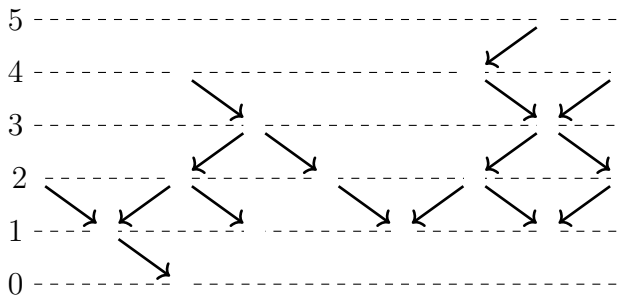

Figure 2.6 – A DAG with a level mapping (dashed lines), see Definition 2.3.5.

Likewise, if there are two vertices u, v of G and directed paths χ, χ' in G from u to v such that χ and χ' have different lengths, then again G cannot have a homomorphism to a subgraph of H: indeed, any subgraph of H is a directed forest and there is at most one directed path between each pair of nodes. So we can assume without loss of generality that there is no such pattern in G, and G is therefore graded.

Letting μ be a level mapping of G, we call the *difference of levels* of μ the difference between the largest and smallest value of its image; the *difference of levels* of G itself is the minimum difference of levels of a level mapping of G. As the level mappings of G only differ in the constant value that they add to all vertices of each connected component, the difference of levels can clearly be computed in PTIME by shifting each connected component so that its minimal level is zero, and computing the difference; we call the result of the shifting the *minimal level mapping* of G.

Letting m be the difference of levels of G, we now make the following claim: *in any subgraph H' of H, there is a homomorphism from G to H' if and only if H' has a directed path of length m.*

This claim implies the result. Indeed, we can first check in PTIME if G has no cycles and has no pairs of paths of different lengths between two endpoints, and return 0 if the conditions are violated. We can then compute in PTIME the difference of levels m of G using the observations above. Now, on any subgraph of H, the query G is equivalent to the 1WP graph $\rightarrow^m$, so our result follows from Proposition 2.5.3 and Lemma 2.3.7 (that we will prove later).

All that remains is to prove the claim. We first note that it suffices to show the claim under the assumption that G is connected. Indeed, if the claim is true for all connected G, then the claim is implied for arbitrary G by considering each of its connected components, applying the claim, and observing that G has a suitable homomorphism to H' iff each one of its connected components does, i.e., iff H' has a directed path whose length is the maximal difference of levels of a connected component of G, and this is precisely the difference of levels m of G. Hence, we now prove the claim for connected G.

We start with the backwards direction of the claim. It is easily seen that there is a homomorphism h' from G to the 1WP graph $\rightarrow^m$. Indeed, we define h' according to the minimal level mapping μ of G: we set h' to map all the vertices whose level is i to the $(m-i)$-th vertex of $\rightarrow^m$. From the existence of h', we know that, whenever there is a homomorphism h from $\rightarrow^m$ to H' (i.e., when H' has a directed path of length m), then $h \circ h'$ is a homomorphism from G to H', which shows the backwards implication.

For the forward direction of the claim, suppose that there exists a homomorphism h from G to H', and let m be the difference of levels of G. Because G is connected and H' is in $\bigsqcup\mathsf{DWT}$, the image of h is actually a DWT, call it T. Now it is easy to see that the image of a node that has level $m - i$ in G has depth i in T, so that T (and so H') contains the $\mathsf{1WP} \rightarrow^m$. This finishes the proof of the converse and thus the proof of Proposition 2.3.6. □

2.3.3 Disconnected Instances

We conclude our study of the disconnected case with the case of disconnected *instance graphs*, which we show to be less interesting than the disconnected *query graphs* that we studied so far. Specifically, when the query is *connected*, PHom on arbitrary instances can reduce in PTIME to PHom of the same queries on a corresponding class of connected instances:

Lemma 2.3.7. *For any class of graphs $\mathcal{H}$, let $\mathcal{H}'$ be the class of connected components of graphs in $\mathcal{H}$. Then for any class of* connected *graphs $\mathcal{G}$, $\mathsf{PHom}_\mathsf{L}(\mathcal{G}, \mathcal{H})$ reduces in PTIME to $\mathsf{PHom}_\mathsf{L}(\mathcal{G}, \mathcal{H}')$, and $\mathsf{PHom}_\not{\mathsf{L}}(\mathcal{G}, \mathcal{H})$ reduces in PTIME to $\mathsf{PHom}_\not{\mathsf{L}}(\mathcal{G}, \mathcal{H}')$.*

Proof. Let $G \in \mathcal{G}$, $H \in \mathcal{H}$, and write $H = H'_1 \sqcup \ldots \sqcup H'_n$: we have $H'_i \in \mathcal{H}'$ for all $1 \leqslant i \leqslant n$. Let π be a probability distribution over H: the independence assumption ensures that the edges of any H'_i are pairwise independent from those of any H'_j for $i \neq j$. Now, as G is connected, any image of a homomorphism from G to H must actually be included in some H'_i. Thus, the computation of $\Pr(G \rightsquigarrow H)$ reduces to that of the $\Pr(G \rightsquigarrow H'_i)$ for $1 \leqslant i \leqslant n$, as follows:

$$\Pr(G \rightsquigarrow H) = 1 - \prod_{1 \leqslant i \leqslant n} (1 - \Pr(G \rightsquigarrow H'_i)).$$ □

We last discuss the case when both the query and instance graphs are disconnected. Let us consider the results of Table 2.1 for connected instance graphs. Clearly, any hardness result for a connected class carries over to the corresponding disconnected class. Conversely, we have shown in Proposition 2.3.6 that $\mathsf{PHom}_\not{\mathsf{L}}(\mathsf{All}, \bigsqcup\mathsf{DWT})$ is PTIME; this implies that all tractable cases in Table 2.1 also hold for unions of the indicated instance classes, except $\mathsf{PHom}_\not{\mathsf{L}}(\bigsqcup\mathsf{1WP}, \bigsqcup\mathsf{PT})$ and $\mathsf{PHom}_\not{\mathsf{L}}(\bigsqcup\mathsf{DWT}, \bigsqcup\mathsf{PT})$. But we have noted at the end of Section 2.3.1 that, in the unlabeled setting, $\bigsqcup\mathsf{1WP}$ or $\bigsqcup\mathsf{DWT}$ query graphs are equivalent to $\mathsf{1WP}$ query graphs: thus, Lemma 2.3.7, together with the tractability of $\mathsf{PHom}_\not{\mathsf{L}}(\mathsf{1WP}, \mathsf{PT})$, implies that $\mathsf{PHom}_\not{\mathsf{L}}(\bigsqcup\mathsf{1WP}, \bigsqcup\mathsf{PT})$ and $\mathsf{PHom}_\not{\mathsf{L}}(\bigsqcup\mathsf{DWT}, \bigsqcup\mathsf{PT})$ are both in PTIME. Hence, the results of Table 2.1 also hold when instances are unions of the indicated classes.

We have thus completed our study of PHom_L and $\mathsf{PHom}_\not{\mathsf{L}}$ for disconnected instances and/or disconnected queries. We accordingly focus on connected queries and instances in the next two sections.

2.4 Labeled Connected Queries

In this section, we focus on the *labeled* setting, i.e., the PHom_L problem, for classes of connected queries and instances. Table 2.2 shows the entire classification of the labeled setting for the classes that we consider.

Table 2.2 – Tractability of $\mathsf{PHom_L}$ in the connected case (Section 2.4)

$\downarrow G$ $H\rightarrow$	1WP	2WP	DWT	PT	Connected
1WP			2.4.9	2.4.1	
2WP			2.4.4		
DWT			2.4.3		
PT					
Connected		2.4.10			

PTIME #P-hard Numbers given in cells correspond to the propositions for border cases, the remaining cells can be filled using the inclusions from Figure 2.2.

Intuitively, we show intractability for polytree instance graphs, and for downward trees instance graphs when the query graphs allow either two-wayness or branching. Conversely, we show tractability of one-way path query graphs on downward trees, and of arbitrary connected queries on two-way path instances. We first present the hardness results, and then the tractability results.

2.4.1 Hardness Results

We recall that, if we allow *arbitrary* connected unlabeled probabilistic instance graphs (or even just 4-partite graphs), then computing the probability that there exists a path of length 2 is already #P-hard: this is shown in [Suciu, Olteanu, Ré, and Koch 2011], and we will state this result in our context as Proposition 2.5.1 in the next section. Hence, if we want to obtain PTIME complexity for PHom, we need to restrict the class of instances. We can start by restricting the instances to be polytrees, but as we show, this does not suffice to ensure tractability:

Proposition 2.4.1. $\mathsf{PHom_L}(\mathsf{1WP}, \mathsf{PT})$ *is #P-hard.*

To show this result, we will reduce from the Boolean formula probability computation problem, as defined in Section 1.7. This problem is known to be #P-hard, even under severe restrictions on the formula φ. We will use the `#PP2DNF` formulation of the above problem, which is #P-hard [Provan and Ball 1983; Suciu, Olteanu, Ré, and Koch 2011]:

Definition 2.4.2. Recall from Section 1.7 that a positive DNF is a Boolean formula that is a disjunction of (conjunctive) *clauses* that are themselves conjunctions of variables of $\mathcal{X}$. Hence we can write a positive DNF φ as

$$\varphi = \bigvee_{1 \leqslant j \leqslant m} \left(\bigwedge_{1 \leqslant i \leqslant n_j} X_{l_{j,i}} \right),$$

and we assume that each variable of $\mathcal{X}$ occurs in φ, as we can eliminate the others without loss of generality.

A *positive partitioned 2-DNF* (PP2DNF) is a positive DNF φ on a partitioned set of variables where each clause contains one variable from each partition. Formally, the variables of φ are $\mathcal{X} \sqcup \mathcal{Y}$, where we write $\mathcal{X} = \{X_1, \ldots, X_{n_1}\}$ and $\mathcal{Y} = \{Y_1, \ldots, Y_{n_2}\}$, and φ is of the form $\bigvee_{j=1\ldots m}(X_{x_j} \wedge Y_{y_j})$ with $1 \leqslant x_j \leqslant n_1$ and $1 \leqslant y_j \leqslant n_2$ for $1 \leqslant j \leqslant m$.

The `#PP2DNF` problem is the Boolean function probability computation problem when we impose that π maps every variable to $1/2$, and that φ is a PP2DNF. ◁

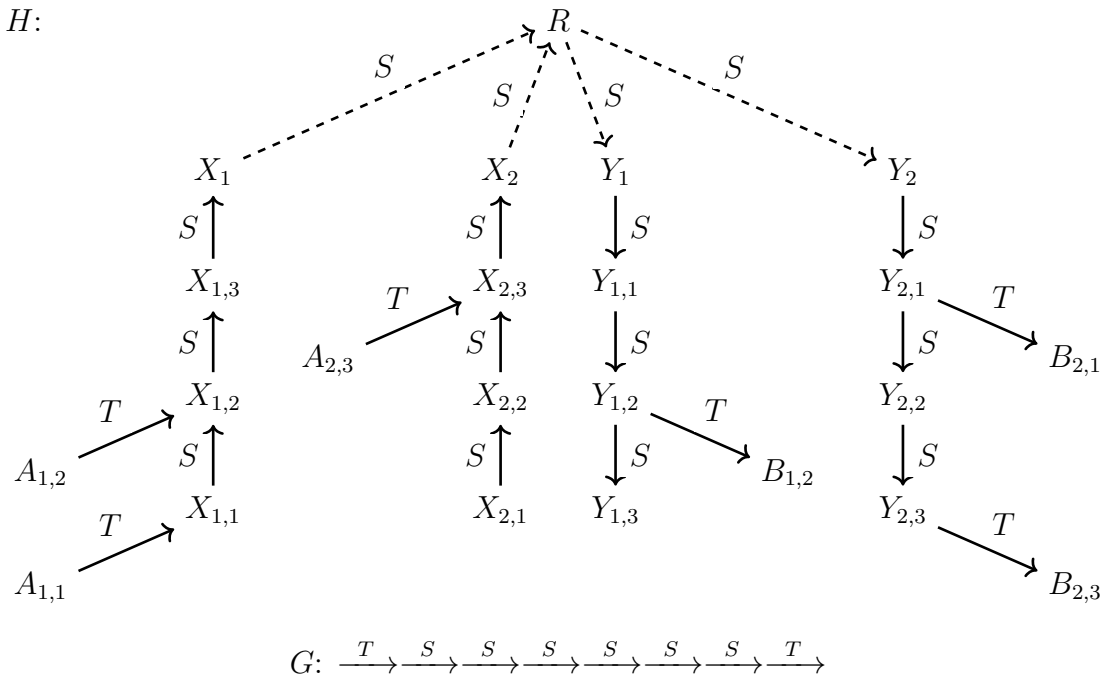

Figure 2.7 – Illustration of the proof of Proposition 2.4.2 for the PP2DNF formula $X_1Y_2 \vee X_1Y_1 \vee X_2Y_2$. Dashed edges have probability $\frac{1}{2}$, all others have probability 1.

We show Proposition 2.4.1 by reducing from #PP2DNF:

Proof of Proposition 2.4.1. See Figure 2.7 for an illustration of the construction for the PP2DNF formula $X_1Y_2 \vee X_1Y_1 \vee X_2Y_2$.

From the PP2DNF formula φ, we construct a PT probabilistic instance where each branch starting at the root describes a variable of the formula. The first edge is probabilistic and represents the choice of valuation. The edges are oriented upwards or downwards depending on whether the variable belongs to $\mathcal{X}$ or to $\mathcal{Y}$. We add a special gadget at different depths of the branch to code the index of each of the clauses where the variable occurs. Formally, we construct the following $\{S, T\}$-labeled probabilistic graph H:

- The vertices of H are $\{R\} \sqcup \{X_1, \dots X_{n_1}\} \sqcup \{Y_1, \dots, Y_{n_2}\} \sqcup \{X_{i,j} \mid 1 \leqslant i \leqslant n_1,\ 1 \leqslant j \leqslant m\} \sqcup \{Y_{i,j} \mid 1 \leqslant i \leqslant n_2,\ 1 \leqslant j \leqslant m\} \sqcup \{A_{x_j,j} \mid 1 \leqslant j \leqslant m\} \sqcup \{B_{y_j,j} \mid 1 \leqslant j \leqslant m\}$.
- The edges of H, all of which have probability 1 except when specified, are:
  - $X_i \xrightarrow{S} R$ for all $1 \leqslant i \leqslant n_1$ and $R \xrightarrow{S} Y_i$ for all $1 \leqslant i \leqslant n_2$, all having probability $\frac{1}{2}$ and intuitively coding the valuation of each variable;
  - For all $1 \leqslant i \leqslant n_1$, the edge $X_{i,m} \xrightarrow{S} X_i$ and the edges $X_{i,j} \xrightarrow{S} X_{i,j+1}$ for all $1 \leqslant j \leqslant m-1$;
  - For all $1 \leqslant i \leqslant n_2$, the edge $Y_i \xrightarrow{S} Y_{i,1}$ and the edges $Y_{i,j} \xrightarrow{S} Y_{i,j+1}$ for all $1 \leqslant j \leqslant m-1$;
  - For all $1 \leqslant j \leqslant m$, the edges $A_{x_j,j} \xrightarrow{T} X_{x_j,j}$ and $Y_{y_j,j} \xrightarrow{T} B_{y_j,j}$, intuitively indicating that variables X_{x_j} and Y_{y_j} belong to clause j.

We then code satisfaction of the formula by a query that tests for a path of a specific length that starts and ends with the gadget. The query has a match exactly on possible worlds where we have set two variables to true such that the sum of the depths of the gadgets corresponds to the query length: this happens iff the two variables occur in the same clause. Specifically, the $\{S, T\}$-labeled graph G is $\xrightarrow{T} (\xrightarrow{S})^{m+3} \xrightarrow{T}$. It is clear that G is a 1WP query graph, H is a polytree and that both can be constructed in PTIME from φ. We now show that $\Pr(G \rightsquigarrow H)$ is exactly the number of satisfying assignments of φ divided by 2^n, so that the computation of one reduces in PTIME to the computation of the other, concluding the proof. To see why, we define a bijection between the valuations ν of $\{X_1, \ldots, X_{n_1}\} \sqcup \{Y_1, \ldots, Y_{n_2}\}$ to the possible worlds H' of H that have non-zero probability, in the expected way: keep the edge $X_i \xrightarrow{S} R$ (resp., $R \xrightarrow{S} Y_i$) iff X_i (resp., Y_i) is assigned to true in the valuation. We then show that there is a homomorphism from G to H' if and only if φ evaluates to true under ν.

Indeed, if there is a homomorphism from G to H', then by considering the only possible matches of the T-edges, one can check easily that the image of the match in H' must be of the following form for some $1 \leqslant j \leqslant m$: $A_{x_j,j} \xrightarrow{T} X_{x_j,j} \xrightarrow{S} X_{x_j,j+1} \xrightarrow{S} \cdots \xrightarrow{S} X_{x_j,m} \xrightarrow{S} X_{x_j} \xrightarrow{S} R \xrightarrow{S} Y_{y_{j'}} \xrightarrow{S} Y_{y_{j'},1} \xrightarrow{S} Y_{y_{j'},2} \xrightarrow{S} \cdots \xrightarrow{S} Y_{y_{j'},j'} \xrightarrow{T} B_{y_{j'},j'}$; further, from the length of the S-path we must have $(m - j) + 4 + (j' - 1) = m + 3$, so that we must have $j = j'$. Then, by construction, X_{x_j} and Y_{y_j} belong to clause j, so the valuation satisfies φ. Conversely, suppose that the valuation satisfies φ, then for some $1 \leqslant j \leqslant m$ we know that X_{x_j} and Y_{y_j} are assigned to true by the valuation, and so we can build the homomorphism as above from G to H'. □

Hence, restricting instances to polytrees is not sufficient to ensure tractability, even for 1WP query graphs. We must thus restrict the instance further, by disallowing one of the two remaining features, namely branching and two-wayness. The first option of disallowing branching, i.e., requiring the instance to be a 2WP, is studied in Section 2.4.2 below, where we show that the problem is tractable for arbitrary query graphs.

The second option is to forbid two-wayness on the instance, i.e., restrict it to be a DWT. In this case, we first show that intractability holds even when we also forbid two-wayness in the query graph, i.e., we also restrict it to be a DWT:

Proposition 2.4.3. $\mathsf{PHom_L}(\mathsf{DWT}, \mathsf{DWT})$ *is #P-hard.*

To show this result, we will reduce from the problem `#PCNF`, which is the Boolean formula probability computation problem when we again impose that π maps every variable to 1/2 and φ is a positive CNF. This problem is #P-hard, as there is a straightforward reduction from `#PP2DNF`.

Proof of Proposition 2.4.3. Let $\mathcal{X} = \{X_1, \ldots, X_n\}$ be a set of variables and $\varphi = \bigwedge_{1 \leqslant j \leqslant m} \left(\bigvee_{1 \leqslant i \leqslant n_j} X_{l_{j,i}} \right)$ be a positive CNF on $\mathcal{X}$ (where $X_{l_{j,i}} \in \mathcal{X}$), where we assume that each variable occurs in φ. From φ, we construct a DWT probabilistic instance where we have probabilistic edges from the root to distinct nodes representing the variables. These edges represent the choice of the valuation. For every clause C_j and variable X appearing in C_j, we have a path of length j starting from the vertex representing the variable X. Formally, we construct the following $\{S, T\}$-labeled probabilistic graph H:

- The vertices of H are $\{R\} \cup \{X_1, \ldots, X_n\} \cup \{C_{i,j,k} \mid 1 \leqslant j \leqslant m,\ 1 \leqslant i \leqslant n_j,\ 1 \leqslant k \leqslant j\} \cup \{C'_{i,j} \mid 1 \leqslant j \leqslant m,\ 1 \leqslant i \leqslant n_j\}$
- The edges of H, all of which have probability 1 except when specified, are:
  - $R \xrightarrow{S} X_i$ for all $1 \leqslant i \leqslant n$, all having probability 1/2 and intuitively coding the valuation of each variable;
  - For all $1 \leqslant j \leqslant m$ and all $1 \leqslant i \leqslant n_j$ (i.e., for each variable occurrence), the edge $X_{l_{j,i}} \xrightarrow{R} C_{j,i,1}$ and edges $C_{j,i,j} \xrightarrow{R} C_{j,i,k+1}$ for $1 \leqslant k \leqslant j-1$; this path intuitively codes that variable $X_{l_{i,j}}$ occurs in the j-th clause;
  - For all $1 \leqslant j \leqslant m$ and all $1 \leqslant i \leqslant n_j$, the edge $C_{j,i,j} \xrightarrow{S} C'_{j,i}$, which we use to detect the end of the path.

We then code satisfaction of the formula by a query that tests for a tree whose branches correspond to the clauses of φ. The $\{S,T\}$-labeled graph G has as vertices $\{R\} \cup \{C_{j,k} \mid 1 \leqslant j \leqslant m,\ 0 \leqslant k \leqslant j\} \cup \{C'_j \mid 1 \leqslant j \leqslant m\}$. For each $1 \leqslant j \leqslant m$, G has one edge $R \xrightarrow{S} C_{j,0}$, a directed R-path $C_{j,0} \xrightarrow{R} C_{j,1} \xrightarrow{R} \cdots \xrightarrow{R} C_{j,j}$, and one edge $C_{j,j} \xrightarrow{S} C'_j$. To show that $\Pr(G \rightsquigarrow H)$ is exactly the number of satisfying assignments of φ divided by 2^n, we define a bijection between the valuations ν of $\mathcal{X}$ to the possible worlds H' of H that have non-zero probability, in the expected way: keep the edge $R \xrightarrow{S} X_i$ iff X_i is assigned to true in the valuation. It is then clear that any valuation ν satisfies φ if and only if G has an homomorphism to H'. □

Moreover, if we forbid branching in the query graph instead of two-wayness, requiring it to be a 2WP, then intractability still holds:

Proposition 2.4.4. $\mathsf{PHom_L}(\mathsf{2WP}, \mathsf{DWT})$ *is #P-hard.*

Proof. The idea of the reduction is the same as that of Proposition 2.4.3, except that we code satisfaction of the formula by a 2WP query. From the positive CNF φ we construct the same probabilistic instance graph H as in Proposition 2.4.3. Concerning the query graph G, we define for $1 \leqslant j \leqslant m$ the following 2WP graphs:

$$G_j = \xrightarrow{S} (\xrightarrow{R})^j \xrightarrow{S} \xleftarrow{S} (\xleftarrow{R})^j$$

Then, G is defined as $\xrightarrow{S} G_1 \xleftarrow{S} G_2 \xleftarrow{S} \ldots \xleftarrow{S} G_j \xleftarrow{S}$, which we construct in PTIME. We then observe that G has an homomorphism to a subgraph H' of H iff the query graph constructed in Proposition 2.4.3 has an homomorphism to H', which concludes. □

Thus, on DWT instances, the only remaining case is when the query is a one-way path. We will now show in the section below that this case is tractable, in addition to the case of arbitrary queries on 2WP instances that we left open above.

2.4.2 Tractability Results

The general proof technique to obtain PTIME combined complexity in this section is inspired by the probabilistic database literature [Suciu, Olteanu, Ré, and Koch 2011]: compute the *lineage* of G on H as a Boolean formula in positive disjunctive normal form (DNF), then compute its probability. We already defined lineages in Section 1.7, but we recall here the definition for our graph setting:

Definition 2.4.5. Let G be a query graph and (H, π) be a probabilistic graph with edge set E. For any valuation $\nu : E \to \{0, 1\}$, we denote by $\nu(H)$ the possible world of H where each edge $e \in E$ is kept iff $\nu(e) = 1$. Letting φ be a Boolean function whose variables are the edges of E, we say that φ *captures the lineage of* G *on* H if, for any valuation $\nu : E \to \{0, 1\}$, the function φ evaluates to 1 under ν iff we have $G \rightsquigarrow \nu(H)$. ◁

Lineage representations allow us to reduce the PHom problem to the Boolean probability computation problem on the lineage function. Formally, for any query graph G and probabilistic graph (H, π), given a Boolean function φ that captures the lineage of G on H, we compute the answer to PHom on G and (H, π) as the probability $\Pr(\varphi, \pi)$ of φ under π: it is immediate by definition that these two quantities are equal.

Of course, computing a lineage representation does not generally suffice to show tractability, because, as we explained in Section 1.7, the Boolean formula probability computation problem is generally intractable. However, computing a Boolean lineage allows us to leverage the known tractable classes of Boolean formulas. Specifically, we will show how to use the class of *β-acyclic positive DNF formulas*, which are known to be tractable [Brault-Baron, Capelli, and Mengel 2015]. We define this notion, by first recalling the notion of a *β-acyclic hypergraph*, and then defining a *β-acyclic positive DNF*:

Definition 2.4.6. Let $\mathcal{H} = (V, E)$ be a hypergraph (see Section 1.1) that does not contain the empty hyperedge. For $v \in V$, we write $\mathcal{H} \setminus v$ for the hypergraph $(V \setminus \{v\}, E')$ where E' is $\{e \setminus \{v\} \mid e \in E\} \setminus \{\emptyset\}$.

A vertex $v \in V$ of $\mathcal{H}$ is called a *β-leaf* [Brault-Baron 2014] if the set of hyperedges that contain it, i.e., $\{e \in E \mid v \in e\}$, is totally ordered by inclusion. In other words, we can write $\{e \in E \mid v \in e\}$ as $(e_1, \ldots, e_k)$ in a way that ensures that $e_i \subseteq e_{i+1}$ for all $1 \leqslant i < k$.

A *β-elimination order* for a hypergraph $\mathcal{H} = (V, E)$ is defined inductively as follows:

- if $E = \emptyset$, then the empty tuple is a β-elimination order for $\mathcal{H}$;
- otherwise, a tuple $(v_1, \ldots, v_n)$ of vertices of $\mathcal{H}$ is a β-elimination order for $\mathcal{H}$ if v_1 is a β-leaf in $\mathcal{H}$ and $(v_2, \ldots, v_n)$ is a β-elimination order for $\mathcal{H} \setminus v_1$.

The hypergraph $\mathcal{H}$ is *β-acyclic* if there is a β-elimination order for $\mathcal{H}$. ◁

We can see a positive DNF (recall Definition 2.4.2) as a hypergraph of clauses on the variables, and introduce the notion of *β-acyclic positive DNFs* accordingly:

Definition 2.4.7. The *hypergraph* $\mathcal{H}(\varphi)$ of a positive DNF on variables $\mathcal{X} = \{X_1, \ldots, X_n\}$ has $\mathcal{X}$ as vertex set and has one hyperedge per clause, i.e., we have $\mathcal{H}(\varphi) := (\mathcal{X}, E)$ with $E := \{\{X_{l_{j,i}} \mid 1 \leqslant i \leqslant n_j\} \mid 1 \leqslant j \leqslant m\}$. We say that the positive DNF φ is *β-acyclic* if $\mathcal{H}(\varphi)$ is β-acyclic. ◁

It follows directly from results by Brault-Baron, Capelli, and Mengel [Brault-Baron, Capelli, and Mengel 2015] about the β-acyclic $\#\mathtt{CSP}_\mathrm{d}$ problem that we can tractably compute the probability of β-acyclic positive DNFs:

Theorem 2.4.8. *The Boolean formula probability computation problem is in PTIME when restricted to β-acyclic positive DNF formulas.*

This is indeed a special case of the $\#\mathrm{CSP}_\mathrm{d}$ problem studied in [Brault-Baron, Capelli, and Mengel 2015], as we remind here:

Proof. The $\#\mathrm{CSP}_\mathrm{d}$ problem studied in [Brault-Baron, Capelli, and Mengel 2015] is about computing a partition function over the hypergraph, under weighted constraints on hyperedges: it generalizes the problem of counting the number of valuations of β-acyclic formulas in conjunctive normal form (CNF) by [Brault-Baron, Capelli, and Mengel 2015, Lemma 3]. We show how the result extends to β-acyclic positive DNF, using de Morgan's law, and to probability computation for weighted variables, using additional constraints on singleton variable sets.

First, we recall their definition of $\#\mathrm{CSP}_\mathrm{d}$ (Definitions 1 and 2 in [Brault-Baron, Capelli, and Mengel 2015]) in the case of a Boolean domain. We denote by $\mathbb{Q}_+$ the nonnegative rational numbers. Denote by $\{0,1\}^{\mathcal{X}}$ the set of functions from $\mathcal{X}$ to $\{0,1\}$, i.e., the Boolean valuations of $\mathcal{X}$. For $\nu \in \{0,1\}^{\mathcal{X}}$ and $\mathcal{Y} \subseteq \mathcal{X}$, we denote by $\nu_{|\mathcal{Y}}$ the restriction of ν to $\mathcal{Y}$. A *weighted constraint (with default value)* on variables $\mathcal{X}$ is a pair $c = (f, \mu)$ that consists of a function $f : S \to \mathbb{Q}_+$ for some subset S of $\{0,1\}^{\mathcal{X}}$, called the *support* of c, and a *default value* $\mu \in \mathbb{Q}_+$; we write $\mathrm{var}(c) := \mathcal{X}$. The constraint c induces a total function on $\{0,1\}^{\mathcal{X}}$, also denoted c, that maps $\nu \in \{0,1\}^{\mathcal{X}}$ to $f(\nu)$ if $\nu \in S$, and to μ otherwise. The *size* of c is $|c| = |S| \times |\mathcal{X}|$. Intuitively, a constraint with default value assigns a weight in $\mathbb{Q}_+$ to all valuations of $\mathcal{X}$, but the default value mechanism allows us to avoid writing explicitly the complete table of this mapping.

An instance of the $\#\mathrm{CSP}_\mathrm{d}$ problem then consists of a finite set I of weighted constraints. The size of I is $|I| := \sum_{c \in I} |c|$, and we write $\mathrm{var}(I) := \bigcup_{c \in I} \mathrm{var}(c)$. The output of the problem is the *partition function*

$$w(I) = \sum_{\nu \in \{0,1\}^{\mathrm{var}(I)}} \prod_{c \in I} c(\nu_{|\mathrm{var}(c)}).$$

The *hypergraph* $\mathcal{H}(I)$ of the $\#\mathrm{CSP}_\mathrm{d}$ instance I (defined in Section 2.2 of [Brault-Baron, Capelli, and Mengel 2015]) is the hypergraph $(\mathrm{var}(I), E_I)$ where $E_I = \{\mathrm{var}(c) \mid c \in I\}$. We say that I is *β-acyclic* if $\mathcal{H}(I)$ is a β-acyclic hypergraph (recall Definition 2.4.6), and we call *β-acyclic* $\#\mathrm{CSP}_\mathrm{d}$ the problem $\#\mathrm{CSP}_\mathrm{d}$ restricted to β-acyclic instances. By Theorem 26 of [Brault-Baron, Capelli, and Mengel 2015], the problem β-acyclic $\#\mathrm{CSP}_\mathrm{d}$ is in PTIME.

We now explain how to reduce the probability computation problem to the β-acyclic $\#\mathrm{CSP}_\mathrm{d}$ problem. Let $\varphi = \bigvee_{1 \leqslant j \leqslant m} \left(\bigwedge_{1 \leqslant i \leqslant n_j} X_{l_{j,i}} \right)$ be a Boolean β-acyclic DNF on variables $\mathcal{X}$, with probabilities $\pi(X) \in [0,1]$ for each $X \in \mathcal{X}$. We construct in linear time from φ and π the variable set $\mathcal{X}' := \{X' \mid X \in \mathcal{X}\}$, the CNF formula $\varphi' := \bigwedge_{1 \leqslant j \leqslant m} \left(\bigvee_{1 \leqslant i \leqslant n_j} X'_{l_{j,i}} \right)$, and the probability valuation π' on $\mathcal{X}'$ defined by $\pi'(X'_{l_{j,i}}) = 1 - \pi(X_{l_{j,i}})$. By De Morgan's duality law we have $\Pr(\varphi, \pi) = 1 - \Pr(\varphi', \pi')$; hence, the probability computation problem for φ and π reduces in PTIME to the same problem for φ' and π'.

We then construct in linear time a β-acyclic $\#\mathrm{CSP}_\mathrm{d}$ instance I such that $\Pr(\varphi', \pi')$ equals $w(I)$, which concludes the proof. For each variable $X' \in \mathcal{X}'$, we define a weighted constraint $c_{X'}$ on variables $\{X'\}$ by $c_{X'}(X' \mapsto 1) = \pi'(X')$ and $c_{X'}(X' \mapsto 0) = 1 - \pi'(X')$, which codes the probability of the variables. Now, for each clause

$1 \leqslant j \leqslant m$, just like in Lemma 3 of [Brault-Baron, Capelli, and Mengel 2015], we define a weighted constraint $c_j = (f_j, 1)$ with default value 1 whose variables are $\{X'_{l_{j,i}} \mid 1 \leqslant i \leqslant n_j\}$, i.e., those that occur in the clause: $f_j(\nu)$ is 0 for the (unique) valuation that sets all variables of the clause to 0, intuitively coding the constraint of the clause. From the fact that φ was β-acyclic, it is clear that I is also β-acyclic. Now, the result $w(I)$ of the partition function sums over all valuations of the variables of I, namely the variables $\mathcal{X}'$ of φ'. Whenever a valuation does not satisfy some clause $1 \leqslant j \leqslant m$, the weighted constraint c_j will give it weight 0, hence ensuring that the product evaluates to 0, so we can restrict the sum to valuations that satisfy φ': such valuations are given weight 1 by all weighted constraints c_j. Now, it is easy to see that the weight of valuations ν that satisfy φ is their probability $\pi'(\nu)$, as each $c_{X'}$ gives them weight $\pi'(X')$ or $1 - (\pi'(X'))$ depending on whether $\nu(X')$ is 1 or 0. Hence, we have reduced the probability computation problem for β-acyclic DNF formulas to β-acyclic $\#\mathsf{CSP}_\mathrm{d}$ in PTIME, which concludes the proof. □

We will then use the tractability of β-acyclic formulas to show PTIME combined complexity results for our PHom_L problem. The first result that we show is tractability for labeled 1WP query graphs on DWT probabilistic instance graphs:[1]

Proposition 2.4.9. $\mathsf{PHom}_\mathsf{L}(\mathsf{1WP}, \mathsf{DWT})$ *is PTIME.*

Proof. Let $G := u_1 \xrightarrow{R_1} \cdots \xrightarrow{R_{m-1}} u_m$ be the 1WP query (where all R_i are not necessarily distinct), and H be the downwards tree instance. The idea is to construct the lineage of G on H as a β-acyclic DNF φ, so that we can conclude with Theorem 2.4.8. It is clear that any match of G can only be a downwards path of H, hence we construct φ as follows: for every downwards path $a_1 \xrightarrow{R'_1} \cdots \xrightarrow{R'_{m-1}} a_m$ of length m of H (their number is linear in $|H|$ because each path is uniquely defined by the choice of a_m) check if the path is a match of G (i.e, check that $R_i = R'_i$ for $1 \leqslant i \leqslant m-1$), and if it is the case then create a new clause of φ whose variables are all the facts $a_i \xrightarrow{R_i} a_{i+1}$ for $1 \leqslant i \leqslant m-1$.

The formula φ thus obtained is then a DNF representation of the lineage of H on G, and has been built in time $O(|H| \cdot |G|)$, i.e., in PTIME. We now justify that φ is β-acyclic by giving a β-elimination order for φ: while H still has edges, repeatedly pick a leaf b of H and, letting a be the parent of b, eliminate the variable $a \xrightarrow{R} b$ from φ. Such a variable will always be a β-leaf, as any set of downwards paths of a downwards tree all ending at a leaf is necessarily ordered by inclusion. From the above, the fact that φ is β-acyclic suffices to conclude the proof. □

Interestingly, we were not able to prove Proposition 2.4.9 using a tree automata-based dynamic programming approach (like we will do later for Proposition 2.5.2).

The second result that we show is tractability when restricting the instance to be a 2WP, and allowing arbitrary *connected* queries (remember from Proposition 2.3.3 that the problem is hard even on 1WP instances if we allow *disconnected* queries):

Proposition 2.4.10. $\mathsf{PHom}_\mathsf{L}(\mathsf{Connected}, \mathsf{2WP})$ *is PTIME.*

To show this result, we follow the same scheme as in the proof of Proposition 2.4.9 above: (i) enumerate all candidate matches; (ii) check whether they are indeed

[1] The connection to β-acyclicity in this context is due to Florent Capelli.

matches; and (iii) argue that the resulting lineage is β-acyclic. For the first step, there are polynomially many candidate matches to consider, because matches are necessarily connected subgraphs of the instance graph H, that are uniquely defined by their endpoints: this is where we use connectedness of the query. For the third step, the resulting lineage is β-acyclic for the same reason as in Proposition 2.4.9, as we can eliminate variables following the order of the path H: all connected subpaths containing an endpoint of the path are ordered by inclusion. What changes, however, is the second step: from the quadratically many possible matches, to compute the lineage expression, we must decide which ones are actually matches.

Deciding this for each subpath amounts to testing, given the connected query graph G and a candidate match H', whether $G \rightsquigarrow H'$, in the non-probabilistic sense. This graph homomorphism problem is generally intractable, but here the minimal match H' is a 2WP (as it is a subpath of H), so it turns out to enjoy combined tractability. The corresponding result was first shown in [Gutjahr, Welzl, and Woeginger 1992] for *unlabeled* graphs, when the instance graph is a path, or for more general instances satisfying a condition called the $\underline{X}$-property; this was generalized to labeled graphs in [Gottlob, Koch, and Schulz 2006]. We recall here the definition of this property:

Definition 2.4.11 (Definition 3.2 of [Gottlob, Koch, and Schulz 2006])**.** Let $H = (V, E, \lambda)$ be a directed graph with labels on σ, let $R \in \sigma$, and let $<$ be a total order on V. Then R is said to have the $\underline{X}$*-property* w.r.t. $<$ if for all $n_0, n_1, n_2, n_3 \in V$ such that $n_0 < n_1$ and $n_2 < n_3$, if we have $n_0 \xrightarrow{R} n_3$ and $n_1 \xrightarrow{R} n_2$ then we also have $n_0 \xrightarrow{R} n_2$. H is said to have the $\underline{X}$*-property* w.r.t. $<$ if it is the case of each label R. ◁

Theorem 2.4.12. *(Theorem 3.5 of [Gottlob, Koch, and Schulz 2006], extending Theorem 3.1 of [Gutjahr, Welzl, and Woeginger 1992]) Given a labeled query graph G, and given a labeled directed graph H with the $\underline{X}$-property w.r.t. some order $<$, we can determine in time $O(|H| \times |G|)$ whether $G \rightsquigarrow H$.*

We can use this result to check, for all connected subpaths of the 2WP instance graph, whether the query graph has a homomorphism to the subpath. This leads to the following proof of Proposition 2.4.10:

Proof of Proposition 2.4.10. We proceed following the three-step process outlined above. We first enumerate the possible query matches in the instance, i.e., the quadratic number of connected subpaths. Second, we test for each subpath $a_i - \cdots - a_{i+k}$ whether it satisfies the query. We can do so tractably because the subpath clearly has the $\underline{X}$-property w.r.t. the order $a_i < \cdots < a_{i+k}$: using the notation of Definition 2.4.11, there are in fact no n_0, n_1, n_2, n_3 that satisfy the conditions. Third, having computed the resulting DNF, we compute its probability using β-acyclicity, eliminating variables in the order of the path as we explained above. □

2.5 Unlabeled Connected Queries

We now turn to the unlabeled setting, whose classification is presented in Table 2.3. We start with an intractability result which follows directly from the well-known intractability of query evaluation in probabilistic databases [Suciu, Olteanu, Ré, and Koch 2011], that we have already mentioned:

Table 2.3 – Tractability of $\mathsf{PHom}_{\not\mathsf{L}}$ in the connected case (Section 2.5)

$\downarrow G$ $H\rightarrow$	1WP	2WP	DWT	PT	Connected
1WP					2.5.1
2WP				2.5.4	
DWT				2.5.3	
PT					
Connected		2.4.10	2.3.6		

PTIME #P-hard Numbers given in cells correspond to the propositions for border cases, the remaining cells can be filled using the inclusions from Figure 2.2.

Proposition 2.5.1 ([Suciu, Olteanu, Ré, and Koch 2011])**.** *The problem* $\mathsf{PHom}_{\not\mathsf{L}}(\mathsf{1WP}, \mathsf{Connected})$ *is #P-hard.*

Proof. Example 3.3 of [Suciu, Olteanu, Ré, and Koch 2011] states that the conjunctive query $\exists x \exists y \exists z\ U(x,y) \wedge U(y,z)$ is #P-hard on TID instances. In other words, $\mathsf{PHom}_{\not\mathsf{L}}(\{\rightarrow\rightarrow\}, \mathsf{All})$ is #P-hard, which implies the #P-hardness of $\mathsf{PHom}_{\not\mathsf{L}}(\mathsf{1WP}, \mathsf{All})$. We conclude using Lemma 2.3.7, which provides a PTIME (Turing) reduction from the $\mathsf{PHom}_{\not\mathsf{L}}(\mathsf{1WP}, \mathsf{Connected})$ problem to the $\mathsf{PHom}_{\not\mathsf{L}}(\mathsf{1WP}, \mathsf{All})$ problem. □

Note that this $\mathsf{PHom}_{\not\mathsf{L}}(\mathsf{1WP}, \mathsf{Connected})$ problem can be phrased in a very simple way: given an unlabeled connected probabilistic graph (H, π) and a length m as input (namely, that of the 1WP graph query), compute the probability that H contains a directed path of length m.

This result suggests that, to obtain tractability, we need to restrict the instance graphs. In fact, such tractability results were already obtained in the previous sections. In Section 2.3, we proved (Proposition 2.3.6) that $\mathsf{PHom}_{\not\mathsf{L}}(\mathsf{All}, \mathsf{DWT})$ has PTIME combined complexity. Similarly, in the previous section, we proved that $\mathsf{PHom}_{\mathsf{L}}(\mathsf{Connected}, \mathsf{2WP})$ is PTIME (Proposition 2.4.10), which implies that $\mathsf{PHom}_{\not\mathsf{L}}(\mathsf{Connected}, \mathsf{2WP})$ is also PTIME. This completes the analysis of the unlabeled case for 1WP, 2WP and DWT instances (see Table 2.3), so the only remaining case is that of PT instances.

We start our study of $\mathsf{PHom}_{\not\mathsf{L}}$ for PT instances with the simplest queries, namely, 1WP, for which we will show tractability. We will proceed by translating the 1WP query to a bottom-up deterministic tree automata (defined in Section 1.5) that will run on uncertain trees. This will use again the notion of lineage, which was extended in [Amarilli, Bourhis, and Senellart 2015] to tree automata running on trees with uncertain Boolean labels: the *lineage* of an automaton on such a tree describes the set of annotations of the tree that makes the automaton accept. In this context, the lineage of *deterministic* tree automata was shown in [Amarilli, Bourhis, and Senellart 2016] to be compilable to a d-DNNF (see Section 1.7). Combining these tools, we can show that $\mathsf{PHom}_{\not\mathsf{L}}$ on one-way path queries and polytree instances is tractable:

Proposition 2.5.2. $\mathsf{PHom}_{\not\mathsf{L}}(\mathsf{1WP}, \mathsf{PT})$ *is PTIME.*

Proof. The idea of the proof is to construct in polynomial time in the query graph G a bottom-up deterministic automaton A_G, which runs on binary trees T representing possible worlds of the polytree instance H, and accepts such a tree T iff the corresponding possible world satisfies G. We can then construct a d-DNNF representation

of the lineage of G on H by [Amarilli, Bourhis, and Senellart 2016, Theorem 6.11], which allows us to efficiently compute $\Pr(G \rightsquigarrow H)$: the complexity of this process is in $O(|A_G| \cdot |H|)$, hence polynomial in $|H| \cdot |G|$. (An alternative way to see this is to use the results of [Cohen, Kimelfeld, and Sagiv 2009].)

Intuitively, the design of the bottom-up automaton ensures that, when it reaches a node n after having processed the subtree T_n rooted at n, its state reflects three linear-sized quantities about T_n:

1. the length of the longest path leading out of n;
2. the length of the longest path leading to n;
3. the length of the longest path overall in T_n (not necessarily via n).

The final states are those where the third quantity is greater than the length of G. The transitions compute each triple from the child triples by considering how the longest leading paths are extended, and how longer overall paths can be formed by joining an incoming and outgoing path.

We now describe formally the construction. Let G, (H, π) be the 1WP query graph and the probabilistic PT instance, and m be the length of G. Because we will use automata that run on full binary trees, we will have to represent possible worlds of H as full binary trees. The first step is to transform H in linear time into a full binary polytree H' by applying a variant of the left-child-right-sibling encoding: in so doing, in addition to unlabeled edges of both orientations that exist in the polytree, we will also introduce some edges called *ε-edges* that are labeled by ε and whose orientation does not matter (so we see them as undirected edges and write them $a - b$); intuitively, the ε-edge $a - b$ means that a and b are in fact the same. For a node $a \in H$ and a child b of a, we say that b is an *up-child* of a if we have $b \to a$ and a *down-child* of a if we have $a \to b$. We do this transformation by processing H bottom-up as follows:

- If n is a leaf node of H, then create a node n' in H'.
- If n is an internal node of H with up-children $u_1, \ldots, u_k$ and down-children $d_1, \ldots, d_l$ then, letting $u'_1, \ldots, u'_k$ and $d'_1, \ldots, d'_l$ be the corresponding nodes in H': create a node n' in H' and nodes $n'_1, \ldots, n'_{k+l-2}$ with the following ε-edges: $n' - n'_1 - \ldots - n'_{k+l-2}$, all having probability 1. Create an edge $u'_1 \to n'$ whose probability is that of $u_1 \to n$. For $2 \leqslant i \leqslant k$ create an edge $u'_i \to n'_{i-1}$ annotated with the same probability as $u_i \to n$. For $1 \leqslant i \leqslant l-1$ create an edge $n'_{k-1+i} \to d'_i$ annotated with the same probability as $n \to d'_i$, and finally create an edge $n'_{k-2+l} \to d'_l$ annotated with the same probability as $n \to d'_l$. Last, if any node has exactly one children (specifically, n', in case $k + l = 1$), then create a node n'' in H' and connect it with an ε-edge to the node.

One can check that H' is indeed a full binary polytree (with some edges being labeled by ε and being undirected) and that $\Pr(G \rightsquigarrow H)$ equals the probability that H' contains a path of the form $(\to -^*)^m$, that is, m occurrences of a directed edge $\to$ followed by some sequence of ε-edges $-$.

The second step is to transform in linear time H' into a probabilistic tree T to which we can apply the construction of [Amarilli, Bourhis, and Senellart 2015].

Specifically, T must be an ordered full binary rooted tree whose edges do not have a label or an orientation, but whose nodes n carry a label in some finite alphabet Γ (written $\lambda(n)$, where λ is the labeling function) and with a probability value written $\pi(n)$. Writing $\overline{\Gamma} := \Gamma \times \{0,1\}$ as in [Amarilli, Bourhis, and Senellart 2015], the semantics of T is that it stands for a probability distribution on $\overline{\Gamma}$-trees, i.e., trees T' labeled with $\Gamma \times \{0,1\}$, which have same skeleton as T: for each node n of T, the corresponding node n' in a possible world T' has label $(\lambda(n), 1)$ with probability $\pi(n)$ and label $(\lambda(n), 0)$ otherwise. We do this transformation by first adding a new root vertex to H' with an ε-edge with probability 1 to the original root (this clearly does not change the probability that H' has a path of the prescribed form), and then simply create T from H' by assigning the label and probability of each node that is not the new root as the direction of its parent edge (in $\Gamma := \{\uparrow, \downarrow, -\}$) and its probability (so the root of T' has label $-$ and probability 1).

Our last step is to construct a bDTA A_G running on $\overline{\Gamma}$-trees such that for every possible world W of H', letting T_W be its representation as a $\overline{\Gamma}$-tree, A_G accepts T_W if and only if W contains a path of the form $(\rightarrow -^*)^m$. The states of A_G are of the form $\langle \uparrow\!\!: i,\ \downarrow\!\!: j,\ \text{Max: } k \rangle$ for $0 \leqslant i, j \leqslant k \leqslant m$, which ensures that A_G is of size polynomial in $|G|$ (and we will construct it in PTIME from G). The idea is that when a node n of T_W will be in such a state, it will mean that:

- Letting W_n be the subinstance of W which is represented by the subtree of T_W rooted at n, and letting r_n be the root of W_n, the longest directed upwards path in W_n finishing at r_n has length i (the path is the longest of the form $(\uparrow -^*)^*$ that ends at r_n).
- The longest directed downwards path in W_n beginning at r_n has length j (the path is the longest of the form $(\downarrow -^*)^*$ that begins at r_n).
- The longest directed path in W_n has length k (the path is of the form $(\rightarrow -^*)^k$ and is the longest in W_n).

We now describe the initialization function ι of A_G:

- $\iota((s,0)) := \langle \uparrow\!\!: 0,\ \downarrow\!\!: 0,\ \text{Max: } 0 \rangle$ for any $s \in \Gamma$.
- $\iota((-,1)) := \langle \uparrow\!\!: 0,\ \downarrow\!\!: 0,\ \text{Max: } 0 \rangle$.
- $\iota((\uparrow,1)) := \langle \uparrow\!\!: 1,\ \downarrow\!\!: 0,\ \text{Max: } 1 \rangle$.
- $\iota((\downarrow,1)) := \langle \uparrow\!\!: 0,\ \downarrow\!\!: 1,\ \text{Max: } 1 \rangle$.
- $\Delta((\uparrow,1), \langle \uparrow\!\!: i,\ \downarrow\!\!: j,\ \text{Max: } k \rangle, \langle \uparrow\!\!: i',\ \downarrow\!\!: j',\ \text{Max: } k' \rangle) := \langle \uparrow\!\!: i'',\ \downarrow\!\!: 0,\ \text{Max: } k'' \rangle$ where $i'' := \min(m, \max(i+1, i'+1))$ and $k'' := \min(m, \max(i'', i+j', i'+j, k, k'))$.
- $\Delta((\downarrow,1), \langle \uparrow\!\!: i,\ \downarrow\!\!: j,\ \text{Max: } k \rangle, \langle \uparrow\!\!: i',\ \downarrow\!\!: j',\ \text{Max: } k' \rangle) := \langle \uparrow\!\!: 0,\ \downarrow\!\!: j'',\ \text{Max: } k'' \rangle$ where $j'' := \min(m, \max(j+1, j'+1))$ and $k'' := \min(m, \max(j'', i+j', i'+j, k, k'))$.
- $\Delta((-,1), \langle \uparrow\!\!: i,\ \downarrow\!\!: j,\ \text{Max: } k \rangle, \langle \uparrow\!\!: i',\ \downarrow\!\!: j',\ \text{Max: } k' \rangle) := \langle \uparrow\!\!: i'',\ \downarrow\!\!: j'',\ \text{Max: } k'' \rangle$ where $i'' := \max(i, i')$ and $j'' := \max(j, j')$ and $k'' := \min(m, \max(k, k', i+j', i'+j))$.

- $\Delta((s,0), \langle\uparrow\!\!:\, i,\ \downarrow\!\!:\, j,\ \text{Max:}\ k\rangle, \langle\uparrow\!\!:\, i',\ \downarrow\!\!:\, j',\ \text{Max:}\ k'\rangle) := \langle\uparrow\!\!:\, 0,\ \downarrow\!\!:\, 0,\ \text{Max:}\ k''\rangle$ where $k'' := \min(m, \max(k, k', i+j', i'+j))$ for every $s \in \{-, \uparrow, \downarrow\}$.

The final state of A_G are all the states $\langle\uparrow\!\!:\, i,\ \downarrow\!\!:\, j,\ \text{Max:}\ k\rangle$ such that $k = m$. One can check by a straightforward induction that the semantics of each state is respected, so that indeed the automaton tests the query G.

We conclude thanks to Proposition 3.1 of [Amarilli, Bourhis, and Senellart 2015] by computing in linear time in $|A_G|$ and $|H'|$ a representation of the lineage on H' of the query that checks whether the input contains a directed path of the form $(\rightarrow -^*)^m$, and observe by Theorem 6.11 of [Amarilli, Bourhis, and Senellart 2016] that it is a d-DNNF. We then compute the probability of this d-DNNF [Darwiche 2001], yielding $\Pr(G \rightsquigarrow H)$ in PTIME: this concludes the proof. □

Hence, PT instances enjoy tractability for the simplest query graphs (i.e., 1WP). We now study whether Proposition 2.5.2 can be extended to more general queries. We first notice that this result immediately extends to branching (i.e., to DWT queries), and even to unions of DWT queries. Indeed, in the unlabeled setting, as we already observed at the end of Section 2.3.1, such queries are equivalent to 1WP queries:

Proposition 2.5.3. $\mathsf{PHom}_{\not L}(\mathsf{DWT}, \mathsf{PT})$ *and* $\mathsf{PHom}_{\not L}(\bigsqcup\mathsf{DWT}, \mathsf{PT})$ *are PTIME.*

Proof. We first show the result for a DWT query graph G. Let m be its height, i.e., the length of the longest directed path it contains, and let G' be the 1WP of length m, which can be computed in PTIME from G. It is easy to observe that G and G' are equivalent. Indeed, we can find G' as a subgraph of G by taking any directed path of maximal length, and conversely there is a homomorphism from G to G': map the root of G to the first vertex of G' and each element of G' at distance i from the root to the i-th element of G'. Hence, $\mathsf{PHom}_{\not L}$ on G and an input probabilistic PT instance reduces to $\mathsf{PHom}_{\not L}$ on G' and the same instance, so that the result follows from Proposition 2.5.2.

The same argument extends to $\bigsqcup$DWT by considering the greatest height of a connected component of G. □

Thus, we have successfully extended from 1WP to $\bigsqcup$DWT queries while preserving tractability on PT instances. However, as we now show, tractability is not preserved if we extend queries to allow two-wayness. Indeed:

Proposition 2.5.4. $\mathsf{PHom}_{\not L}(\mathsf{2WP}, \mathsf{PT})$ *is #P-hard.*

Proof. We adapt the proof of Proposition 2.4.1 by using the same idea than in Proposition 2.3.4: two-wayness allows us to simulate labels. An illustration of the reduction is given in Figure 2.8.

We reduce from #PP2DNF (recall Definition 2.4.2): the input consists of two disjoint sets $\mathcal{X} = \{X_1, \ldots, X_{n_1}\}$, $\mathcal{Y} = \{Y_1, \ldots, Y_{n_2}\}$ of Boolean variables, and a PP2DNF formula φ. We construct a 2WP query graph G' and PT instance H' with the same construction as the one that yielded H and G in that proof, except that we perform the following replacements (see Figure 2.8):

- replace every edge $a \xrightarrow{S} b$ of H and G by 3 edges $a \rightarrow\rightarrow\leftarrow b$;

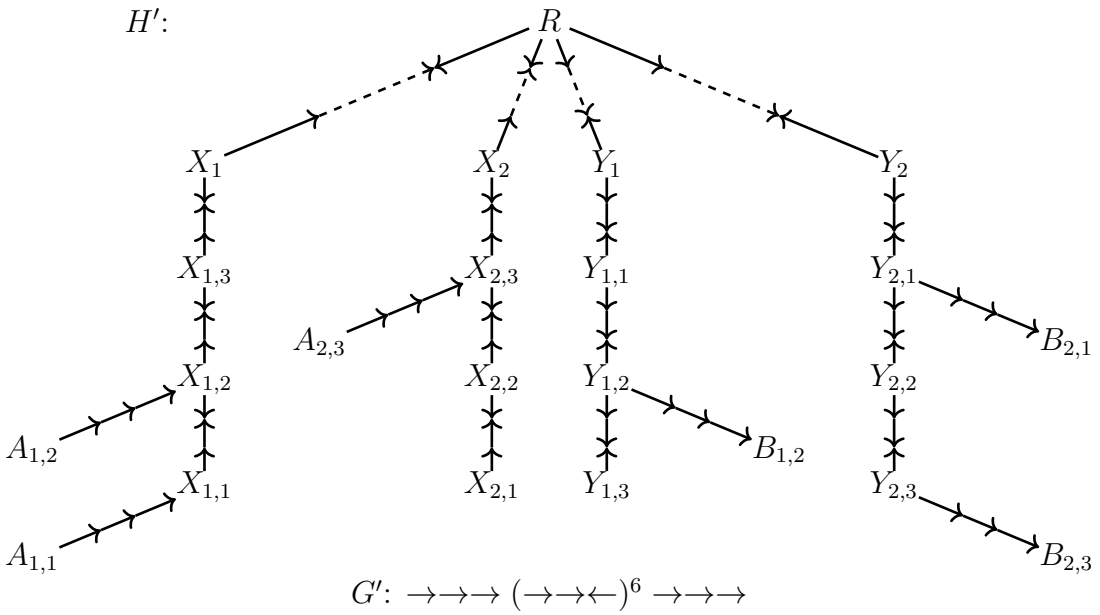

Figure 2.8 – Illustration of the proof of Proposition 2.5.4 for the PP2DNF formula $X_1Y_2 \vee X_1Y_1 \vee X_2Y_2$. Dashed edges have probability $\frac{1}{2}$, all others have probability 1.

- replace every edge $a \xrightarrow{T} b$ of H and G by 3 edges $a \rightarrow\rightarrow\rightarrow b$.

In particular, the query graph is then defined as follows:

$$G' := \rightarrow\rightarrow\rightarrow (\rightarrow\rightarrow\leftarrow)^{m+3} \rightarrow\rightarrow\rightarrow .$$

All the edges of H' have probability 1, except the middle edge of the paths that replaced the S-labeled edges used to code the valuation of the variables (e.g., for X_i, the middle edge of the 3 edges $X_i \rightarrow\rightarrow\leftarrow R$), which have probability $\frac{1}{2}$.

One can check that any image of G' must again go from the vertex $A_{x_j,j}$ to the vertex $B_{y_j,j}$ for some $1 \leqslant j \leqslant m$. The key insight is that the first $\rightarrow^5$ of G must be matched to a $\rightarrow^5$-path in H', which only exist as the concatenation of a $\rightarrow^3$ obtained by rewriting a T-edge for some variable X_j, and of the first $\rightarrow^2$ of the (undirected) path from $X_{x_j,j}$ to R. Then there is no choice left to match the next edges without failing.

From this we deduce that, from any possible world H'_{W} of the modified instance H', considering the corresponding possible world H_{W} of the unmodified instance H following the natural bijection, the modified query graph G' has a homomorphism to H'_{W} iff the unmodified query graph G has a homomorphism to H_{W}. We thus conclude that the probabilistic homomorphism problem on G' and H' has the same answer as the one on G and H, which finishes the proof. □

Conclusion. This concludes our study of PHom on all the graph classes than we considered. Our results are summarized in Tables 2.1, 2.2, and 2.3, and classify the combined complexity of the probabilistic graph homomorphism problem for all combinations of query and instance graph classes that we considered. It is apparent from these results that cases where the combined complexity of probabilistic query evaluation is PTIME are very restricted. In the next chapters, we relax our notion of combined tractability to capture more expressive queries and more general data.

Chapter 3

Fixed Parameter Tractability of Provenance Computation

This chapter of my thesis presents my work with Antoine Amarilli, Pierre Bourhis, and Pierre Senellart on the combined complexity of computing provenance information for Datalog queries on bounded treewidth instances. The results presented here were published at ICDT'2017 [Amarilli, Bourhis, Monet, and Senellart 2017] and in ToCS [Amarilli, Bourhis, Monet, and Senellart 2018].

3.1 Introduction

In this chapter we temporarily leave probabilistic evaluation aside and focus on the evaluation and computation of provenance of Boolean queries on non-probabilistic relational instances. We will see in Chapter 4 how to use our results for probabilistic query evaluation (PQE).

As we know, query evaluation is intractable in combined complexity [Vardi 1982] even for simple query languages such as conjunctive queries [Abiteboul, Hull, and Vianu 1995]. To address this issue, two main directions have been investigated. The first is to restrict the class of *queries* to ensure tractability, for instance, to α-acyclic conjunctive queries [Yannakakis 1981], this being motivated by the idea that many real-world queries are simple and usually small. The second approach restricts the structure of database instances, e.g., requiring them to have bounded *treewidth* [Robertson and Seymour 1986] (we call them *treelike*). This has been notably studied by Courcelle [Courcelle 1990], to show the tractability of monadic-second order logic on treelike instances, but in *data complexity* (i.e., for fixed queries); the combined complexity is generally nonelementary [Meyer 1975].

This leaves open the main question studied in this chapter: *Which queries can be efficiently evaluated, in combined complexity, on treelike databases?* This question has been addressed in [Gottlob, Pichler, and Wei 2010] by introducing *quasi-guarded Datalog*; however, an unusual feature of this language is that programs must explicitly refer to the tree decomposition of the instance. Instead, we try to follow Courcelle's approach and investigate which queries can be efficiently *translated to automata*. Specifically, rather than restricting to a fixed class of "efficient" queries, we study *parameterized* query classes, i.e., we define an efficient class of queries for each value of the parameter. We further make the standard assumption that the signature is fixed; in particular, its arity is constant. This allows us to aim for low combined

complexity for query evaluation, namely, fixed parameter tractability with linear time complexity in the product of the input query and instance, which we call *FPT-bilinear* complexity (recall Section 1.4).

Surprisingly, we are not aware of further existing work on tractable combined query evaluation for parameterized instances and queries, except from an unexpected angle: the translation of restricted query fragments to tree automata on treelike instances was used in the context of *guarded logics* and other fragments, to decide *satisfiability* [Benedikt, Bourhis, and Vanden Boom 2016] and *containment* [Barceló, Romero, and Vardi 2014]. To do this, one usually establishes a *treelike model property* to restrict the search to models of low treewidth (but dependent on the formula), and then translates the formula to an automaton, so that the problems reduce to emptiness testing: expressive automata formalisms, such as *alternating two-way automata*, are typically used. Exploiting this connection, we show how query evaluation on treelike instances can benefit from these ideas: for instance, as we show, some queries can only be translated efficiently to such concise automata, and not to the more common bottom-up tree automata.

Results. From there, the first main contribution of this chapter is to consider the language of *clique-frontier-guarded Datalog* (CFG-Datalog), and show an efficient FPT-linear translation procedure for this language, parameterized by the body size of rules: this implies FPT-bilinear combined complexity on treelike instances. While it is a Datalog fragment, CFG-Datalog shares some similarities with guarded logics; yet, its design incorporates several features (fixpoints, clique-guards, negation, guarding positive subformulas) that are not usually found together in guarded fragments, but are important for query evaluation. We show how the tractability of this language captures the tractability of such query classes as two-way regular path queries [Barceló 2013] and α-acyclic conjunctive queries. We further show that, in contrast with guarded negation logics, satisfiability of CFG-Datalog is undecidable.

Already for conjunctive queries, we show that the treewidth of queries is not the right parameter to ensure efficient translatability. In fact, the second contribution of this chapter is a lower bound: we show that bounded-treewidth queries cannot be efficiently translated to automata at all, so we cannot hope to show combined tractability for them via automata methods. By contrast, CFG-Datalog implies the combined tractability of bounded-treewidth queries with an additional requirement (interfaces between bags must be clique-guarded), which is the notion of *simplicial decompositions* previously studied in [Tarjan 1985]. To our knowledge, we are the first to introduce this query class and to show its tractability on treelike instances. CFG-Datalog can be understood as an extension of this fragment to disjunction, clique-guardedness, stratified negation, and inflationary fixpoints, that preserves tractability.

To derive our main FPT-bilinear combined complexity result, we define an operational semantics for our tree automata by introducing a notion of *cyclic provenance circuits*, that we call *cycluits*. These cycluits, the third contribution of the chapter, are well-suited as a provenance representation for alternating two-way automata encoding CFG-Datalog programs, as they naturally deal with both recursion and two-way traversal of a treelike instance, which is less straightforward with provenance formulas [Green, Karvounarakis, and Tannen 2007] or circuits [Deutch, Milo, Roy, and Tannen 2014]. While we believe that this natural generalization of Boolean

circuits may be of independent interest, it does not seem to have been studied in detail, except in the context of integrated circuit design [Malik 1993; Riedel and Bruck 2012], where the semantics often features feedback loops that involve negation; we prohibit these by focusing on *stratified* circuits, which we show can be evaluated in linear time. We show that the provenance of alternating two-way automata can be represented as a stratified cycluit in FPT-bilinear time, generalizing results on bottom-up automata and circuits from [Amarilli, Bourhis, and Senellart 2015].

Outline. We first position our approach relative to existing work in Section 3.2. We then present our tractable fragment, first for bounded-simplicial-width conjunctive queries in Section 3.3, then for CFG-Datalog in Section 3.4. We then define the automata variants that we use and translate CFG-Datalog to them in Section 3.5, before introducing cycluits and showing our provenance computation result in Section 3.6. We last present the proof of our translation result in Section 3.7.

3.2 Approaches for Tractability

We now review existing approaches to ensure the tractability of query evaluation, starting by query languages whose evaluation is tractable in combined complexity on *all* input instances. We then study more expressive query languages which are tractable on *treelike* instances, but where tractability only holds in data complexity. We then present the goals of the chapter.

3.2.1 Tractable Queries on All Instances

The best-known query language to ensure tractable query complexity is the language of *α-acyclic queries* [Fagin 1983], i.e., those CQs that have a tree decomposition where the domain of each bag corresponds exactly to an atom: this is called a *join tree* [Gottlob, Leone, and Scarcello 2002]. With Yannakakis's algorithm [Yannakakis 1981], we can evaluate an α-acyclic conjunctive query Q on an arbitrary instance I in time $O(|I| \cdot |Q|)$.

Yannakakis's result was generalized in two main directions. One direction [Gottlob, Greco, and Scarcello 2014] has investigated more general CQ classes, in particular CQs of bounded treewidth [Flum, Frick, and Grohe 2002], *hypertreewidth* [Gottlob, Leone, and Scarcello 2002], and *fractional hypertreewidth* [Grohe and Marx 2014]. Bounding these query parameters to some fixed k makes query evaluation run in time $O((|I| \cdot |Q|)^{f(k)})$ for some function f, hence in PTIME; for treewidth, since the decomposition can be computed in FPT-linear time (Theorem 1.5.2), this goes down to $O(|I|^k \cdot |Q|)$. However, query evaluation on arbitrary instances is unlikely to be FPT when parameterized by the query treewidth, since it would imply that deciding if a graph contains a k-clique is FPT parameterized by k, which is widely believed to be false in parameterized complexity theory (this is the W[1] $\neq$ FPT assumption). Further, even for treewidth 2 (e.g., triangles), it is not known if we can achieve linear data complexity [Alon, Yuster, and Zwick 1997].

In another direction, α-acyclicity has been generalized to queries with more expressive operators, e.g., disjunction or negation. The result on α-acyclic CQs thus extends to the *guarded fragment* (GF) of first-order logic, which can be evaluated on arbitrary instances in time $O(|I| \cdot |Q|)$ [Leinders, Marx, Tyszkiewicz, and Van den

Bussche 2005]. Tractability is independently known for FO^k, the fragment of FO where subformulas use at most k variables, with a simple evaluation algorithm in $O(|I|^k \cdot |Q|)$ [Vardi 1995].

Another important operator are *fixpoints*, which can be used to express, e.g., reachability queries. Though FO^k is no longer tractable when adding fixpoints [Vardi 1995], query evaluation is tractable for μGF [Berwanger and Grädel 2001, Theorem 3], i.e., GF with some restricted least and greatest fixpoint operators, when *alternation depth* is bounded; without alternation, the combined complexity is in $O(|I| \cdot |Q|)$. We could alternatively express fixpoints in Datalog, but, sadly, most known tractable fragments are nonrecursive: nonrecursive stratified Datalog is tractable [Flum, Frick, and Grohe 2002, Corollary 5.26] for rules with restricted bodies (i.e., strictly acyclic, or bounded strict treewidth). This result was generalized in [Gottlob, Leone, and Scarcello 2003] when bounding the number of guards: this nonrecursive fragment is shown to be equivalent to the k-guarded fragment of FO, with connections to the bounded-hypertreewidth approach. One recursive tractable fragment is Datalog LITE, which is equivalent to alternation-free μGF [Gottlob, Grädel, and Veith 2002]. Fixpoints were independently studied for graph query languages such as reachability queries and *regular path queries* (RPQ), which enjoy linear combined complexity on arbitrary input instances: this extends to two-way RPQs (2RPQs) and even strongly acyclic conjunctions of 2RPQs (SAC2RPQs), which are expressible in alternation-free μGF. Tractability also extends to acyclic C2RPQs but with PTIME complexity [Barceló 2013].

3.2.2 Tractability on Treelike Instances

We now study another approach for tractable query evaluation: this time, we restrict the shape of the *instances*, using treewidth. This ensures that we can translate them to a tree for efficient query evaluation, using tree automata techniques. Recall from Section 1.5 the definitions of treewidth, bottom-up tree automata, and tree encodings.

We say that a bNTA A *tests a query* Q *on instances of treewidth* $\leqslant k$ if, for any Γ_σ^k-encoding $\langle E, \lambda \rangle$ coding an instance I (of treewidth $\leqslant k$), A accepts $\langle E, \lambda \rangle$ iff $I \models Q$. By a well-known result of Courcelle on graphs ([Courcelle 1990], extended to higher-arity in [Flum, Frick, and Grohe 2002]), we can use bNTAs to evaluate all queries in *monadic second-order logic* (MSO), i.e., first-order logic with second-order variables of arity 1. MSO subsumes in particular CQs and monadic Datalog (but not general Datalog). Courcelle showed that MSO queries can be translated to a bNTA that tests them:

Theorem 3.2.1 ([Courcelle 1990; Flum, Frick, and Grohe 2002])**.** *For any MSO query* Q *and treewidth* $k \in \mathbb{N}$*, we can compute a bNTA that tests* Q *on instances of treewidth* $\leqslant k$*.*

This implies that evaluating any MSO query Q has FPT-linear *data complexity* when parameterized by Q and the instance treewidth [Courcelle 1990; Flum, Frick, and Grohe 2002], i.e., is in $O\left(f(|Q|, k) \cdot |I|\right)$ for some computable function f. However, this tells little about the combined complexity, as f is generally nonelementary in Q [Meyer 1975]. A better combined complexity bound is known for unions of conjunctions of two-way regular path queries (UC2RPQs) that are further required

to be acyclic and to have a constant number of edges between pairs of variables: these can be translated into polynomial-sized alternating two-way automata [Barceló, Romero, and Vardi 2014].

3.2.3 Restricted Queries on Treelike Instances

Our approach combines both ideas: we use instance treewidth as a parameter, but also restrict the queries to ensure tractable translatability. We are only aware of two approaches in this spirit. First, Gottlob, Pichler, and Wei have proposed a *quasiguarded* Datalog fragment on *relational structures and their tree decompositions*, for which query evaluation is in $O(|I| \cdot |Q|)$ [Gottlob, Pichler, and Wei 2010]. However, this formalism requires queries to be expressed in terms of the tree decomposition, and not just in terms of the relational signature. Second, Berwanger and Grädel [Berwanger and Grädel 2001] remark (after Theorem 4) that, when alternation depth and *width* are bounded, μCGF (the *clique-guarded* fragment of FO with fixpoints) enjoys FPT-linear query evaluation when parameterized by instance treewidth. Their approach does not rely on automata methods, and subsumes the tractability of α-acyclic CQs and alternation-free μGF (and hence SAC2RPQs), on treelike instances. However, μCGF is a restricted query language (the only CQs that it can express are those with a chordal primal graph), whereas we want a richer language, with a parameterized definition.

Our goal is thus to develop an expressive parameterized query language, which can be translated in *FPT-linear time* to an automaton that tests it (with the treewidth of instances also being a parameter). We can then evaluate the automaton, and obtain FPT-bilinear combined complexity for query evaluation. Further, as we will show, the use of tree automata will yield *provenance representations* for the query as in [Amarilli, Bourhis, and Senellart 2015] (see Section 3.6).

3.3 Conjunctive Queries on Treelike Instances

To identify classes of queries that can be efficiently translated to tree automata, we start by the simplest queries: *conjunctive queries*.

α-acyclic queries. A natural candidate for a tractable query class via automata methods would be *α-acyclic* CQs, which, as we explained in Section 3.2.1, can be evaluated in time $O(|I| \cdot |Q|)$ on all instances. Sadly, we show that such queries cannot be translated efficiently to bNTAs, so the translation result of Theorem 3.2.1 does not extend directly:

Proposition 3.3.1. *There is an arity-two signature σ and an infinite family $(Q_i)_{i\in\mathbb{N}}$ of α-acyclic CQs such that, for any $i \in \mathbb{N}$, any bNTA that tests Q_i on instances of treewidth $\leqslant 1$ must have $\Omega(2^{|Q_i|^{1-\varepsilon}})$ states for any $\varepsilon > 0$.*

Proof. We fix the signature σ to consist of binary relations S, S_0, S_1, and C. We will code binary numbers as gadgets on this fixed signature. The coding of $i \in \mathbb{N}$ at length k, with $k \geqslant 1 + \lceil \log_2 i \rceil$, consists of an S-chain $S(a_1, a_2), \ldots, S(a_{k-1}, a_k)$, and facts $S_{b_j}(a_{j+1}, a'_{j+1})$ for $1 \leqslant j \leqslant k-1$ where a'_{j+1} is a fresh element and b_j is the j-th bit in the binary expression of i (padding the most significant bits with 0). We

now define the query family Q_i: each Q_i is formed by picking a root variable x and gluing 2^i chains to x; for $0 \leqslant j \leqslant 2^i - 1$, we have one chain that is the concatenation of a chain of C of length i and the coding of j at length $(i+1)$ using a gadget. Clearly the size of Q_i is $\Theta(i \times 2^i)$. Now, fix $\varepsilon > 0$ As $|Q_i|$ is in $O(i \times 2^i)$, there exist $N, \beta > 0$ such that $\forall i \geqslant N,\ |Q_i| \leqslant \beta \times i \times 2^i$. But clearly, there exists $M > 0$ such that $\forall i \geqslant M,\ (\beta \times i \times 2^i)^{1-\varepsilon} \leqslant 2^i - i/2$. Hence for $i \geqslant \max(N, M)$ we have $2^i - i/2 \geqslant |Q_i|^{1-\varepsilon}$.

Fix $i > 0$. Let A be a bNTA testing Q_i on instances of treewidth 1. We will show that A must have at least $\binom{2^i}{2^{i-1}} = \Omega\left(2^{2^i - \frac{i}{2}}\right)$ states (the lower bound is obtained from Stirling's formula), from which the claim follows since for $i \geqslant \max(N, M)$ we have $2^{2^i - \frac{i}{2}} \geqslant 2^{|Q_i|^{1-\varepsilon}}$. In fact, we will consider a specific subset $\mathcal{I}$ of the instances of treewidth $\leqslant 1$, and a specific set $\mathcal{E}$ of tree encodings of instances of $\mathcal{I}$, and show the claim on $\mathcal{E}$, which suffices to conclude.

To define $\mathcal{I}$, let $\mathcal{S}_i$ be the set of subsets of $\{0, \ldots, 2^i - 1\}$ of cardinality 2^{i-1}, so that $|\mathcal{S}_i|$ is $\binom{2^i}{2^{i-1}}$. We will first define a family $\mathcal{I}'$ of instances indexed by $\mathcal{S}_i$ as follows. Given $S \in \mathcal{S}_i$, the instance I'_S of $\mathcal{I}'$ is obtained by constructing a full binary tree of the C-relation of height $i-1$, and identifying, for all j, the j-th leaf node with element a_1 of the length-$(i+1)$ coding of the j-th smallest number in S. We now define the instances of $\mathcal{I}$ to consist of a root element with two C-children, each of which are the root element of an instance of $\mathcal{I}'$ (we call the two the *child instances*). It is clear that instances of $\mathcal{I}$ have treewidth 1, and we can check quite easily that an instance of $\mathcal{I}$ satisfies Q_i iff the child instances I'_{S_1} and I'_{S_2} are such that $S_1 \cup S_2 = \{1, \ldots, 2^i\}$.

We now define $\mathcal{E}$ to be tree encodings of instances of $\mathcal{I}$. First, define $\mathcal{E}'$ to consist of tree encodings of instances of $\mathcal{I}'$, which we will also index with $\mathcal{S}_i$, i.e., E_S is a tree encoding of I'_S. We now define $\mathcal{E}$ as the tree encodings E constructed as follows: given an instance $I \in \mathcal{I}$, we encode it as a root bag with domain $\{r\}$, where r is the root of the tree I, and no fact, the first child n_1 of the root bag having domain $\{r, r_1\}$ and fact $C(r, r_1)$, the second child n_2 of the root being defined in the same way. Now, n_1 has one dummy child with empty domain and no fact, and one child which is the root of some tree encoding in $\mathcal{E}$ of one child instance of I. We define n_2 analogously with the other child instance.

For each $S \in \mathcal{S}_i$, letting $\bar{S}$ be the complement of S relative to $\{0, \ldots, 2^i - 1\}$, we call $I_S \in \mathcal{I}$ the instance where the first child instance is I'_S and the second child instance is $I'_{\bar{S}}$, and we call $E_S \in \mathcal{E}$ the tree encoding of I_S according to the definition above. We then call $\mathcal{Q}_S$ the set of states q of A such that there exists a run of A on E_S where the root of the encoding of the first child instance is mapped to q. As each I_S satisfies Q, each E_S should be accepted by the automaton, so each $\mathcal{Q}_S$ is non-empty.

Further, we show that the $\mathcal{Q}_S$ are pairwise disjoint: for any $S_1 \neq S_2$ of $\mathcal{S}_i$, we show that $\mathcal{Q}_{S_1} \cap \mathcal{Q}_{S_2} = \emptyset$. Assume to the contrary the existence of q in the intersection, and let ρ_{S_1} and ρ_{S_2} be runs of A respectively on I_{S_1} and I_{S_2} that witness respectively that $q \in \mathcal{Q}_{S_1}$ and $q \in \mathcal{Q}_{S_2}$. Now, consider the instance $I \in \mathcal{I}$ where the first child instance is I_1, and the second child instance is $\bar{I_2}$, and let $E \in \mathcal{E}$ be the tree encoding of I. We can construct a run ρ of A on E by defining ρ according to ρ_{S_2} except that, on the subtree of E rooted at the root r' of the tree encoding of the first child instance, ρ is defined according to ρ_{S_1}: this is possible because ρ_{S_1} and ρ_{S_2} agree on r'_1 as they both map r' to q. Hence, ρ witnesses that A accepts E. Yet, as $I_1 \neq I_2$,

we know that I does not satisfy Q, so that, letting $E \in \mathcal{E}$ be its tree encoding, A rejects E. We have reached a contradiction, so indeed the $\mathcal{Q}_S$ are pairwise disjoint.

As the $\mathcal{Q}_S$ are non-empty, we can construct a mapping from $\mathcal{S}_i$ to the state set of A by mapping each $S \in \mathcal{S}_i$ to some state of $\mathcal{Q}_S$: as the $\mathcal{Q}_S$ are pairwise disjoint, this mapping is injective. We deduce that the state set of A has size at least $|\mathcal{S}_i|$, which concludes from the bound on the size of $\mathcal{S}_i$ that we showed previously. □

Faced by this, we propose to use different tree automata formalisms, which are generally more concise than bNTAs. There are two classical generalizations of nondeterministic automata, on words [Birget 1993] and on trees [Comon et al. 2007]: one goes from the inherent existential quantification of nondeterminism to *quantifier alternation*; the other allows *two-way* navigation instead of imposing a left-to-right (on words) or bottom-up (on trees) traversal. On words, both of these extensions independently allow for exponentially more compact automata [Birget 1993]. In this chapter, we combine both extensions and use *alternating two-way tree automata* [Comon et al. 2007; Cachat 2002], formally introduced in Section 3.5, which leads to tractable combined complexity for evaluation. Our general results in the next section will then imply:

Proposition 3.3.2. *For any treewidth bound $k_{\mathrm{I}} \in \mathbb{N}$, given an α-acyclic CQ Q, we can compute in FPT-linear time in $O(|Q|)$ (parameterized by k_{I}) an alternating two-way tree automaton that tests it on instances of treewidth $\leqslant k_{\mathrm{I}}$.*

Hence, if we are additionally given a relational instance I of treewidth $\leqslant k_{\mathrm{I}}$, one can determine whether $I \models Q$ in FPT-bilinear time in $|I| \cdot |Q|$ (parameterized by k_{I}).

Proof. This proof depends on notions and results that are given in the rest of the chapter, and should be read after studying the rest of this chapter.

Given the α-acyclic CQ Q, we can compute in linear time in Q a chordal decomposition T (also called a join tree) of Q by using Theorem 5.6 of [Flum, Frick, and Grohe 2002] (attributed to [Tarjan and Yannakakis 1984]). We recall that a chordal decomposition of a hypergraph $\mathcal{H}$ is a tree decomposition T of $\mathcal{H}$ such that for every bag b of T, there exists a hyperedge e of $\mathcal{H}$ such that $e = b$. As T is in particular a simplicial decomposition of Q of width $\leqslant \mathrm{arity}(\sigma) - 1$, i.e., of constant width, we use Proposition 3.4.5 to obtain in linear time in $|Q|$ a CFG-Datalog program P equivalent to Q with body size bounded by a constant k_{P}.

We now use Theorem 3.5.4 to construct, in FPT-linear time in $|P|$ (hence, in $|Q|$), parameterized by k_{I} and the constant k_{P}, an automaton A testing P on instances of treewidth $\leqslant k_{\mathrm{I}}$; specifically, a stratified isotropic alternating two-way automata or SATWA (to be introduced in Definition 3.5.1).

We now observe that, thanks to the fact that Q is monotone, the SATWA A does not actually feature any negation: the translation in the proof of Proposition 3.4.5 does not produce any negated atom, and the translation in the proof of Theorem 3.5.4 only produces a negated state within a Boolean formula when there is a corresponding negated atom in the Datalog program. Hence, A is actually an alternating two-way tree automaton, which proves the first part of the claim.

For the second part of the claim, we use Theorem 3.4.4 to evaluate P on I in FPT-bilinear time in $|I| \cdot |P|$, parameterized by the constant k_{P} and k_{I}. This proves the claim. □

Bounded-treewidth queries. Having re-proven the combined tractability of α-acyclic queries (on bounded-treewidth instances), we naturally try to extend to *bounded-treewidth* CQs. Recall from Section 3.2.1 that these queries have PTIME combined complexity on all instances, but are unlikely to be FPT when parameterized by the query treewidth (unless W[1] = FPT). Can they be efficiently evaluated on *treelike* instances by translating them to automata? We answer in the negative: that bounded-treewidth CQs *cannot* be efficiently translated to automata to test them, even when using the expressive formalism of alternating two-way tree automata:

Theorem 3.3.3. *There is an arity-two signature σ for which there is no algorithm $\mathcal{A}$ with exponential running time and polynomial output size for the following task: given a conjunctive query Q of treewidth $\leqslant 2$, produce an alternating two-way tree automaton A_Q on Γ_σ^5-trees that tests Q on σ-instances of treewidth $\leqslant 5$.*

This result is obtained from a variant of the 2EXPTIME-hardness of monadic Datalog containment [Benedikt, Bourhis, and Senellart 2012]. We do not prove it here, as the proof is technical and not directly related to our PhD research; it can be found in the extended version of [Amarilli, Bourhis, Monet, and Senellart 2017].

Bounded simplicial width. We have shown that we cannot translate bounded-treewidth queries to automata efficiently. We now show that efficient translation can be ensured with an additional requirement on tree decompositions. As it turns out, the resulting decomposition notion has been independently introduced for graphs:

Definition 3.3.4 ([Diestel 1989])**.** A *simplicial decomposition of a graph G* is a tree decomposition T of G such that, for any bag b of T and child bag b' of b, letting S be the intersection of the domains of b and b', then the subgraph of G induced by S is a complete subgraph of G. ◁

We extend this notion to CQs, and introduce the *simplicial width* measure:

Definition 3.3.5. A *simplicial decomposition of a CQ Q* is a simplicial decomposition of its primal graph. Note that any CQ has a simplicial decomposition (e.g., the trivial one that puts all variables in one bag). The *simplicial width* of Q is the minimum, over all simplicial tree decompositions, of the size of the largest bag minus 1. ◁

Bounding the simplicial width of CQs is of course more restrictive than bounding their treewidth, and this containment relation is strict: cycles have treewidth $\leqslant 2$ but have unbounded simplicial width. This being said, bounding the simplicial width is less restrictive than imposing α-acyclicity: the join tree of an α-acyclic CQ is in particular a simplicial decomposition, so α-acyclic CQs have simplicial width at most $\text{arity}(\sigma) - 1$, which is constant as σ is fixed. Again, the containment is strict: a triangle has simplicial width 2 but is not α-acyclic.

To our knowledge, simplicial width for CQs has not been studied before. Yet, we show that bounding the simplicial width ensures that CQs can be efficiently translated to automata. This is in contrast to bounding the treewidth, which we have shown in Theorem 3.3.3 not to be sufficient to ensure efficient translatbility to tree automata. Hence:

Theorem 3.3.6. *For any $k_\text{I}, k_\text{Q} \in \mathbb{N}$, given a CQ Q and a simplicial decomposition T of simplicial width k_Q of Q, we can compute in FPT-linear in $|Q|$ (parameterized*

by k_{I} and k_{Q}) an alternating two-way tree automaton that tests Q on instances of treewidth $\leqslant k_{\mathrm{I}}$.

Hence, if we are additionally given a relational instance I of treewidth $\leqslant k_{\mathrm{I}}$, one can determine whether $I \models Q$ in FPT-bilinear time in $|I| \cdot (|Q| + |T|)$ (parameterized by k_{I} and k_{Q}).

Proof. This proof depends on notions and results that are given in the rest of the chapter, and should be read after studying the rest of this chapter.

We use Proposition 3.4.5 to transform the CQ Q to a CFG-Datalog program P with body size at most $k_{\mathrm{P}} := f_\sigma(k_{\mathrm{Q}})$, in FPT-linear time in $|Q| + |T|$ parameterized by k_{Q}.

We now use Theorem 3.5.4 to construct, in FPT-linear time in $|P|$ (hence, in $|Q|$), parameterized by k_{I} and k_{P}, hence in k_{I} and k_{Q}, a SATWA A testing P on instances of treewidth $\leqslant k_{\mathrm{I}}$ (see Definition 3.5.1). For the same reasons as in the proof of Proposition 3.3.2, it is actually a two-way alternating tree automaton, so we have shown the first part of the result.

To prove the second part of the result, we now use Theorem 3.4.4 to evaluate P on I in FPT-bilinear time in $|I| \cdot |P|$, parameterized by k_{P} and k_{I}, hence again by k_{Q} and k_{I}. This proves the claim. □

Notice the technicality that the simplicial decomposition T must be provided as input to the procedure, because it is not known to be computable in FPT-linear time, unlike tree decompositions. While we are not aware of results on the complexity of this specific task, quadratic-time algorithms are known for the related problem of computing the *clique-minimal separator decomposition* [Leimer 1993; Berry, Pogorelcnik, and Simonet 2010].

The intuition for the efficient translation of bounded-simplicial-width CQs is as follows. The *interface* variables shared between any bag and its parent must be "clique-guarded" (each pair is covered by an atom). Hence, consider any subquery rooted at a bag of the query decomposition, and see it as a non-Boolean CQ with the interface variables as free variables. Each result of this CQ must then be covered by a clique of facts of the instance, which ensures [Gavril 1974] that it occurs in some bag of the instance tree decomposition and can be "seen" by a tree automaton. This intuition can be generalized, beyond conjunctive queries, to design an expressive query language featuring disjunction, negation, and fixpoint, with the same properties of efficient translation to automata and FPT-linear combined complexity of evaluation on treelike instances. We introduce such a Datalog variant in the next section.

3.4 CFG-Datalog on Treelike Instances

To design a Datalog fragment with efficient translation to automata, we must of course impose some limitations, as we did for CQs. In fact, we can even show that the full Datalog language (even without negation) *cannot* be translated to automata, no matter the complexity:

Proposition 3.4.1. *There is a signature σ and Datalog program P such that the language of Γ^1_σ-trees that encode instances satisfying P is not a regular tree language.*

Proof. Let σ be the signature containing two binary relations Y and Z and two unary relations Begin and End. Consider the following program P:

$$\begin{aligned}
\text{Goal}() &\leftarrow S(x,y), \text{Begin}(x), \text{End}(y)\\
S(x,y) &\leftarrow Y(x,w), S(w,u), Z(u,y)\\
S(x,y) &\leftarrow Y(x,w), Z(w,y)
\end{aligned}$$

Let L be the language of the tree encodings of instances of treewidth 1 that satisfy P. We will show that L is not a regular tree language, which clearly implies the second claim, as a bNTA or an alternating two-way tree automaton can only recognize regular tree languages [Comon et al. 2007]. To show this, let us assume by contradiction that L is a regular tree language, so that there exists a Γ^1_σ-bNTA A that accepts L, i.e., that tests P.

We consider instances that are chains of facts which are either Y- or Z-facts, and where the first end is the only node labeled Begin and the other end is the only node labeled End. This condition on instances can clearly be expressed in MSO, so that by Theorem 3.2.1 there exists a bNTA A_{chain} on Γ^1_σ that tests this property. In particular, we can build the bNTA A' which is the intersection of A and A_{chain}, which tests whether instances are of the prescribed form and are accepted by the program P.

We now observe that such instances must be the instance

$$\begin{aligned}
I_k = \{&\text{Begin}(a_1), Y(a_1,a_2), \ldots, Y(a_{k-1},a_k), Y(a_k,a_{k+1}),\\
&Z(a_{k+1},a_{k+2}), \ldots, Z(a_{2k-1},a_{2k}), Z(a_{2k},a_{2k+1}), \text{End}(a_{2k+1})\}
\end{aligned}$$

for some $k \in \mathbb{N}$. Indeed, it is clear that I_k satisfies P for all $k \in \mathbb{N}$, as we derive the facts

$$S(a_k,a_{k+2}), S(a_{k-1},a_{k+3}), \ldots, S(a_{k-(k-1)},a_{k+2+(k-1)}), \text{ that is, } S(a_1,a_{2k+1}),$$

and finally Goal(). Conversely, for any instance I of the prescribed shape that satisfies P, it is easily seen that the derivation of Goal justifies the existence of a chain in I of the form I_k, which by the restrictions on the shape of I means that $I = I_k$.

We further restrict our attention to tree encodings that consist of a single branch of a specific form, namely, their contents are as follows (given from leaf to root) for some integer $n \geqslant 0$: $(\{a_1\}, \text{Begin}(a_1))$, $(\{a_1,a_2\}, X(a_1,a_2))$, $(\{a_2,a_3\}, X(a_2,a_3))$, $(\{a_3,a_1\}, X(a_3,a_1))$, $\ldots$, $(\{a_n,a_{n+1}\}, X(a_n,a_{n+1}))$, $(\{a_{n+1}\}, \text{End}(a_{n+1}))$, where we write X to mean that we may match either Y or Z, where addition is modulo 3, and where we add dummy nodes $(\bot,\bot)$ as left children of all nodes, and as right children of the leaf node $(\{a_1\}, \text{Begin}(a_1))$, to ensure that the tree is full. It is clear that we can design a bNTA A_{encode} which recognizes tree encodings of this form, and we define A'' to be the intersection of A' and A_{encode}. In other words, A'' further enforces that the Γ^1_σ-tree encodes the input instance as a chain of consecutive facts with a certain prescribed alternation pattern for elements, with the Begin end of the chain at the top and the End end at the bottom.

Now, it is easily seen that there is exactly one tree encoding of every I_k which is accepted by A'', namely, the one of the form tested by A_{encode} where $n = 2k$, the first k X are matched to Y and the last k X are matched to Z.

Now, we observe that as A'' is a bNTA which is forced to operate on chains (completed to full binary trees by a specific addition of binary nodes). Thus, we can translate it to a deterministic automaton A''' on words on the alphabet $\Sigma = \{B, Y, Z, E\}$, by looking at its behavior in terms of the X-facts. Formally, A''' has same state space as A'', same final states, initial state $\delta(\iota((\bot, \bot)), \iota((\bot, \bot)))$ and transition function $\delta(q, x) = \delta(\iota((\bot, \bot)), q, (s, f))$ for every domain s, where f is a fact corresponding to the letter $x \in \Sigma$ (B stands here for Begin, and E for End). By definition of A'', the automaton A''' on words recognizes the language $\{BY^kZ^kE \mid k \in \mathbb{N}\}$. However, this language is not regular. This contradicts our hypothesis about the existence of automaton A, which establishes the desired result. □

Hence, there is no bNTA or alternating two-way tree automaton that tests P for treewidth 1. To work around this problem and ensure that translation is possible and efficient, the key condition that we impose on Datalog programs, pursuant to the intuition of simplicial decompositions, is that the rules must be *clique-frontier-guarded*, i.e., the variables in the head must co-occur in *positive* predicates of the rule body. We can then use the *body size* of the program rules as a parameter, and will show that the fragment can then be translated to automata in FPT-linear time. Remember that we assume that the arity of the extensional signature is fixed.

Definition 3.4.2. Let P be a stratified Datalog program. A rule r of P is *clique-frontier-guarded* if for any two variables $x_i \neq x_j$ in the head of r, we have that x_i and x_j co-occur in some positive (extensional or intensional) predicate of the body of r. P is *clique-frontier-guarded* (CFG) if all its rules are clique-frontier-guarded. The *body size* of P is the maximal number of atoms in the body of its rules, multiplied by its arity. ◁

Example 3.4.3. Let P be the stratified Datalog program from Example 1.2.1. We recall that $\sigma = \{R\}$, and that P tests if there are two elements that are not connected by a directed R-path. Then P is not a CFG-Datalog program, since the rule $T(x, y) \leftarrow R(x, z) \wedge T(z, y)$ is not clique-frontier-guarded. In fact, it is easy to show that no CFG-Datalog program can contain a binary intensional predicate T computing the transitive closure of an extensional binary relation R.

However, it is possible to express a similar query in CFG-Datalog. Let σ be $\{R, A, B\}$, with R being binary, A and B being unary. Let $\sigma_{\text{int}} = \{T\}$, with T unary. Consider the stratified Datalog program P' with two strata P'_1 and P'_2. The stratum P'_1 contains the following two rules:

- $T(x) \leftarrow A(x)$
- $T(y) \leftarrow T(x) \wedge R(x, y)$

And P'_2 contains the rule:

- $\text{Goal}() \leftarrow A(x) \wedge B(y) \wedge \neg T(y)$

Then P' is a CFG-Datalog program (of body size $3 \times 2 = 6$). Moreover, P' tests if there exist two distinct elements $a \neq b$ such that $A(a)$ and $B(b)$ hold, and such that a and b are not connected by a directed R-path. ◁

We will see later in this section what interesting query languages CFG-Datalog capture: (Boolean) CQs, (Boolean) 2RPQs, (Boolean) SAC2RPQs, guarded negation logics, monadic Datalog, etc.

The main result of this chapter is that evaluation of CFG-Datalog is *FPT-bilinear* in *combined complexity*, when parameterized by the body size of the program and the instance treewidth.

Theorem 3.4.4. *Given a CFG-Datalog program P of body size k_{P} and a relational instance I of treewidth k_{I}, checking if $I \models P$ is FPT-bilinear time in $|I| \cdot |P|$ (parameterized by k_{P} and k_{I}).*

We will show this result in the next section by translating CFG-Datalog programs in FPT-linear time to a special kind of tree automata (Theorem 3.5.4), and showing in Section 3.6 that we can efficiently evaluate such automata and even compute *provenance representations*. The rest of this section presents consequences of our main result for various languages.

Conjunctive queries. Our tractability result for bounded-simplicial-width CQs (Theorem 3.3.6), including α-acyclic CQs, is shown by rewriting to CFG-Datalog of bounded body size:

Proposition 3.4.5. *There is a function f_σ (depending only on σ) such that for all $k \in \mathbb{N}$, for any conjunctive query Q and simplicial tree decomposition T of Q of width at most k, we can compute in $O(|Q| + |T|)$ an equivalent CFG-Datalog program with body size at most $f_\sigma(k)$.*

To prove Proposition 3.4.5, we first prove the following lemma about simplicial tree decompositions:

Lemma 3.4.6. *For any simplicial decomposition T of width k of a query Q, we can compute in linear time a simplicial decomposition T_{bounded} of Q such that each bag has degree at most 2^{k+1}.*

Proof. Fix Q and T. We construct the simplicial decomposition T_{bounded} of Q in a process which shares some similarity with the routine rewriting of tree decompositions to make them binary, by creating copies of bags. However, the process is more intricate because we need to preserve the fact that we have a *simplicial* tree decomposition, where interfaces are guarded.

We go over T bottom-up: for each bag b of T, we create a bag b' of T_{bounded} with same domain as b. Now, we partition the children of b depending on their intersection with b: for every subset S of the domain of b such that b has some children whose intersection with b is equal to S, we write these children $b_{S,1}, \ldots, b_{S,n_S}$ (so we have $S = \mathrm{dom}(b) \cap \mathrm{dom}(b_{S,j})$ for all $1 \leqslant j \leqslant n_S$), and we write $b'_{S,1}, \ldots, b'_{S,n_S}$ for the copies that we already created for these bags in T_{bounded}. Now, for each S, we create n_S fresh bags $b'_{=S,j}$ in T_{bounded} (for $1 \leqslant j \leqslant n_S$) with domain equal to S, and we set $b'_{=S,1}$ to be a child of b', $b'_{=S,j+1}$ to be a child of $b'_{=S,j}$ for all $1 \leqslant j < n_S$, and we set each $b'_{S,i}$ to be a child of $b'_{=S,i}$.

This process can clearly be performed in linear time. Now, the degree of the fresh bags in T_{bounded} is at most 2, and the degree of the copies of the original bags is at most 2^{k+1}, as stated. Further, it is clear that the result is still a tree

decomposition (each fact is still covered, the occurrences of each element still form a connected subtree because they are as in T with the addition of some paths of the fresh bags), and the interfaces in T_{bounded} are the same as in T, so they still satisfy the requirement of simplicial decompositions. □

We can now prove Proposition 3.4.5. In fact, as will be easy to notice from the proof, our construction further ensures that the equivalent CFG-Datalog program is positive, nonrecursive, and conjunctive. Recall that a Datalog program is *positive* if it contains no negated atoms. It is *nonrecursive* if there is no cycle in the directed graph on σ_{int} having an edge from R to S whenever a rule contains R in its head and S in its body. It is *conjunctive* [Benedikt and Gottlob 2010] if each intensional relation R occurs in the head of at most one rule.

Proof of Proposition 3.4.5. Using Lemma 3.4.6, we can start by rewriting in linear time the input simplicial decomposition to ensure that each bag has degree at most 2^{k+1}. Hence, let us assume without loss of generality that T has this property. We further add an empty root bag if necessary to ensure that the root bag of T is empty and has exactly one child.

We start by using Lemma 3.1 of [Flum, Frick, and Grohe 2002] to annotate in linear time each node b of T by the set of atoms $\mathcal{A}_b$ of Q whose free variables are in the domain of b and such that for each atom A of $\mathcal{A}_b$, b is the topmost bag of T which contains all the variables of A. As the signature σ is fixed, note that we have $|\mathcal{A}_b| \leqslant g_\sigma(k)$ for some function g_σ depending only on σ.

We now perform a process similar to Lemma 3.1 of [Flum, Frick, and Grohe 2002]. We start by precomputing in linear time a mapping μ that associates, to each pair $\{x, y\}$ of variables of Q, the set of all atoms in Q where $\{x, y\}$ co-occur. We can compute μ in linear time by processing all atoms of Q and adding each atom as an image of μ for each pair of variables that it contains (remember that the arity of σ is constant). Now, we do the following computation: for each bag b which is not the root of T, letting S be its interface with its parent bag, we annotate b by a set of atoms $\mathcal{A}_b^{\text{guard}}$ defined as follows: for all $x, y \in S$ with $x \neq y$, letting $A(\mathbf{z})$ be an atom of Q where x and y appear (which must exist, by the requirement on simplicial decompositions, and which we retrieve from μ), we add $A(\mathbf{w})$ to $\mathcal{A}_b^{\text{guard}}$, where, for $1 \leqslant i \leqslant |\mathbf{z}|$, we set $w_i := z_i$ if $z_i \in \{x, y\}$, and w_i to be a fresh variable otherwise. In other words, $\mathcal{A}_b^{\text{guard}}$ is a set of atoms that ensures that the interface S of b with its parent is covered by a clique, and we construct it by picking atoms of Q that witness the fact that it is guarded (which it is, because T is a simplicial decomposition), and replacing their irrelevant variables to be fresh. Note that $\mathcal{A}_b^{\text{guard}}$ consists of at most $k \times (k+1)/2$ atoms, but the domain of these atoms is not a subset of $\text{dom}(b)$ (because they include fresh variables). This entire computation is performed in linear time.

We now define the function $f_\sigma(k)$ as follows, remembering that $\text{arity}(\sigma)$ denotes the arity of the *extensional* signature:

$$f_\sigma(k) := (k+1) \times \left(g_\sigma(k) + 2^{k+1} + k(k+1)/2\right).$$

We now build our CFG-Datalog program P of body size $f_\sigma(k)$ which is equivalent to Q. We define the intensional signature σ_{int} by creating one intensional predicate P_b for each non-root bag b of T, whose arity is the size of the intersection of b with its parent. As we ensured that the root bag b_{r} of T is empty and has exactly one child

b'_{r}, we use $P_{b'_{\mathrm{r}}}$ as our 0-ary Goal() predicate (because its interface with its parent b_{r} is necessarily empty). We now define the rules of P by processing T bottom-up: for each bag b of T, we add one rule ρ_b with head $P_b(\mathbf{x})$, defined as follows:

- If b is a leaf, then ρ_b is $P_b \leftarrow \bigwedge \mathcal{A}_b^{\mathrm{guard}} \wedge \bigwedge \mathcal{A}_b$.
- If b is an internal node with children $b_1, \ldots, b_m$ (remember that $m \leqslant 2^{k+1}$), then ρ_b is $P_b \leftarrow \bigwedge \mathcal{A}_b^{\mathrm{guard}} \wedge \bigwedge \mathcal{A}_b \wedge \bigwedge_{1 \leqslant i \leqslant m} P_{b_i}$.

We first check that P is clique-frontier-guarded, but this is the case because by construction the conjunction of atoms $\bigwedge \mathcal{A}_b^{\mathrm{guard}}$ is a suitable guard for $\mathbf{x}$: for each $\{x, y\} \in \mathbf{x}$, it contains an atom where both x and y occur.

Second, we check that the body size of P is indeed $f_\sigma(k)$. It is clear that $\mathrm{arity}(P) = \mathrm{arity}(\sigma_{\mathrm{int}} \cup \sigma) \leqslant k + 1$. Further, the maximal number of atoms in the body of a rule is $g_\sigma(k) + 2^{k+1} + k(k+1)/2$, so we obtain the desired bound.

What is left to check is that P is equivalent to Q. It will be helpful to reason about P by seeing it as the conjunctive query Q' obtained by recursively inlining the definition of rules: observe that this a conjunctive query, because P is conjunctive, i.e., for each intensional atom P_b, the rule ρ_b is the only one where P_b occurs as head atom. It is clear that P and Q' are equivalent, so we must prove that Q and Q' are equivalent.

For the forward direction, it is obvious that Q' implies Q, because Q' contains every atom of Q by construction of the $\mathcal{A}_b$. For the backward direction, noting that the only atoms of Q' that are not in Q are those added in the sets $\mathcal{A}_b^{\mathrm{guard}}$, we observe that there is a homomorphism from Q' to Q defined by mapping each atom $A(\mathbf{w})$ occurring in some $\mathcal{A}_b^{\mathrm{guard}}$ to the atom $A(\mathbf{z})$ of Q used to create it; this mapping is the identity on the two variables x and y used to create $A(\mathbf{w})$, and maps each fresh variables w_i to z_i: the fact that these variables are fresh ensures that this homomorphism is well-defined. This shows Q and Q', hence P, to be equivalent, which concludes the proof. □

Proposition 3.4.5 implies that CFG-Datalog can express any CQ up to increasing the body size parameter (since any CQ has a simplicial decomposition), unlike, e.g., μCGF[Berwanger and Grädel 2001]. Conversely, we can show that bounded-simplicial-width CQs *characterize* the queries expressible in CFG-Datalog when disallowing negation, recursion, and disjunction.

Proposition 3.4.7. *For any positive, conjunctive, nonrecursive CFG-Datalog program P with body size k, there is a CQ Q of simplicial width $\leqslant k$ that is equivalent to P.*

To prove Proposition 3.4.7, we will use the notion of *call graph* of a Datalog program. This is the graph G on the relations of σ_{int} which has an edge from R to S whenever a rule contains relation R in its head and S in its body. From the requirement that P is nonrecursive, we know that this graph G is a DAG.

Proof of Proposition 3.4.7. We first check that every intensional relation reachable from Goal in the call graph G of P appears in the head of a rule of P (as P is conjunctive, this rule is then unique). Otherwise, it is clear that P is not satisfiable (it has no derivation tree), so we can simply rewrite P to the query False. We also

assume without loss of generality that each intensional relation except Goal() occurs in the body of some rule, as otherwise we can simply drop them and drop all rules where they appear as the head relation.

In the rest of the proof we will consider the rules of P in some order, and create an equivalent CFG-Datalog program P' with rules $r'_0, \ldots, r'_m$. We will ensure that P' is also positive, conjunctive, and nonrecursive, and that it further satisfies the following additional properties:

1. Every intensional relation other than Goal appears in the body of exactly one rule of P', and appears there exactly once;

2. For every $0 \leqslant i \leqslant m$, for every variable z in the body of rule r'_i that does not occur in its head, then for every $0 \leqslant j < i$, z does not occur in r'_j.

We initialize a queue that contains only the one rule that defines Goal in P, and we do the following until the queue is empty:

- Pop a rule r from the queue. Let r' be defined from r as follows: for every intensional relation R that occurs in the body of r', letting $R(\mathbf{x^1}), \ldots, R(\mathbf{x^n})$ be its occurrences, rewrite these atoms to $R^1(\mathbf{x^1}), \ldots, R^n(\mathbf{x^n})$, where the R^i are *fresh* intensional relations.

- Add r' to P'.

- For each intensional atom $R^i(\mathbf{x})$ of r', letting R be the relation from which R^i was created, let r_R be the rule of P that has R in its head (by our initial considerations, there is one such rule, and as the program is conjunctive there is exactly one such rule). Define r'_{R^i} from r_R by replacing its head relation from R to R^i, and renaming its head and body variables such that the head is exactly $R^i(\mathbf{x})$. Further rename all variables that occur in the body but not in the head, to replace them by fresh new variables. Add r'_{R^i} to the queue.

We first argue that this process terminates. Indeed, considering the graph G, whenever we pop from the queue a rule with head relation R (or a fresh relation created from a relation R), we add to the queue a finite number of rules for head relations created from relations R' such that the edge (R, R') is in the graph G. The fact that G is acyclic ensures that the process terminates (but note that its running time may generally be exponential in the input). Second, we observe that, by construction, P satisfies the first property, because each occurrence of an intensional relation in a body of P' is fresh, and satisfies the second property, because each variable which is in the body of a rule but not in its head is fresh, so it cannot occur in a previous rule

Last, we verify that P and P' are equivalent, but this is immediate, because any derivation tree for P can be rewritten to a derivation tree for P' (by renaming relations and variables), and vice-versa.

We define Q to be the conjunction of all extensional atoms occurring in P'. To show that it is equivalent to P', the fact that Q implies P' is immediate as the leaves are sufficient to construct a derivation tree, and the fact that P' implies Q is because, letting G' be the call graph of P', by the first property of P' we can easily observe that it is a tree, so the structure of derivation trees of G' also corresponds to P, and by the second property of P' we know that two variables are equal in two extensional

atoms iff they have to be equal in any derivation tree. Hence, P' and Q are indeed equivalent.

We now justify that Q has simplicial width at most k. We do so by building from P' a simplicial decomposition T of Q of width $\leqslant k$. The structure of T is the same as G' (which is actually a tree). For each bag b of T corresponding to a node of G' standing for a rule r of P', we set the domain of b to be the variables occurring in r. It is clear that T is a tree decomposition of Q, because each atom of Q is covered by a bag of T (namely, the one for the rule whose body contained that atom) and the occurrences of each variable form a connected subtree (whose root is the node of G' standing for the rule where it was introduced, using the second condition of P'). Further, T is a simplicial decomposition because P' is clique-frontier-guarded; further, from the second condition, the variables shared between one bag and its child are precisely the head variables of the child rule. The width is $\leqslant k$ because the body size of a CFG-Datalog program is an upper bound on the maximal number of variables in a rule body. □

However, our CFG-Datalog fragment is still exponentially more *concise* than such CQs:

Proposition 3.4.8. *There is a signature σ and a family $(P_n)_{n\in\mathbb{N}}$ of CFG-Datalog programs with body size at most 6 which are positive, conjunctive, and nonrecursive, such that $|P_n| = O(n)$ and any conjunctive query Q_n equivalent to P_n has size $\Omega(2^n)$.*

To prove Proposition 3.4.8, we recall the following classical notion:

Definition 3.4.9. A *match* of a conjunctive query Q in an instance I is a subinstance M of I which is an image of a homomorphism from the canonical instance of Q to I, i.e., M witnesses that $I \models Q$, in particular $M \models Q$. ◁

Our proof will rely on the following elementary observation:

Lemma 3.4.10. *If a CQ Q has a match M in an instance I, then necessarily $|Q| \geqslant |M|$.*

Proof. As M is the image of Q by a homomorphism, it cannot have more facts than Q has atoms. □

We are now ready to prove Proposition 3.4.8:

Proof of Proposition 3.4.8. Fix σ to contain a binary relation R and a binary relation G. Consider the rule $\rho_0 : R_0(x, y) \leftarrow R(x, y)$ and define the following rules, for all $i > 0$:

$$\rho_i : R_i(x, y) \leftarrow G(x, y), R_{i-1}(x, z), R_{i-1}(z, y)$$

For each $i > 0$, we let P_i consist of the rules ρ_j for $0 \leqslant j \leqslant i$, as well as the rule $\mathrm{Goal}() \leftarrow R_i(x, y)$. It is clear that each P_i is positive, conjunctive, and nonrecursive; further, the predicate G ensures that it is a CFG-Datalog program. The arity is 2 and the maximum number of atoms is the body is 3, so the body size is indeed 6.

We first prove by an immediate induction that, for each $i \geqslant 0$, considering the rules of P_i and the intensional predicate R_i, whenever an instance I satisfies $R_i(a, b)$ for two elements $a, b \in \mathrm{dom}(I)$ then there is an R-path of length 2^i from a to b. Now, fixing $i \geqslant 0$, this clearly implies there is an instance I_i of size (number of facts) $\geqslant 2^i$,

namely, an R-path of this length with the right set of additional G-facts, such that $I_i \models P_i$ but any strict subset of I_i does not satisfy P_i.

Now, let us consider a CQ Q_i which is equivalent to P_i, and let us show the desired size bound. By equivalence, we know that $I_i \models Q_i$, hence Q_i has a match M_i in I_i, but any strict subset of I_i does not satisfy Q_i, which implies that, necessarily, $M_i = I_i$ (indeed, otherwise M_i would survive as a match in some strict subset of I_i). Now, by Lemma 3.4.10, we deduce that $|Q_i| \geqslant |M_i|$, and as $|M_i| = |I_i| \geqslant 2^i$, we obtain the desired size bound, which concludes the proof. □

Guarded negation fragments. Having explained the connections between CFG-Datalog and CQs, we now study its connections to the more expressive languages of guarded logics, specifically, the *guarded negation fragment* (GNF), a fragment of first-order logic [Bárány, Cate, and Segoufin 2015]. Indeed, when putting GNF formulas in *GN-normal form* [Bárány, Cate, and Segoufin 2015] or even *weak GN-normal form* [Benedikt, Cate, and Vanden Boom 2014], we can translate them to CFG-Datalog, and we can use the *CQ-rank* parameter [Benedikt, Cate, and Vanden Boom 2014] (that measures the maximal number of atoms in conjunctions) to control the body size parameter. We first recall from [Benedikt, Cate, and Vanden Boom 2014], Appendix B.1, the definitions of a weak GN-normal form formulas and of CQ-rank:

Definition 3.4.11. A weak GN-normal form formulas is a φ-formula in the inductive definition below:

- A disjunction of existentially quantified conjunctions of ψ-formulas is a φ-formula;
- An atom is a ψ-formula;
- The conjunction of a φ-formula and of a guard is a ψ-formula;
- The conjunction of the negation of a φ-formula and of a guard is a ψ-formula.

The *CQ-rank* of a φ-formula is the overall number of conjuncts occurring in the disjunction of existentially quantified conjunctions that defines this subformula. ◁

We can then show:

Proposition 3.4.12. *There is a function f_σ (depending only on σ) such that, for any weak GN-normal form GNF query Q of CQ-rank r, we can compute in time $O(|Q|)$ an equivalent nonrecursive CFG-Datalog program P of body size $f_\sigma(r)$.*

Proof. We define $f_\sigma : n \mapsto \text{arity}(\sigma) \times n$.

We consider an input Boolean GN-normal form formula Q of CQ-rank r, and call T its abstract syntax tree. We rewrite T in linear time to inline in φ-formulas the definition of their ψ-formulas, so all nodes of T consist of φ-formulas, in which all subformulas are guarded (but they can be used positively or negatively).

We now process T bottom-up. We introduce one intensional Datalog predicate R_n per node n in T: its arity is the number of variables that are free at n. We then introduce one rule $\rho_{n,\delta}$ for each disjunct δ of the disjunction that defines n in T: the head of $\rho_{n,\delta}$ is an R_n-atom whose free variables are the variables that are free in n,

and the body of $\rho_{n,\delta}$ is the conjunction that defines δ, with each subformula replaced by the intensional relation that codes it. Of course, we use the predicate R_r for the root r of T as our goal predicate; note that it must be 0-ary, as Q is Boolean so there are no free variables at the root of T. This process defines our CFG-Datalog program P: it is clear that this process runs in linear time.

We first observe that body size for an intensional predicate R_n is less than the CQ-rank of the corresponding subformula. Hence, as the arity of σ is bounded, clearly P has body size $\leqslant f_\sigma(r)$. We next observe that intentional predicates in the bodies of rules of P are always guarded, thanks to the guardedness requirement on Q. Further, it is obvious that P is nonrecursive, as it is computed from the abstract syntax tree T. Last, it is clear that P is equivalent to the original formula Q, as we can obtain Q back simply by inlining the definition of the intensional predicates. □

In fact, the efficient translation of bounded-CQ-rank normal-form GNF programs (using the fact that subformulas are "answer-guarded", like our guardedness requirements) has been used recently (e.g., in [Benedikt, Bourhis, and Vanden Boom 2016]), to give efficient procedures for GNF *satisfiability*. The *satisfiability* problem for a logic formally asks, given a sentence in this logic, whether it is satisfiable (i.e., there is an instance that satisfies it), and two variants of the problem exist: *finite satisfiability*, where we ask for the existence of a *finite* instance (as we defined them in this work), and *unrestricted satisfiability*, where we also allow the satisfying instance to be infinite. The decidability of both finite and unrestricted satisfiability for GNF is shown by translating GNF to automata (for a treewidth which is not fixed, unlike in our context, but depends on the formula). CFG-Datalog further allows clique guards (similar to CGNFO [Bárány, Cate, and Segoufin 2015]), can reuse subformulas (similar to the idea of DAG-representations in [Benedikt, Cate, and Vanden Boom 2014]), and supports recursion (similar to GNFP [Bárány, Cate, and Segoufin 2015], or GN-Datalog [Bárány, Cate, and Otto 2012] but whose combined complexity is intractable — P^{NP}-complete). CFG-Datalog also resembles μCGF [Berwanger and Grädel 2001], but recall that μCGF is not a *guarded negation* logic, so, e.g., μCGF cannot express all CQs, unlike CFG-Datalog or GNF.

Hence, the design of CFG-Datalog, and its translation to automata, has similarities with guarded logics. However, to our knowledge, the idea of applying it to query evaluation is new, and CFG-Datalog is designed to support all relevant features to capture interesting query languages (e.g., clique guards are necessary to capture bounded-simplicial-width queries). Moreover CFG-Datalog is intrinsically more expressive than guarded negation logics as its satisfiability is undecidable, in contrast with GNF [Bárány, Cate, and Segoufin 2015], CGNFO [Bárány, Cate, and Segoufin 2015], GNFP [Bárány, Cate, and Segoufin 2015], GN-Datalog [Bárány, Cate, and Otto 2012], μCGF [Grädel 2002], the satisfiability of all of which is decidable.

Proposition 3.4.13. *Given a signature σ and a CFG-Datalog P over σ, determining if P is satisfiable is undecidable, in both the finite and unrestricted cases.*

Proof. We reduce from the implication problem for functional dependencies and inclusion dependencies, a problem known to be undecidable [Mitchell 1983; Ashok and Vardi 1985] over both finite and unrestricted instances. See also [Abiteboul, Hull, and Vianu 1995] for a general presentation of the problem and formal definitions and notation for functional dependencies and inclusion dependencies.

Let σ be a relational signature, let d be a functional dependency or an inclusion dependency over σ, and let Δ be a set of functional dependencies and inclusion dependencies over σ. The problem is to determine if Δ implies d.

We construct a CFG-Datalog program P over σ which is satisfiable over finite (resp., unrestricted) instances iff Δ implies d over finite (resp., unrestricted) instances, which establishes that CFG-Datalog satisfiability is undecidable.

The intensional signature of the program P is made of:

- a binary relation Eq;
- a nullary relation $P_{\neg\delta}$ for every dependency $\delta \in \Delta \cup \{d\}$;
- a relation $P_{\Pi_Z(S)}$ whose arity is $|Z|$ whenever there is at least one inclusion dependency $R[Y] \subseteq S[Z] \in \Delta \cup \{d\}$;
- the nullary relation Goal.

For every extensional relation R and for every $1 \leqslant i \leqslant \text{arity}(R)$, we add rules of the form:

$$\text{Eq}(x_i, x_i) \leftarrow R(\mathbf{x}).$$

Consequently, for every instance I over σ, the Eq-facts of $P(I)$ will be exactly $\{\text{Eq}(v, v) \mid v \in \text{dom}(I)\}$.

For every functional dependency δ in $\Delta \cup \{d\}$ with $\delta = R[Y] \to R[Z]$, we add the following rules, for $1 \leqslant j \leqslant |Z|$:

$$P_{\neg\delta}() \leftarrow R(\mathbf{x}), R(\mathbf{x}'), \text{Eq}(y_1, y'_1), \ldots, \text{Eq}(y_{|Y|}, y'_{|Y|}), \neg\text{Eq}(z_j, z'_j)$$

where for each $1 \leqslant i \leqslant |Y|$, the variables y_i and y'_i are those at the Y_i-th position in $R(\mathbf{x})$ and $R(\mathbf{x}')$, respectively; and where the variables z_j and z'_j are those at the Z_j-th position in $R(\mathbf{x})$ and $R(\mathbf{x}')$, respectively.

For every inclusion dependency $\delta \in \Delta \cup \{d\}$, with $\delta = R[Y] \subseteq S[Z]$ we add two rules:

$$P_{\Pi_Z(S)}(\mathbf{z}) \leftarrow S(\mathbf{x}) \qquad P_{\neg\delta}() \leftarrow R(\mathbf{x}), \neg P_{\Pi_Z(S)}(\mathbf{y})$$

where $\mathbf{z}$ are the variables at positions Z within $S(\mathbf{x})$ and $\mathbf{y}$ are the variables at positions Y within $R(\mathbf{x})$.

Finally, we add one rule for the goal predicate:

$$\text{Goal}() \leftarrow P_{\neg d}(), \neg P_{\neg\delta_1}(), \cdots, \neg P_{\neg\delta_k}()$$

where $\Delta = \{\delta_1, \ldots, \delta_k\}$.

Note that all the rules that we have written are clearly in CFG-Datalog. Now, let I be some instance. It is clear that for each functional dependency δ, $P_{\neg\delta}()$ is in $P(I)$ iff I does not satisfy δ. Similarly, for each inclusion dependency δ, $P_{\neg\delta}()$ is in $P(I)$ iff I does not satisfy δ. Therefore, for each instance I, Goal() is in $P(I)$ iff I satisfies Δ and I does not satisfy d. Thus P is satisfiable over finite instances (resp., unrestricted instances) iff there exists a finite instance (resp., a finite or infinite instance) that satisfies Δ and does not satisfy d, i.e., iff Δ does imply d over finite instances (resp., over unrestricted instances). □

We point out that the extensional signature is not fixed in this proof, unlike in the rest of the chapter. This is simply to establish the expressiveness of CFG-Datalog, it has no impact on our study of the combined complexity of query evaluation.

Recursive languages. The use of fixpoints in CFG-Datalog, in particular, allows us to capture the combined tractability of interesting recursive languages. First, observe that our guardedness requirement becomes trivial when all intensional predicates are monadic (arity-one), so our main result implies that *monadic Datalog* of bounded body size is tractable in combined complexity on treelike instances. This is reminiscent of the results of [Gottlob, Pichler, and Wei 2010]. We show:

Proposition 3.4.14. *The combined complexity of monadic Datalog query evaluation on bounded-treewidth instances is FPT when parameterized by instance treewidth and body size (as in Definition 3.4.2) of the monadic Datalog program.*

Proof. This is simply by observing that any monadic Datalog program is a CFG-Datalog program with the same body size, so we can simply apply Theorem 3.4.4. □

Second, CFG-Datalog can capture *two-way regular path queries* (2RPQs) [Calvanese, De Giacomo, Lenzeniri, and Vardi 2000; Barceló 2013], a well-known query language in the context of graph databases and knowledge bases. The definition can be found in Section 1.2.

Proposition 3.4.15 ([Mendelzon and Wood 1989; Barceló 2013])**.** *2RPQ query evaluation (on arbitrary instances) has linear time combined complexity.*

CFG-Datalog allows us to capture this result for Boolean 2RPQs on treelike instances. In fact, the above result extends to SAC2RPQs, which are trees of 2RPQs with no multi-edges or loops. We can prove the following result, for Boolean 2RPQs and SAC2RPQs, which further implies translatability to automata (and efficient computation of provenance representations). We do not know whether this extends to the more general classes studied in [Barceló, Romero, and Vardi 2014].

Proposition 3.4.16. *Given a Boolean SAC2RPQ Q (where each 2RPQ is given as a regular expression), we can compute in time $O(|Q|)$ an equivalent CFG-Datalog program P of body size 4.*

Proof. We first show the result for 2RPQs, and then explain how to extend it to SAC2RPQs.

We first use Thompson's construction [Thompson 1968] to compute in linear time an equivalent NFA A (with ε-transitions) on the alphabet $\Sigma^{\pm}$. Note that the result of Thompson's construction has exactly one final state, so we may assume that A has exactly one final state.

We now define the intensional signature of the CFG-Datalog program to consist of one unary predicate P_q for each state q of the automaton, in addition to Goal(). We add the rule $\text{Goal}() \leftarrow P_{q_\text{f}}(x)$ for the final state q_f, and for each extensional relation $R(x, y)$, we add the rules $P_{q_0}(x) \leftarrow R(x, y)$ and $P_{q_0}(y) \leftarrow R(x, y)$, where q_0 is the initial state. We then add rules corresponding to automaton transitions:

- for each transition from q to q' labeled with a letter R, we add the rule $P_{q'}(y) \leftarrow P_q(x), R(x, y)$;
- for each transition from q to q' labeled with a negative letter R^-, we add the rule $P_{q'}(y) \leftarrow P_q(x), R(y, x)$;
- for each ε-transition from q to q' we add the rule $P_{q'}(x) \leftarrow P_q(x)$

This transformation is clearly in linear time, and the result clearly satisfies the desired body size bound. Further, as the result is a monadic Datalog program, it is clearly a CFG-Datalog program. Now, it is clear that, in any instance I where Q holds, from two witnessing elements a and b and a path $\pi : a = c_0, c_1, \ldots, c_n = b$ from a to b satisfying Q, we can build a derivation tree of the Datalog program by deriving $P_{q_0}(a), P_{q_1}(c_1), \ldots, P_{q_n}(c_n)$, where q_0 is the initial state and q_n is final, to match the accepting path in the automaton A that witnesses that π is a match of Q. Conversely, any derivation tree of the Datalog program P that witnesses that an instance satisfies P can clearly be used to extract a path of relations which corresponds to an accepting run in the automaton.

We now extend this argument to SAC2RPQs. Recall from Section 1.2 that a C2RPQ is a conjunction of 2RPQs, i.e., writing a 2RPQ as $Q(x, y)$ with its two free variables, a C2RPQ is a CQ built on 2RPQs. An AC2RPQ is a C2RPQ where the undirected graph on variables defined by co-occurrence between variables is acyclic, and a SAC2RPQ further imposes that there are no self-loops (i.e., atoms of the C2RPQ of the form $Q(x, x)$) and no multiedges (i.e., for each variable pair, there is at most one atom where it occurs).

We will also make a preliminary observation on CFG-Datalog programs: any rule of the form (*) $A(x) \leftarrow A_1(x), \ldots, A_n(x)$, where A and each A_i is a unary atom, can be rewritten in linear time to rules with bounded body size, by creating unary intensional predicates A'_i for $1 \leqslant i \leqslant n$, writing the rule $A'_n(x) \leftarrow A_n(x)$, writing the rule $A'_i(x) \leftarrow A'_{i+1}(x), A_i(x)$ for each $1 \leqslant i < n$, and writing the rule $A(x) \leftarrow A'_1(x)$. Hence, we will write rules of the form (*) in the transformation, with unbounded body size, being understood that we can finish the process by rewriting out each rule of this form to rules of bounded body size.

Given a SAC2RPQ Q, we compute in linear time the undirected graph G on variables, and its connected components. Clearly we can rewrite each connected component separately, by defining one $\mathrm{Goal}_i()$ 0-ary predicate for each connected component i, and adding the rule $\mathrm{Goal}() \leftarrow \mathrm{Goal}_1(), \ldots, \mathrm{Goal}_n()$: this is a rule of form (*), which we can rewrite. Hence, it suffices to consider each connected component separately.

Hence, assuming that the graph G is connected, we root it at an arbitrary vertex to obtain a tree T. For each node n of T (corresponding to a variable of the SAC2RPQ), we define a unary intensional predicate P'_n which will intuitively hold on elements where there is a match of the sub-SAC2RPQ defined by the subtree of T rooted at n, and one unary intensional predicate $P''_{n,n'}$ for all non-root n and children n' of n in T which will hold whenever there is a match of the sub-SAC2RPQ rooted at n which removes all children of n except n'. Of course we add the rule $\mathrm{Goal}() \leftarrow P'_{n_\mathrm{r}}(x)$, where n_r is the root of T.

Now, we rewrite the SAC2RPQ to monadic Datalog by rewriting each edge of T independently, as in the argument for 2RPQs above. Specifically, we assume that the edge when read from bottom to top corresponds to a 2RPQ; otherwise, if the edge is oriented in the wrong direction, we can clearly compute an automaton for the reverse language in linear time from the Thompson automaton, by reversing the direction of transitions in the automaton, and swapping the initial state and the final state. We modify the previous construction by replacing the rule for the initial state P_{q_0} by $P_{q_0}(x) \leftarrow P'_{n'}(x)$ where n' is the lower node of the edge that we are rewriting, and the rule for the goal predicate in the head is replaced by a rule $P''_{n,n'}(x) \leftarrow P_{q_\mathrm{f}}(x)$,

where n is the upper node of the edge, and q_f is the final state of the automaton for the edge: this is the rule that defines the $P''_{n,n'}$.

Now, we define each P'_n as follows:

- If n is a leaf node of T, we define P'_n by the same rules that we used to define P_{q_0} in the previous construction, so that P'_n holds of all elements in the active domain of an input instance.
- If n is an internal node of T, we define $P'_n(x) \leftarrow P''_{n,n_1}(x), \ldots, P''_{n,n_m}(x)$, where $n_1, \ldots, n_m$ are the children of n in T: this is a rule of form (*).

Now, given an instance I satisfying the SAC2RPQ, from a match of the SAC2RPQ as a rooted tree of paths, it is easy to see by bottom-up induction on the tree that we derive P_v with the desired semantics, using the correctness of the rewriting of each edge. Conversely, a derivation tree for the rewriting can be used to obtain a rooted tree of paths with the correct structure where each path satisfies the RPQ corresponding to this edge. □

The rest of the chapter presents the tools needed for our tractability results (alternating two-way automata and cyclic provenance circuits) and their technical proofs.

3.5 Translation to Automata

In this section, we study how we can translate CFG-Datalog queries on treelike instances to tree automata, to be able to evaluate them efficiently. As we showed with Proposition 3.3.1, we need more expressive automata than bNTAs. Hence, we use instead the formalism of *alternating two-way automata* [Comon et al. 2007], i.e., automata that can navigate in trees in any direction, and can express transitions using Boolean formulas on states. Specifically, we introduce for our purposes a variant of these automata, which are *stratified* (i.e., allow a form of stratified negation), and *isotropic* (i.e., no direction is privileged, in particular order is ignored).

As in Section 3.2.2, we will define tree automata that run on Γ-*trees* (remember Section 1.7) for some alphabet Γ. In the rest of this chapter we will always consider that Γ-trees $\langle T, \lambda \rangle$ are rooted and ordered (but not necessarily binary). The *neighborhood* $\mathrm{Nbh}(n)$ of a node $n \in T$ is the set which contains n, all children of n, and the parent of n if it exists.

Stratified isotropic alternating two-way automata. To define the transitions of our alternating automata, we write $\mathcal{B}(X)$ the set of propositional formulas (not necessarily monotone) over a set X of variables: we will assume w.l.o.g. that *negations are only applied to variables*, which we can always enforce using De Morgan's laws. A *literal* is a propositional variable $x \in X$ (*positive* literal) or the negation of a propositional variable $\neg x$ (*negative* literal).

A *satisfying assignment* of $\varphi \in \mathcal{B}(X)$ consists of two *disjoint* sets $P, N \subseteq X$ (for "positive" and "negative") such that φ is a tautology when substituting the variables of P with 1 and those of N with 0, i.e., when we have $\nu(\varphi) = 1$ for every valuation ν of X such that $\nu(x) = 1$ for all $x \in P$ and $\nu(x) = 0$ for all $x \in N$. Note that we

allow satisfying assignments with $P \sqcup N \subsetneq X$, which will be useful for our technical results. We now define our automata:

Definition 3.5.1. A *stratified isotropic alternating two-way automaton* on Γ-trees (Γ-SATWA) is a tuple $A = (\mathcal{Q}, q_{\mathrm{I}}, \Delta, \zeta)$ with $\mathcal{Q}$ a finite set of *states*, q_{I} the *initial state*, Δ the *transition function* from $\mathcal{Q} \times \Gamma$ to $\mathcal{B}(\mathcal{Q})$, and ζ a *stratification function*, i.e., a surjective function from $\mathcal{Q}$ to $\{0, \ldots, m\}$ for some $m \in \mathbb{N}$, such that for any $q, q' \in \mathcal{Q}$ and $f \in \Gamma$, if $\Delta(q, f)$ contains q' as a positive literal (resp., negative literal), then $\zeta(q') \leqslant \zeta(q)$ (resp., $\zeta(q') < \zeta(q)$).

We define by induction on $0 \leqslant i \leqslant m$ an *i-run* of A on a Γ-tree $\langle T, \lambda \rangle$ as a finite tree $\langle T_{\mathrm{r}}, \lambda_{\mathrm{r}} \rangle$, with labels of the form (q, w) or $\neg(q, w)$ for $w \in T$ and $q \in \mathcal{Q}$ with $\zeta(q) \leqslant i$, by the following (nested) inductive definition on T_r:

1. For $q \in \mathcal{Q}$ such that $\zeta(q) < i$, the singleton tree $\langle T_{\mathrm{r}}, \lambda_{\mathrm{r}} \rangle$ with one node labeled by (q, w) (resp., by $\neg(q, w)$) is an i-run if there is a $\zeta(q)$-run of A on $\langle T, \lambda \rangle$ whose root is labeled by (q, w) (resp., if there is no such run);

2. For $q \in \mathcal{Q}$ such that $\zeta(q) = i$, if $\Delta(q, \lambda(w))$ has a satisfying assignment (P, N), if we have an i-run T_{q^-} for each $q^- \in N$ with root labeled by $\neg(q^-, w)$, and an i-run T_{q^+} for each $q^+ \in P$ with root labeled by (q^+, w_{q^+}) for some w_{q^+} in $\mathrm{Nbh}(w)$, then the tree $\langle T_{\mathrm{r}}, \lambda_{\mathrm{r}} \rangle$ whose root is labeled (q, w) and has as children all the T_{q^-} and T_{q^+} is an i-run.

A *run* of A starting in a state $q \in \mathcal{Q}$ at a node $w \in T$ is an m-run whose root is labeled (q, w). We say that A *accepts* $\langle T, \lambda \rangle$ (written $\langle T, \lambda \rangle \models A$) if there exists a run of A on $\langle T, \lambda \rangle$ starting in the initial state q_{I} at the root of T. $\lhd$

Observe that the internal nodes of a run starting in some state q are labeled by states q' in the same stratum as q. The leaves of the run may be labeled by states of a strictly lower stratum or negations thereof, or by states of the same stratum whose transition function is tautological, i.e., by some (q', w) such that $\Delta(q', \lambda(w))$ has $\emptyset, \emptyset$ as a satisfying assignment. Intuitively, if we disallow negation in transitions, our automata amount to the alternating two-way automata used by [Cachat 2002], with the simplification that they do not need parity acceptance conditions (because we only work with finite trees), and that they are *isotropic*: the run for each positive child state of an internal node may start indifferently on *any* neighbor of w in the tree (its parent, a child, or w itself), no matter the direction. (Note, however, that the run for negated child states must start on w itself.)

We will soon explain how the translation of CFG-Datalog is performed, but we first note that evaluation of Γ-SATWAs is in linear time.

Proposition 3.5.2. *For any alphabet Γ, given a Γ-tree $\langle T, \lambda \rangle$ and a Γ-SATWA A, we can determine whether $\langle T, \lambda \rangle \models A$ in time $O(|T| \cdot |A|)$.*

In fact, this result follows from the definition of provenance cycluits for SATWAs in the next section, and the claim that these cycluits can be evaluated in linear time. We will prove Proposition 3.5.2 at the end of Section 3.6.

We now give our main translation result: we can efficiently translate any CFG-Datalog program of bounded body size into a stratified alternating two-way automaton that *tests* it (in the same sense as for bNTAs). For pedagogical purposes, we present the translation for a subclass of CFG-Datalog, namely, *CFG-Datalog with*

guarded negations (CFG$^{\text{GN}}$-Datalog), in which invocations of negative intensional predicates are guarded in rule bodies:

Definition 3.5.3. Let P be a stratified Datalog program. A negative intensional literal $\neg A(\mathbf{x})$ in a rule body ψ of P is *clique-guarded* if, for any two variables $x_i \neq x_j$ of $\mathbf{x}$, it is the case that x_i and x_j co-occur in some positive atom of ψ. A CFG$^{\text{GN}}$-Datalog program is a CFG-Datalog program such that for any rule $R(\mathbf{x}) \leftarrow \psi(\mathbf{x}, \mathbf{y})$, every negative intensional literal in ψ is clique-guarded in ψ. ◁

We will then prove in Section 3.7 the following translation result, and explain at the end of Section 3.7 how it can be extended to full CFG-Datalog:

Theorem 3.5.4. *Given a CFG^{GN}-Datalog program P of body size k_{P} and $k_{\text{I}} \in \mathbb{N}$, we can build in FPT-linear time in $|P|$ (parameterized by $k_{\text{P}}, k_{\text{I}}$) a SATWA A_P testing P on instances of treewidth $\leqslant k_{\text{I}}$.*

Proof sketch. The idea is to have, for every relational symbol R, states of the form $q^{\nu}_{R(\mathbf{x})}$, where ν is a partial valuation of $\mathbf{x}$. This will be the starting state of a run if it is possible to navigate the tree encoding from some starting node and build in this way a total valuation ν' that extends ν and such that $R(\nu'(\mathbf{x}))$ holds. When R is intensional, once ν' is total on $\mathbf{x}$, we go into a state of the form $q_r^{\nu', \mathcal{A}}$ where r is a rule with head relation R and $\mathcal{A}$ is the set of atoms in the body of r (whose size is bounded by the body size). This means that we choose a rule to prove $R(\nu'(\mathbf{x}))$. The automaton can then navigate the tree encoding, build ν' and coherently partition $\mathcal{A}$ so as to inductively prove the atoms of the body. The clique-guardedness condition ensures that, when there is a match of $R(\mathbf{x})$, the elements to which $\mathbf{x}$ is mapped can be found together in a bag. The fact that the automaton is isotropic relieves us from the syntactic burden of dealing with directions in the tree, as one usually has to do with alternating two-way automata. □

3.6 Provenance Cycluits

In the previous section, we have seen how CFG-Datalog programs could be translated efficiently to tree automata that test them on treelike instances. To show that SATWAs can be evaluated in linear time (stated earlier as Proposition 3.5.2), we will introduce an operational semantics for SATWAs based on the notion of *cyclic circuits*, or *cycluits* for short.

We will also use these cycluits as a new powerful tool to compute (Boolean) provenance information. We point to Section 1.7 for the definition of the Boolean provenance of a Boolean query on a relational database. As we already mentioned, we can represent Boolean provenance as Boolean formulas [Imielinski and Lipski 1984; Green, Karvounarakis, and Tannen 2007], or (more recently) as Boolean circuits [Deutch, Milo, Roy, and Tannen 2014; Amarilli, Bourhis, and Senellart 2015]. In this section, we first introduce *monotone cycluits* (monotone Boolean circuits with cycles), for which we define a semantics (in terms of the Boolean function that they express); we also show that cycluits can be evaluated in linear time, given a valuation. Second, we extend them to *stratified cycluits*, allowing a form of stratified negation. We conclude the section by showing how to construct the *provenance* of a SATWA as a cycluit, in FPT-bilinear time. Together with Theorem 3.5.4, this claim implies our main provenance result:

Theorem 3.6.1. *Given a CFG-Datalog program P of body size k_{P} and a relational instance I of treewidth k_{I}, we can construct in FPT-bilinear time in $|I| \cdot |P|$ (parameterized by k_{P} and k_{I}) a representation of the provenance of P on I as a stratified cycluit.*

Of course, this result implies the analogous claims for query languages that are captured by CFG-Datalog parameterized by the body size, as we studied in Section 3.4. When combined with the fact that cycluits can be tractably evaluated, it yields our main result, Theorem 3.4.4. The rest of this section formally introduces cycluits and proves Theorem 3.6.1.

Cycluits. We coin the term *cycluits* for Boolean circuits without the acyclicity requirement. This is the same kind of objects studied in [Riedel and Bruck 2012]. To avoid the problem of feedback loops, however, we first study *monotone cycluits*, and then cycluits with stratified negation.

Definition 3.6.2. A *monotone Boolean cycluit* $C = (G, W, g_0, \mu)$ is defined just like a Boolean circuit (see Section 1.7), except that is does not contain NOT gates and the graph (G, W) is not required to be acyclic. ◁

We now define the semantics of monotone cycluits. A (Boolean) *valuation* of C is a function $\nu : C_{\mathsf{inp}} \to \{0, 1\}$ indicating the value of the input gates. As for standard monotone circuits, a valuation yields an *evaluation* $\nu' : C \to \{0, 1\}$, that we will define shortly, indicating the value of each gate under the valuation ν: we abuse notation and write $\nu(C) \in \{0, 1\}$ for the *evaluation result*, i.e., $\nu'(g_0)$ where g_0 is the output gate of C. The Boolean function *captured* by a cycluit C is thus the Boolean function φ on C_{inp} defined by $\nu(\varphi) := \nu(C)$ for each valuation ν of C_{inp}. We define the evaluation ν' from ν by a least fixed-point computation: we set all input gates to their value by ν, and other gates to 0. We then iterate until the evaluation no longer changes, by evaluating OR-gates to 1 whenever some input evaluates to 1, and AND-gates to 1 whenever all their inputs evaluate to 1. Formally, the semantics of monotone cycluits is defined by Algorithm 1.

Algorithm 1: Semantics of monotone cycluits

Input: Monotone cycluit $C = (G, W, g_0, \mu)$, valuation $\nu : C_{\mathsf{inp}} \to \{0, 1\}$
Output: $\{g \in C \mid \nu'(g) = 1\}$

$S_0 := \{g \in C_{\mathsf{inp}} \mid \nu(g) = 1\}$
$i := 0$
do
  $i{+}{+}$
  $S_i := S_{i-1} \cup \left\{g \in C \mid (\mu(g) = \vee), \exists g' \in S_{i-1}, g' \to g \in W\right\} \cup$
  $\left\{g \in C \mid (\mu(g) = \wedge), \{g' \mid g' \to g \in W\} \subseteq S_{i-1}\right\}$
While $S_i \neq S_{i-1}$
return S_i

The Knaster–Tarski theorem [Tarski 1955] gives an equivalent characterization:

Proposition 3.6.3. *For any monotone cycluit C and Boolean valuation ν of C, letting ν' be the evaluation (as defined by Algorithm 1), the set $S := \{g \in C \mid \nu'(g) = 1\}$ is* the *minimal set of gates (under inclusion) such that:*

(i) S contains the true input gates, i.e., it contains $\{g \in C_{\mathsf{inp}} \mid \nu(g) = 1\}$;

(ii) for any g such that $\mu(g) = \vee$, if some input gate of g is in S, then g is in S;

(iii) for any g such that $\mu(g) = \wedge$, if all input gates of g are in S, then g is in S.

Proof. The operator used in Algorithm 1 is clearly monotone, so by the Knaster–Tarski theorem, the outcome of the computation is the intersection of all sets of gates satisfying the conditions in Proposition 3.6.3. □

Algorithm 1 is a naive fixpoint algorithm running in quadratic time, but we show that the same output can be computed in linear time with Algorithm 2.

Algorithm 2: Linear-time evaluation of monotone cycluits

Input: Monotone cycluit $C = (G, W, g_0, \mu)$, valuation $\nu : C_{\mathsf{inp}} \to \{0, 1\}$
Output: $\{g \in C \mid \nu'(g) = 1\}$

```
1  /* Precompute the in-degree of ∧ gates                          */
2  for g ∈ C s.t. μ(g) = ∧ do
3      M[g] := |{g' ∈ C | g' → g}|
4  Q := {g ∈ C_inp | ν(g) = 1} ∪ {g ∈ C | (μ(C) = ∧) ∧ M[g] = 0}   /* as a
     stack */
5  S := ∅                                                  /* as a bit array */
6  while Q ≠ ∅ do
7      pop g from Q
8      if g ∉ S then
9          add g to S
10         for g' ∈ C | g → g' do
11             if μ(g') = ∨ then
12                 push g' into Q
13             if μ(g') = ∧ then
14                 M[g'] := M[g'] − 1
15                 if M[g'] = 0 then
16                     push g' into Q
17 return S
```

Proposition 3.6.4. *Given any monotone cycluit C and Boolean valuation ν of C, we can compute the evaluation ν' of C in linear time.*

Proof. We use Algorithm 2. We first prove the claim about the running time. The preprocessing to compute M is in linear-time in C (we enumerate at most once every wire), and the rest of the algorithm is clearly in linear time as it is a variant of a DFS traversal of the graph, with the added refinement that we only visit nodes that

evaluate to 1 (i.e., OR-gates with some input that evaluates to 1, and AND-gates where all inputs evaluate to 1).

We now prove correctness. We use the characterization of Proposition 3.6.3. We first check that S satisfies the properties:

(i) S contains the true input gates by construction.

(ii) Whenever an OR-gate g' has an input gate g in S, then, when we added g to S, we have necessarily followed the wire $g \to g'$ and added g' to Q, and later added it to S.

(iii) Whenever an AND-gate g' has all its input gates g in S, there are two cases. The first case is when g has no input gates at all, in which case S contains it by construction. The second case is when g' has input gates: in this case, observe that $M[g']$ was initially equal to the fan-in of g', and that we decrement it for each input gate g of g' that we add to S. Hence, considering the last input gate g of g' that we add to S, it must be the case that $M[g']$ reaches zero when we decrement it, and then we add g' to Q, and later to S.

Second, we check that S is minimal. Assume by contradiction that it is not the case, and consider the first gate g which is added to S while not being in the minimal Boolean valuation S'. It cannot be the case that g was added when initializing S, as we initialize S to contain true input gates and AND-gates with no inputs, which must be true also in S' by the characterization of Proposition 3.6.3. Hence, we added g to S in a later step of the algorithm. However, we notice that we must have added g to S because of the value of its input gates. By minimality of g, these input gates have the same value in S and in S'. This yields a contradiction, because the gates that we add to S are added following the characterization of Proposition 3.6.3. This concludes the proof. □

Another way to evaluate cycluits in linear time is by a rewriting of the circuit to a Horn formula, whose minimal model can be computed in linear time [Dowling and Gallier 1984] and corresponds to the cycluit evaluation.

Stratified cycluits. We now move from monotone cycluits to general cycluits featuring negation. However, allowing arbitrary negation would make it difficult to define a proper semantics, because of possible cycles of negations. Hence, we focus on *stratified cycluits*:

Definition 3.6.5. A *Boolean cycluit* C is defined like a *monotone cycluit*, but further allows NOT-gates ($\mu(g) = \neg$), which are required to have a single input. It is *stratified* if there exists a surjective *stratification function* ζ mapping its gates to $\{0, \ldots, m\}$ for some $m \in \mathbb{N}$ such that $\zeta(g) = 0$ iff $g \in C_{\mathsf{inp}}$, and $\zeta(g) \leqslant \zeta(g')$ for each wire $g \to g'$, the inequality being strict if $\mu(g') = \neg$. ◁

This notion of stratification is similar to that of stratification of Datalog programs (see Section 1.2) or that of stratification of Horn formulas [Dantsin, Eiter, Gottlob, and Voronkov 2001].

Equivalently, we can show that C is stratified if and only if it contains no cycle of gates involving a $\neg$-gate. Moreover if C is stratified we can compute a stratification function in linear time, from a topological sort of its strongly connected components:

Definition 3.6.6. A *strongly connected component* (SCC) of a directed graph $G = (V, E)$ is a subset $S \subseteq V$ that is maximal by inclusion and which ensures that for any $x, y \in S$ with $x \neq y$, there is a directed path from x to y in G. Observe that the SCCs of G are disjoint. A *topological sort* of the SCCs of (G, W) is a linear ordering $(S_1, \ldots, S_k)$ of all the SCCs of G such that for any $1 \leqslant i < j \leqslant k$ and $x \in S_i$ and $y \in S_j$, there is no directed path from y to x in G. ◁

Such a topological sort always exists and can be computed in linear time from G [Tarjan 1972]. We can then show:

Proposition 3.6.7. *Any Boolean cycluit C is stratified iff it it contains no cycle of gates involving a $\neg$-gate. Moreover, a stratification function can be computed in linear time from C.*

Proof. To see why a stratified Boolean cycluit C cannot contain a cycle of gates involving a $\neg$-gate, assume by contradiction that it has such a cycle $g_1 \to g_2 \to \cdots \to g_n \to g_1$. As C is stratified, there exists a stratification function ζ. From the properties of a stratification function, we know that $\zeta(g_1) \leqslant \zeta(g_2) \leqslant \cdots \leqslant \zeta(g_1)$, so that we must have $\zeta(g_1) = \cdots = \zeta(g_n)$. However, letting g_i be such that $\mu(g_i) = \neg$, we know that $\zeta(g_{i-1}) < \zeta(g_i)$ (or, if $i = 1$, $\zeta(g_n) < \zeta(g_1)$), so we have a contradiction.

We now prove the converse direction of the claim, i.e., that any Boolean cycluit which does not contain a cycle of gates involving a $\neg$-gate must have a stratification function, and show how to compute such a function in linear time. Compute in linear time the strongly connected components (SCCs) of C, and a topological sort of the SCCs. As the input gates of C do not themselves have inputs, each of them must have their own SCC, and each such SCC must be a leaf, so we can modify the topological sort by merging these SCCs corresponding to input gates, and putting them first in the topological sort. We define the function ζ to map each gate of C to the index number of its SCC in the topological sort, which ensures in particular that the input gates of C are exactly the gates assigned to 0 by ζ. This can be performed in linear time. Let us show that the result ζ is a stratification function:

- For any edge $g \to g'$, we have $\zeta(g) \leqslant \zeta(g')$. Indeed, either g and g' are in the same strongly connected component and we have $\zeta(g) = \zeta(g')$, or they are not and in this case the edge $g \to g'$ witnesses that the SCC of g precedes that of g', whence, by definition of a topological sort, it follows that $\zeta(g) < \zeta(g')$.
- For any edge $g \to g'$ where $\mu(g') = \neg$, we have $\zeta(g) < \zeta(g')$. Indeed, by adapting the reasoning of the previous bullet point, it suffices to show that g and g' cannot be in the same SCC. Indeed, assuming by contradiction that they are, by definition of a SCC, there must be a path from g' to g, and combining this with the edge $g \to g'$ yields a cycle involving a $\neg$-gate, contradicting our assumption on C. □

We can then use any stratification function to define the evaluation of C (which will be independent of the choice of stratification function):

Definition 3.6.8. Let C be a stratified cycluit with stratification function $\zeta : C \to \{0, \ldots, m\}$, and let ν be a Boolean valuation of C. We inductively define the *i-th stratum evaluation* ν_i, for i in the range of ζ, by setting $\nu_0 := \nu$, and letting ν_i extend the ν_j $(j < i)$ as follows:

1. For g such that $\zeta(g) = i$ with $\mu(g) = \neg$, set $\nu_i(g) := \neg\nu_{\zeta(g')}(g')$ for its one input g'.

2. Evaluate all other g with $\zeta(g) = i$ as for monotone cycluits, considering the $\neg$-gates of point 1. and all gates of stratum $< i$ as input gates fixed to their value in ν_{i-1}.

Letting g_0 be the output gate of C, the Boolean function φ *captured* by C is then defined as $\nu(\varphi) := \nu_m(g_0)$ for each valuation ν of C_{inp}. ◁

Proposition 3.6.9. *We can compute $\nu(C)$ in linear time in the stratified cycluit C and in ν. Moreover, the result is independent of the chosen stratification function.*

Proof. Compute in linear time a stratification function ζ of C using Proposition 3.6.7, and compute the evaluation following Definition 3.6.8. This can be performed in linear time. To see why this evaluation is independent from the choice of stratification, observe that any stratification function must clearly assign the same value to all gates in an SCC. Hence, choosing a stratification function amounts to choosing the stratum that we assign to each SCC. Further, when an SCC S precedes another SCC S', the stratum of S must be no higher than the stratum of S'. So in fact the only freedom that we have is to choose a topological sort of the SCCs, and optionally to assign the same stratum to consecutive SCCs in the topological sort: this amounts to "merging" some SCCs, and is only possible when there are no $\neg$-gates between them. Now, in the evaluation, it is clear that the order in which we evaluate the SCCs makes no difference, nor does it matter if some SCCs are evaluated simultaneously. Hence, the evaluation of a stratified cycluit is well-defined. □

Building provenance cycluits. Having defined cycluits as our provenance representation, we compute the provenance of a query on an instance as the *provenance* of its SATWA on a tree encoding. To do so, we must give a general definition of the provenance of SATWAs. Consider a Γ-tree $\mathcal{T} := \langle T, \lambda\rangle$ for some alphabet Γ, as in Section 3.5. We define a (Boolean) *valuation* ν of $\mathcal{T}$ as a mapping from the nodes of T to $\{0, 1\}$. Writing $\overline{\Gamma} := \Gamma \times \{0, 1\}$, each valuation ν then defines a $\overline{\Gamma}$-tree $\nu(\mathcal{T}) := \langle T, (\lambda \times \nu)\rangle$, obtained by annotating each node of $\mathcal{T}$ by its ν-image. As in [Amarilli, Bourhis, and Senellart 2015], we define the provenance of a $\overline{\Gamma}$-SATWA A on $\mathcal{T}$, which intuitively captures all possible results of evaluating A on possible valuations of $\mathcal{T}$:

Definition 3.6.10. The *provenance* of a $\overline{\Gamma}$-SATWA A on a Γ-tree $\mathcal{T}$ is the Boolean function φ defined on the nodes of T such that, for any valuation ν of $\mathcal{T}$, $\nu(\varphi) = 1$ iff A accepts $\nu(\mathcal{T})$. ◁

We then show that we can efficiently build provenance representations of SATWAs on trees as stratified cycluits:

Theorem 3.6.11. *For any fixed alphabet Γ, given a $\overline{\Gamma}$-SATWA A and a Γ-tree $\mathcal{T} = \langle T, \lambda\rangle$, we can build a stratified cycluit capturing the provenance of A on $\mathcal{T}$ in time $O(|A| \cdot |\mathcal{T}|)$.*

The construction generalizes Proposition 3.1 of [Amarilli, Bourhis, and Senellart 2015] from bNTAs and circuits to SATWAs and cycluits. The reason why we

need cycluits rather than circuits is because two-way automata may loop back on previously visited nodes. To prove Theorem 3.6.11, we construct a cycluit $C^A_{\mathcal{T}}$ as follows. For each node w of T, we create an input node g^{i}_w, a $\neg$-gate $g^{\neg\mathrm{i}}_w$ defined as $\mathrm{NOT}(g^{\mathrm{i}}_w)$, and an OR-gate g^q_w for each state $q \in Q$. Now for each g^q_w, for $b \in \{0,1\}$, we consider the propositional formula $\Delta(q, (\lambda(w), b))$, and we express it as a circuit that captures this formula: we let $g^{q,b}_w$ be the output gate of that circuit, we replace each variable q' occurring positively by an OR-gate $\bigvee_{w' \in \mathrm{Nbh}(w)} g^{q'}_{w'}$, and we replace each variable q' occurring negatively by the gate $g^{q'}_w$. We then define g^q_w as $\mathrm{OR}(\mathrm{AND}(g^{\mathrm{i}}_w, g^{q,1}_w), \mathrm{AND}(g^{\neg\mathrm{i}}_w, g^{q,0}_w))$. Finally, we let the output gate of C be $g^{q_{\mathrm{I}}}_r$, where r is the root of T, and q_{I} is the initial state of A.

It is clear that this process runs in linear time in $|A| \cdot |\mathcal{T}|$. The proof of Theorem 3.6.11 then results from the following claim:

Lemma 3.6.12. *The cycluit $C^A_{\mathcal{T}}$ is a stratified cycluit capturing the provenance of A on $\mathcal{T}$.*

Proof. We first show that $C := C^A_{\mathcal{T}}$ is a stratified cycluit. Let ζ be the stratification function of the $\overline{\Gamma}$-SATWA A and let $\{0, \ldots, m\}$ be its range. We use ζ to define ζ' as the following function from the gates of C to $\{0, \ldots, m+1\}$:

- For any input gate g^{i}_w, we set $\zeta'(g^{\mathrm{i}}_w) := 0$ and $\zeta'(g^{\neg\mathrm{i}}_w) := 1$.
- For an OR gate $g := \bigvee_{w' \in \mathrm{Nbh}(w)} g^{q'}_{w'}$, we set $\zeta'(g) := \zeta(q') + 1$.
- For any state g^q_w, we set $\zeta'(g^q_w) := \zeta(q) + 1$, and do the same for the intermediate AND-gates used in its definition, as well as the gates in the two circuits that capture the transitions $\Delta(q, (\lambda(w), b))$ for $b \in \{0,1\}$, except for the input gates of that circuit (i.e., gates of the form $\bigvee_{w' \in \mathrm{Nbh}(w)} g^{q'}_{w'}$, which are covered by the previous point, or $g^{q'}_w$, which are covered by another application of that point).

Let us show that ζ' is indeed a stratification function for C. We first observe that it is the case that the gates in stratum zero are precisely the input gates. We then check the condition for the various possible wires:

- $g^{\mathrm{i}}_w \to g^{\neg\mathrm{i}}_w$: by construction, we have $\zeta(g^{\mathrm{i}}_w) < \zeta'(g^{\neg\mathrm{i}}_w)$.
- $g \to g'$ where g' is a gate of the form g^q_w and g is an intermediate AND-gate in the definition of a gate of the form g^q_w: by construction we have $\zeta'(g) = \zeta'(g')$, so in particular $\zeta'(g) \leqslant \zeta'(g')$.
- $g \to g'$ where g' is an intermediate AND-gate in the definition of a gate of the form g^q_w, and g is g^{i}_w or $g^{\neg\mathrm{i}}_w$: by construction we have $\zeta'(g) \in \{0,1\}$ and $\zeta'(g') \geqslant 1$, so $\zeta'(g) \leqslant \zeta'(g')$.
- $g \to g'$ where g is a gate in a circuit capturing the propositional formula of some transition of $\Delta(q, \cdot)$ without being an input gate or a NOT-gate of this circuit, and g' is also such a gate, or is an intermediate AND-gate in the definition of g^q_w: then g' cannot be a NOT-gate (remembering that the propositional formulas of transitions only have negations on literals), and by construction we have $\zeta'(g) = \zeta'(g')$.

- $g \to g'$ where g is of the form $\bigvee_{w' \in \mathrm{Nbh}(w)} g^q_{w'}$, and g' is a gate in a circuit describing $\Delta(q', \cdot)$ or an intermediate gate in the definition of $g^{q'}_w$. Then we have $\zeta'(g) = \zeta(q)$ and $\zeta'(g') = \zeta(q')$, and as q occurs as a positive literal in a transition of q', by definition of ζ being a transition function, we have $\zeta(q) \leqslant \zeta(q')$. Now we have $\zeta'(g) = \zeta(q)$ and $\zeta'(g') = \zeta'(q')$ by definition of ζ', so we deduce that $\zeta'(g) \leqslant \zeta'(g')$.

- $g \to g'$ where g' is of the form $\bigvee_{w' \in \mathrm{Nbh}(w)} g^{q'}_{w'}$, and g is one of the $g^{q'}_{w'}$. Then by definition of ζ' we have $\zeta'(g) = \zeta(q')$ and $\zeta'(g') = \zeta(q')$, so in particular $\zeta'(g) \leqslant \zeta'(g')$.

- $g \to g'$ where g is a NOT-gate in a circuit capturing a propositional formula $\Delta(q', (\lambda(w), b))$, and g is then necessarily a gate of the form g^q_w: then clearly q' was negated in φ so we had $\zeta(q) < \zeta(q')$, and as by construction we have $\zeta'(g) = \zeta(q)$ and $\zeta'(g') = \zeta(q')$, we deduce that $\zeta'(g) < \zeta'(g')$.

We now show that C indeed captures the provenance of A on $\langle T, \lambda \rangle$. Let $\nu : T \to \{0, 1\}$ be a Boolean valuation of the inputs of C, that we extend to an evaluation $\nu' : C \to \{0, 1\}$ of C. We claim the following **equivalence**: for all q and w, there exists a run ρ of A on $\nu(T)$ *starting at* w in state q if and only if $\nu'(g^q_w) = 1$.

We prove this claim by induction on the stratum $i = \zeta(q)$ of q. Up to adding an empty first stratum, we can make sure that the base case is vacuous. For the induction step, we prove each implication separately.

Forward direction. First, suppose that there exists a run $\rho = \langle T_r, \lambda_r \rangle$ starting at w in state q, and let us show that $\nu'(g^q_w) = 1$. We show by induction on the run (from bottom to top) that for each node y of the run labeled by a *positive* state (q', w') we have $\nu'(g^{q'}_{w'}) = 1$, and for every node y of the run labeled by a *negative* state $\neg(q', w')$ we have $\nu'(g^{q'}_{w'}) = 0$. The base case concerns the leaves, where there are three possible subcases:

- We may have $\lambda_r(y) = (q', w')$ with $\zeta(q') = i$, so that $\Delta(q', (\lambda(w'), \nu(w')))$ is tautological. In this case, $g^{q'}_{w'}$ is defined as $\mathrm{OR}(\mathrm{AND}(g^{\mathrm{i}}_{w'}, g^{q',1}_{w'}), \mathrm{AND}(g^{\neg \mathrm{i}}_{w'}, g^{q',0}_{w'}))$. Hence, we know that $\nu(g^{q',\nu(w)}_{w'}) = 1$ because the circuit is also tautological, and depending on whether $\nu(w)$ is 0 or 1 we know that $\nu(g^{\neg \mathrm{i}}_{w'}) = 1$ or $\nu(g^{\mathrm{i}}_{w'}) = 1$, so this proves the claim.

- We may have $\lambda_r(y) = (q', w')$ with $\zeta(q') = j$ for $j < i$. By definition of the run ρ, this implies that there exists a run starting at w' in state q'. But then, by the induction on the strata (using the forward direction of the **equivalence**), we must have $\nu(g^{q'}_{w'}) = 1$.

- We may have $\lambda_r(y) = \neg(q', w')$ with $\zeta(q') = j$ for $j < i$. Then by definition there exists no run starting at w' in state q'. Hence again by induction on the strata (using the backward direction of the **equivalence**), we have that $\nu(g^{q'}_{w'}) = 0$.

For the induction case on the run, where y is an internal node, by definition of a run there is a subset $S = \{q_{P_1}, \cdots, q_{P_n}\}$ of positive literals and a subset

$N = \{\neg q_{N_1}, \cdots, \neg q_{N_m}\}$ of negative literals that satisfy $\varphi_{\nu(w')} := \Delta(q', (\lambda(w'), \nu(w')))$ such that:

- For all $q_{P_k} \in P$, there exists a child y_k of y with $\lambda_r(y_k) = (q_{P_k}, w'_k)$ where $w'_k \in \mathrm{Nbh}(w')$;
- For all $\neg q_{N_k} \in N$ there is a child y'_k of y with $\lambda_r(y'_k) = \neg(q_{N_k}, w')$.

Then, by induction on the run, we know that for all q_{P_k} we have $\nu(g_{w'_k}^{q_{P_k}}) = 1$ and for all $\neg q_{N_k}$ we have $\nu(g_{w'}^{q_{N_k}}) = 0$. Let us show that we have $\nu(g_{w'}^{q'}) = 1$, which would finish the induction case on the run. There are two cases: either $\nu(w') = 1$ or $\nu(w') = 0$. In the first case, remember that the first input of the OR-gate $g_{w'}^{q'}$ is an AND-gate of $g_{w'}^{\mathrm{i}}$ and the output gate $g_{w'}^{q',1}$ of a circuit coding φ_1 on inputs including the $g_{w'_k}^{q_{P_k}}$ and $g_{w'}^{q_{N_k}}$. We have $\nu(g_{w'}^{\mathrm{i}}) = 1$ because $\nu(w') = 1$, and the second gate $(g_{w'}^{q',1})$ evaluates to 1 by construction of the circuit, as witnessed by the Boolean valuation of the $g_{w'_k}^{q_{P_k}}$ and $g_{w'}^{q_{N_k}}$. In the second case we follow the same reasoning but with the second input of $g_{w'}^{q'}$ instead, which is an AND-gate on $g_{w'}^{\neg\mathrm{i}}$ and a circuit coding φ_0.

By induction on the run, the claim is proven, and applying it to the root of the run concludes the proof of the first direction of the **equivalence** (for the induction step of the induction on strata).

Backward direction. We now prove the converse implication for the induction step of the induction on strata, i.e., letting i be the current stratum, for every node w and state q with $\zeta(q) = i$, if $\nu(g_w^q) = 1$ then there exists a run ρ of A starting at w. From the definition of the stratification function ζ' of the cycluit from ζ, we have $\zeta'(g_w^q) = \zeta(q) + 1$, so as $\nu(g_w^q) = 1$ we know that $\nu_{i+1}(g_w^q) = 1$, where ν_{i+1} is the $i+1$-th stratum evaluation of C (remember Definition 3.6.8). By induction hypothesis on the strata, we know from the **equivalence** that, for any $j \leqslant i$, for any gate $g_{w''}^{q''}$ of C with $\zeta(g_{w''}^{q''}) = j$, we have $\nu_j(g_{w''}^{q''}) = 1$ iff there exists a run ρ of A on $\nu(T)$ *starting at* w'' in state q''.

Recall that the definition of ν_{i+1} according to Definition 3.6.8 proceeds in three steps. Initially, we fix the value in ν_{i+1} of gates of lower strata, so we can then conclude by induction hypothesis on the strata. We then set the value of all NOT-gates in ν_{i+1}, but these cannot be of the form $g_{w'}^{q'}$ so there is nothing to show. Last, we evaluate all other gates with Algorithm 1. We then show our claim by an induction on the iteration in the application of Algorithm 1 for ν_{i+1} where the gate g_w^q was set to 1. The base case, where g_w^q was initially true, was covered in the beginning of this paragraph.

For the induction step on the application of Algorithm 1, when a gate $g_{w'}^{q'}$ is set to true by ν_{i+1}, as $g_{w'}^{q'}$ is an OR-gate by construction, from the workings of Algorithm 1, there are two possibilities: either its input AND-gate that includes $g_{w'}^{\mathrm{i}}$ was true, or its input AND-gate that includes $g_{w'}^{\neg\mathrm{i}}$ was true. We prove the first case, the second being analogous. From the fact that $g_{w'}^{\mathrm{i}}$ is true, we know that $\nu(w') = 1$. Consider the other input gate to that AND gate, which is the output gate of a circuit C' reflecting $\varphi := \Delta(q', (\lambda(w'), \nu(w')))$, with the input gates adequately substituted. We consider the value by ν_{i+1} of the gates that are used as input gates of C' in the construction of C (i.e., OR-gates, in the case of variables that occur positively, or directly $g_{w'}^{q''}$-gates, in the case of variables that occur negatively). By construction

of C', the corresponding Boolean valuation ν' is a witness to the satisfaction of φ. By induction hypothesis on the strata (for the negated inputs to C'; and for the non-negated inputs to C' which are in a lower stratum) and on the step at which the gate was set to true by Algorithm 1 (for the inputs in the same stratum, which must be positive), the valuation of these inputs reflects the existence of the corresponding runs. Hence, we can assemble these (i.e., a leaf node in the first two cases, a run in the third case) to obtain a run starting at w' for state q' using the Boolean valuation ν' of the variables of φ; this valuation satisfies φ as we have argued.

This concludes the two inductions of the proof of the **equivalence** for the induction step of the induction on strata, which concludes the proof of Theorem 3.6.11. □

Note that the proof can be easily modified to make it work for standard alternating two-way automata rather than our isotropic automata.

Proving Theorem 3.6.1. We are now ready to conclude the proof of our main provenance construction result, i.e., Theorem 3.6.1. We do so by explaining how our provenance construction for $\overline{\Gamma}$-SATWAs can be used to compute the provenance of a CFG-Datalog query on a treelike instance. This is again similar to [Amarilli, Bourhis, and Senellart 2015].

Recall the definition of tree encodings from Section 1.5, and the definition of the alphabet Γ_σ^k. To represent the dependency of automaton runs on the presence of individual facts, we will be working with $\overline{\Gamma_\sigma^k}$-trees, where the Boolean annotation on a node n indicates whether the fact coded by n (if any) is present or absent. The semantics is that we map back the result to Γ_σ^k as follows:

Definition 3.6.13. We define the mapping ε from $\overline{\Gamma_\sigma^k}$ to Γ_σ^k by:

- $\varepsilon((d, s), 1)$ is just (d, s), indicating that the fact of s (if any) is kept;
- $\varepsilon((d, s), 0)$ is $(d, \emptyset)$, indicating that the fact of s (if any) is removed.

We abuse notation and also see ε as a mapping from $\overline{\Gamma_\sigma^k}$-trees to Γ_σ^k-trees by applying it to each node of the tree. ◁

As our construction of provenance applies to automata on $\overline{\Gamma_\sigma^k}$, we show the following easy *lifting lemma* (generalizing Lemma 3.3.4 of [Amarilli 2016]):

Lemma 3.6.14. *For any Γ_σ^k-SATWA A, we can compute in linear time a $\overline{\Gamma_\sigma^k}$-SATWA A' such that, for any $\overline{\Gamma_\sigma^k}$-tree E, we have that A' accepts E iff A accepts $\varepsilon(E)$.*

Proof. The proof is exactly analogous to that of Lemma 3.3.4 of [Amarilli 2016]. □

We are now ready to conclude the proof of our main provenance result (Theorem 3.6.1):

Proof of Theorem 3.6.1. Given the program P and instance I, use Theorem 3.5.4 to compute in FPT-linear time in $|P|$ a Γ_σ^k-SATWA A that tests P on instances of treewidth $\leqslant k_{\mathrm{I}}$, for k_{I} the treewidth bound. Compute also in FPT-linear time a tree encoding $\langle E, \lambda \rangle$ of the instance I (i.e., a Γ_σ^k-tree), using Theorem 1.5.2. Lift the Γ_σ^k-SATWA A in linear time using Lemma 3.6.14 to a $\overline{\Gamma_\sigma^k}$-SATWA A', and use Theorem 3.6.11 on A' and $\langle E, \lambda \rangle$ to compute in FPT-bilinear time a stratified cycluit

C' that captures the provenance of A' on $\langle E, \lambda \rangle$: the inputs of C' correspond to the nodes of E. Let C be obtained from C' in linear time by changing the inputs of C' as follows: those which correspond to nodes n of $\langle E, \lambda \rangle$ containing a fact (i.e., with label (d, s) for $|s| = 1$) are renamed to be an input gate that stands for the fact of I coded in this node; the nodes n of $\langle E, \lambda \rangle$ containing no fact are replaced by a 0-gate, i.e., an OR-gate with no inputs. Clearly, C is still a stratified Boolean cycluit, and C_{inp} is exactly the set of facts of I.

All that remains to show is that C captures the provenance of P on I in the sense of Section 1.7. To see why this is the case, consider an arbitrary Boolean valuation ν mapping the facts of I to $\{0, 1\}$, and call $\nu(I) := \{F \in I \mid \nu(F) = 1\}$. We must show that $\nu(I)$ satisfies P iff $\nu(C) = 1$. By construction of C, it is obvious that $\nu(C) = 1$ iff $\nu'(C') = 1$, where ν' is the Boolean valuation of C_{inp} defined by $\nu'(n) = \nu(F)$ when n codes some fact F in $\langle E, \lambda \rangle$, and $\nu'(n) = 0$ otherwise. By definition of the provenance of A' on $\langle E, \lambda \rangle$, we have $\nu'(C') = 1$ iff A' accepts $\nu'(\langle E, \lambda \rangle)$, that is, by definition of lifting, iff A accepts $\varepsilon(\nu'(\langle E, \lambda \rangle))$. Now all that remains to observe is that $\varepsilon(\nu'(\langle E, \lambda \rangle))$ is precisely a tree encoding of the instance $\nu(I)$: this is by definition of ν' from ν, and by definition of our tree encoding scheme. Hence, by definition of A testing P, the tree $\varepsilon(\nu'(\langle E, \lambda \rangle))$ is accepted by A iff $\nu(I)$ satisfies P. This finishes the chain of equivalences, and concludes the proof of Theorem 3.6.1. □

As promised in Section 3.5, we still need to show how we can use cycluits to evaluate a SATWA in linear time. We recall the statement of Proposition 3.5.2:

Proposition 3.5.2. *For any alphabet Γ, given a Γ-tree $\langle T, \lambda \rangle$ and a Γ-SATWA A, we can determine whether $\langle T, \lambda \rangle \models A$ in time $O(|T| \cdot |A|)$.*

Proof. We use Theorem 3.6.11 to compute a provenance cycluit C of the SATWA (modified to be a $\overline{\Gamma}$-SATWA by simply ignoring the second component of the alphabet) in time $O(|T| \cdot |A|)$. Then we conclude by evaluating the resulting provenance cycluit (for an arbitrary valuation of that circuit) in time $O(|C|)$ using Proposition 3.6.9.

Note that, intuitively, the fixpoint evaluation of the cycluit can be understood as a least fixpoint computation to determine which pairs of states and tree nodes (of which there are $O(|T| \cdot |A|)$) are reachable. □

This concludes the presentation of our provenance results.

3.7 Proof of Translation

In this section, we prove our main technical theorem, Theorem 3.5.4, which we recall here:

Theorem 3.5.4. *Given a CFG^{GN}-Datalog program P of body size k_{P} and $k_{\mathrm{I}} \in \mathbb{N}$, we can build in FPT-linear time in $|P|$ (parameterized by $k_{\mathrm{P}}, k_{\mathrm{I}}$) a SATWA A_P testing P on instances of treewidth $\leqslant k_{\mathrm{I}}$.*

We then explain at the end of the section how this can be extended to full CFG-Datalog (i.e., with negative intensional predicates not being necessarily guarded in rule bodies).

3.7.1 Guarded-Negation Case

First, we introduce some useful notations to deal with valuations of variables as constants of the encoding alphabet. Recall that $\mathcal{D}_{k_\mathrm{I}}$ is the domain of elements for treewidth k_I, used to define the alphabet $\Gamma_\sigma^{k_\mathrm{I}}$ of tree encodings of width k_I.

Definition 3.7.1. Given a tuple $\mathbf{x}$ of variables, a *partial valuation* of $\mathbf{x}$ is a function ν from $\mathbf{x}$ to $\mathcal{D}_{k_\mathrm{I}} \sqcup \{?\}$. The set of *undefined* variables of ν is $U(\nu) = \{x_j \mid \nu(x_j) = ?\}$: we say that the variables of $U(\nu)$ are *not defined* by ν, and the other variables are *defined* by ν.

A *total valuation* of $\mathbf{x}$ is a partial valuation ν of $\mathbf{x}$ such that $U(\nu) = \emptyset$. We say that a valuation ν' *extends* another valuation ν if the domain of ν' is a superset of that of ν, and if all variables defined by ν are defined by ν' and are mapped to the same value. For $\mathbf{y} \subseteq \mathbf{x}$, we say that ν is *total on* $\mathbf{y}$ if its restriction to $\mathbf{y}$ is a total valuation.

For any two partial valuations ν of $\mathbf{x}$ and ν' of $\mathbf{y}$, if we have $\nu(z) = \nu'(z)$ for all z in $(\mathbf{x} \cap \mathbf{y}) \setminus (U(\nu) \cup U(\nu'))$, then we write $\nu \cup \nu'$ for the valuation on $\mathbf{x} \cup \mathbf{y}$ that maps every z to $\nu(z)$ or $\nu'(z)$ if one is defined, and to "?" otherwise.

When ν is a partial valuation of $\mathbf{x}$ with $\mathbf{x} \subseteq \mathbf{x}'$ and we define a partial valuation ν' of $\mathbf{x}'$ with $\nu' := \nu$, we mean that ν' is defined like ν on $\mathbf{x}$ and is undefined on $\mathbf{x}' \setminus \mathbf{x}$. ◁

Definition 3.7.2. Let $\mathbf{x}$ and $\mathbf{y}$ be two tuples of variables of same arity (note that some variables of $\mathbf{x}$ may be repeated, and likewise for $\mathbf{y}$). Let $\nu : \mathbf{x} \to \mathcal{D}_{k_\mathrm{I}}$ be a total valuation of $\mathbf{x}$. We define $\mathrm{Hom}_{\mathbf{y},\mathbf{x}}(\nu)$ to be the (unique) homomorphism from the tuple $\mathbf{y}$ to the tuple $\nu(\mathbf{x})$ (i.e., the unification between the tuple $\mathbf{y}$ and the tuple $\nu(\mathbf{x})$), if such a homomorphism exists; otherwise, $\mathrm{Hom}_{\mathbf{y},\mathbf{x}}(\nu)$ is null. ◁

The rest of this section proves Theorem 3.5.4 in two steps. First, we build a SATWA A'_P and we prove that A'_P tests P on instances of treewidth $\leqslant k_\mathrm{I}$; however, the construction of A'_P that we present is not FPT-linear. Second, we explain how to modify the construction to construct an equivalent SATWA A_P while respecting the FPT-linear time bound.

Construction of A'_P. We formally construct the SATWA A'_P by describing its states and transitions. First, for every extensional atom $S(\mathbf{x})$ appearing in (the body) of a rule of P and partial valuation ν of $\mathbf{x}$, we introduce a state $q^\nu_{S(\mathbf{x})}$. This will be the starting state of a run if it is possible to navigate the tree encoding from some starting node and build in this way a total valuation ν' that extends ν and such that $S(\nu'(\mathbf{x}))$ holds in the tree encoding, in a node whose domain elements that are in the image of ν' will decode to the same element as they do in the node where the automaton can reach state $q^\nu_{S(\mathbf{x})}$. In doing so, one has to be careful not to leave the occurrence subtree of the values in the image of the valuation, which we call the *allowed subtree*. Indeed, remember that in a tree encoding, an element $a \in \mathcal{D}_{k_\mathrm{I}}$ appearing in two bags that are separated by another bag not containing a is used to encode two distinct elements of the original instance, rather than the same element. We now define the transitions needed to implement this.

Let $(d, s) \in \Gamma_\sigma^{k_\mathrm{I}}$ be a symbol; we have the following transitions:

- If there is an $x_j \in \mathbf{x}$ such that $\nu(x_j) \neq ?$ (i.e., x_j is defined by ν) and $\nu(x_j) \notin d$, then $\Delta(q^\nu_{S(\mathbf{x})}, (d, s)) := \bot$. This is to prevent the automaton from leaving the allowed subtree.

- Else if ν is not total, then $\Delta(q^\nu_{S(\mathbf{x})}, (d, s)) := q^\nu_{S(\mathbf{x})} \vee \bigvee_{a \in d, x_j \in U(\nu)} q^{\nu \cup \{x_j \mapsto a\}}_{S(\mathbf{x})}$. That is, either we continue navigating in the same state (but remember that the automaton may move to any neighbor node), or we guess a value for some undefined variable.

- Else if ν is total but $s \neq S(\nu(\mathbf{x}))$, then $\Delta(q^\nu_{S(\mathbf{x})}, (d, s)) := q^\nu_{S(\mathbf{x})}$: if the fact s of the node is not a match, then we continue searching.

- Else, the only remaining possibility is that ν is total and that $s = S(\nu(\mathbf{x}))$, in which case we set $\Delta(q^\nu_{S(\mathbf{x})}, (d, s)) := \top$, i.e., we have found a node containing the desired fact.

Let r be a rule of P and $\mathcal{A}$ be a subset of the literals in the body of r. We write $\mathrm{vars}(\mathcal{A})$ the set of variables that appear in some atom of $\mathcal{A}$. For every rule r of P, for every subset $\mathcal{A}$ of the literals in the body of r, and for every partial valuation ν of $\mathrm{vars}(\mathcal{A})$ that defines all the variables that are also in the head of r, we introduce a state $q^{\nu,\mathcal{A}}_r$. This state is intended to prove the literals in $\mathcal{A}$ with the partial valuation ν. We will describe the transitions for those states later.

For every intensional predicate $R(\mathbf{x})$ appearing in a rule of P and partial valuation ν of $\mathbf{x}$, we have a state $q^\nu_{R(\mathbf{x})}$. This state is intended to prove $R(\mathbf{x})$ with a total extension of ν. Let $(d, s) \in \Gamma^{k_\mathrm{I}}_\sigma$ be a symbol; we have the following transitions:

- If there is a j such that x_j is defined by ν and $\nu(x_j) \notin d$, then $\Delta(q^\nu_{R(\mathbf{x})}, (d, s)) := \bot$. This is again in order to prevent the automaton from leaving the allowed subtree.

- Else if ν is not total, then $\Delta(q^\nu_{R(\mathbf{x})}, (d, s)) := q^\nu_{R(\mathbf{x})} \vee \bigvee_{a \in d, x_j \in U(\nu)} q^{\nu \cup \{x_j \mapsto a\}}_{R(\mathbf{x})}$. Again, either we continue navigating in the same state, or we guess a value for some undefined variable.

- Else (in this case ν is total), $\Delta(q^\nu_{R(\mathbf{x})}, (d, s))$ is defined as the disjunction of all the $q^{\nu',\mathcal{A}}_r$ for each rule r such that the head of r is $R(\mathbf{y})$, $\nu' := \mathrm{Hom}_{\mathbf{y},\mathbf{x}}(\nu)$ is not null and $\mathcal{A}$ is the set of all literals in the body of r. Notice that because ν is total on $\mathbf{x}$, ν' is also total on $\mathbf{y}$. This transition simply means that we need to chose an appropriate rule to prove $R(\mathbf{x})$. We point out here that these transitions are the ones that make the construction quadratic instead of linear in $|P|$, but this will be handled later.

It is now time to describe transitions for the states $q^{\nu,\mathcal{A}}_r$. Let $(d, s) \in \Gamma^{k_\mathrm{I}}_\sigma$, then:

- If there is a variable z in $\mathcal{A}$ such that z is defined by ν and $\nu(z) \notin d$, then $\Delta(q^{\nu,\mathcal{A}}_r, (d, s)) := \bot$. Again, this is to prevent the automaton from leaving the allowed subtree.

- Else, if $\mathcal{A}$ contains at least two literals, then $\Delta(q_r^{\nu,\mathcal{A}}, (d,s))$ is defined as a disjunction of $q_r^{\nu,\mathcal{A}}$ and of $\Big[$ a disjunction over all the non-empty sets $\mathcal{A}_1, \mathcal{A}_2$ that partition $\mathcal{A}$ of $\Big[$a disjunction over all the total valuations ν' of $U(\nu) \cap \mathrm{vars}(\mathcal{A}_1) \cap \mathrm{vars}(\mathcal{A}_2)$ with values in d of $\Big[q_r^{\nu\cup\nu',\mathcal{A}_1} \wedge q_r^{\nu\cup\nu',\mathcal{A}_2}\Big]\Big]\Big]$. This transition means that we allow to partition in two the literals that need to be proved, and for each class of the partition we launch one run that will have to prove the literals of that class. In doing so, we have to take care that the two runs will build valuations that are consistent. This is why we fix the value of the variables that they have in common with a total valuation ν'.

- Else, if $\mathcal{A} = \{T(\mathbf{y})\}$ where T is an extensional or an intensional relation, then $\Delta(q_r^{\nu,\mathcal{A}}, (d,s)) := q_{T(\mathbf{y})}^{\nu}$.

- Else, if $\mathcal{A} = \{\neg R'(\mathbf{y})\}$ where R' is an intensional relation, and if $|\mathbf{y}| = 1$, and if $\nu(y)$ is undefined (where we write y the one element of $\mathbf{y}$), then $\Delta(q_r^{\nu,\mathcal{A}}, (d,s)) := q_r^{\nu,\mathcal{A}} \vee \bigvee_{a\in d} q_r^{\nu\cup\{y\mapsto a\},\mathcal{A}}$.

- Else, if $\mathcal{A} = \{\neg R'(\mathbf{y})\}$ where R' is an intensional relation, then we will only define the transitions in the case where ν is total on $\mathbf{y}$, in which case we set $\Delta(q_r^{\nu,\mathcal{A}}, (d,s)) := \neg q_{R'(\mathbf{y})}^{\nu}$. It is sufficient to define the transitions in this case, because $q_r^{\nu,\{\neg R'(\mathbf{y})\}}$ can only be reached if ν is total on $\mathbf{y}$. Indeed, if $|\mathbf{y}| = 1$, then ν must be total on $\mathbf{y}$ because we would have applied the previous bullet point otherwise. If $|\mathbf{y}| > 1$, the only way we could have reached the state $q_r^{\nu,\{\neg R'(\mathbf{y})\}}$ is by a sequence of transitions involving $q_r^{\nu_0,\mathcal{A}_0}, \ldots, q_r^{\nu_m,\mathcal{A}_m}$, where $\mathcal{A}_0$ are all the literals in the body of r, $\mathcal{A}_m$ is $\mathcal{A}$ and ν_m is ν. We can then see that, during the partitioning process, $\neg R'(\mathbf{y})$ must have been separated from all the (positive) atoms that formed its guard (recall the definition of CFG$^{\mathrm{GN}}$-Datalog), hence all its variables have been assigned a valuation.

Finally, the initial state of A'_P is $q_{\mathrm{Goal}}^{\emptyset}$.

We describe the stratification function ζ' of A'_P. Let ζ be that of P. Observe that we can assume w.l.o.g. that the first stratum of ζ (i.e., relations R with $\zeta(R) = 1$) contains exactly all the extensional relations. For any state q of the form $q_{T(\mathbf{x})}^{\nu}$ or $q_r^{\nu,\mathcal{A}}$ with r having as head relation T (T begin extensional or intensional), then $\zeta'(q)$ is defined to be $\zeta(T) - 1$. Notice that this definition ensures that only the states corresponding to extensional relations are in the first stratum of ζ'. It is then clear from the transitions that ζ' is a valid stratification function for A'_P.

As previously mentioned, the construction of A'_P is not FPT-linear, but we will explain at the end of the proof how to construct in FPT-linear time a SATWA A_P equivalent to A'_P.

A'_P tests P on instances of treewidth $\leqslant k_{\mathrm{I}}$. To show this claim, let $\langle T, \lambda_E\rangle$ be a (σ, k_{I})-tree encoding. Let I be the instance obtained by decoding $\langle T, \lambda_E\rangle$; we know that I has treewidth $\leqslant k_{\mathrm{I}}$ and that we can define from $\langle T, \lambda_E\rangle$ a tree

decomposition $\langle T, \mathrm{dom}\rangle$ of I whose underlying tree is also T. For each node $n \in T$, let $\mathrm{dec}_n : \mathcal{D}_{k_\mathrm{I}} \to \mathrm{dom}(n)$ be the function that decodes the elements in node n of the encoding to the elements of I that are in the corresponding bag of the tree decomposition, and let $\mathrm{enc}_n : \mathrm{dom}(n) \to \mathcal{D}_{k_\mathrm{I}}$ be the inverse function that encodes back the elements, so that we have $\mathrm{dec}_n \circ \mathrm{enc}_n = \mathrm{enc}_n \circ \mathrm{dec}_n = \mathrm{Id}$. We will denote elements of $\mathcal{D}_{k_\mathrm{I}}$ by a and elements in the domain of I by c.

We recall some properties of tree decompositions and tree encodings:

Property 3.7.3. *Let n_1, n_2 be nodes of T and $a \in \mathcal{D}_{k_\mathrm{I}}$ be an (encoded) element that appears in the λ_E-image of n_1 and n_2. Then the element a appears in the λ_E-image of every node in the path from n_1 to n_2 if and only if* $\mathrm{dec}_{n_1}(a) = \mathrm{dec}_{n_2}(a)$.

Property 3.7.4. *Let n_1, n_2 be nodes of T and c be an element of I that appears in* $\mathrm{dom}(n_1) \cap \mathrm{dom}(n_2)$. *Then for every node n' on the path from n_1 to n_2, c is also in* $\mathrm{dom}(n')$, *and moreover* $\mathrm{enc}_{n'}(c) = \mathrm{enc}_{n_1}(c)$.

We start with the following lemma about extensional facts:

Lemma 3.7.5. *For every extensional relation S, node $n \in T$, variables $\mathbf{y}$, and partial valuation ν of $\mathbf{y}$, there exists a run ρ of A'_P starting at node n in state $q^{\nu}_{S(\mathbf{y})}$ if and only if there exists a fact $S(\mathbf{c})$ in I such that we have $\mathrm{dec}_n(\nu(y_j)) = c_j$ for every y_j defined by ν. We call this a match $\mathbf{c}$ of $S(\mathbf{y})$ in I that is* compatible with ν at node n.

Proof. We prove each direction in turn.

Forward direction. Suppose there exists a run ρ of A'_P starting at node n in state $q^{\nu}_{S(\mathbf{y})}$. First, notice that by design of the transitions starting in a state of that form, states appearing in the labeling of the run can only be of the form $q^{\nu'}_{S(\mathbf{y})}$ for an extension ν' of ν. We will show by induction on the run that for every node π of the run labeled by $(q^{\nu'}_{S(\mathbf{y})}, m)$, there exists $\mathbf{c}'$ such that $S(\mathbf{c}') \in I$ and $\mathbf{c}'$ is compatible with ν' at node m. This will conclude the proof of the forward part of the lemma, by taking $m = n$.

The base case is when π is a leaf of ρ. The node π is then labeled by $(q^{\nu'}_{S(\mathbf{y})}, m)$ such that $\Delta(q^{\nu'}_{S(\mathbf{y})}, \lambda_E(m)) = \top$. Let $(d, s) = \lambda_E(m)$. By construction of the automaton we have that ν' is total and $s = S(\nu'(\mathbf{y}))$. We take $\mathbf{c}'$ to be $\mathrm{dec}_m(\nu'(\mathbf{y}))$, which satisfies the compatibility condition by definition and is such that $S(\mathbf{c}') = S(\mathrm{dec}_m(\nu'(\mathbf{y}))) = \mathrm{dec}_m(s) \in I$.

When π is an internal node of ρ, we write again $(q^{\nu'}_{S(\mathbf{y})}, m)$ its label. By definition of the transitions of the automaton, we have $\Delta(q^{\nu'}_{S(\mathbf{y})}, (d, s)) = q^{\nu'}_{S(\mathbf{y})} \vee \bigvee_{a \in d, y_j \in U(\nu')} q^{\nu' \cup \{y_j \mapsto a\}}_{S(\mathbf{y})}$. Hence, the node π has at least one child π', the second component of the label of π' is some $m' \in \mathrm{Nbh}(m)$, and we have two cases depending on the first component of its label (i.e., the state):

- π' may be labeled by $(q^{\nu'}_{S(\mathbf{y})}, m')$. Then by induction on the run there exists $\mathbf{c}''$ such that $S(\mathbf{c}'') \in I$ and $\mathbf{c}''$ is compatible with ν' at node m'. We take $\mathbf{c}'$ to be $\mathbf{c}''$, so that we only need to check the compatibility condition, i.e., that for every y_j defined by ν', $\mathrm{dec}_m(\nu'(y_j)) = c_j = \mathrm{dec}_{m'}(\nu'(y_j))$. This is true by Property 3.7.3. Indeed, for every y_j defined by ν', we must have $\nu'(y_j) \in m'$, otherwise π' would have a label that cannot occur in a run (because this would mean that we have escaped the allowed subtree).

- π' is labeled by $(q_{S(\mathbf{y})}^{\nu' \cup \{y_j \mapsto a\}}, m')$ for some $a \in d$ and for some $y_j \in U(\nu')$. Then by induction on the run there exists $\mathbf{c}''$ such that $S(\mathbf{c}'') \in I$ and $\mathbf{c}''$ is compatible with $\nu' \cup \{y_j \mapsto a\}$ at node m'. We take $\mathbf{c}'$ to be $\mathbf{c}''$, which again satisfies the compatibility condition thanks to Property 3.7.3.

Backward direction. Now, suppose that there exists $\mathbf{c}$ such that $S(\mathbf{c}) \in I$ and $\mathbf{c}$ is compatible with ν at node n. The fact $S(\mathbf{c})$ is encoded somewhere in $\langle T, \lambda_E \rangle$, so there exists a node m such that, letting (d, s) be $\lambda_E(m)$, we have $\mathrm{dec}_m(s) = S(\mathbf{c})$. Let $n = m_1, m_2, \ldots, m_p = m$ be the nodes on the path from n to m, and (d_i, s_i) be $\lambda_E(m_i)$ for $1 \leqslant i \leqslant p$. By compatibility, for every y_j defined by ν we have $\mathrm{dec}_n(\nu(y_j)) = c_j$. But $\mathrm{dec}_n(\nu(y_j)) \in \mathrm{dom}(n)$ and $c_j \in \mathrm{dom}(m)$ so by Property 3.7.4, for every $1 \leqslant i \leqslant p$ we have $c_j \in \mathrm{dom}(m_i)$ and $\mathrm{enc}_{m_i}(c_j) = \mathrm{enc}_n(c_j) = \mathrm{enc}_n(\mathrm{dec}_n(\nu(y_j))) = \nu(y_j)$, so that $\nu(y_j) \in d_i$. We can then construct a run ρ starting at node n in state $q_{S(\mathbf{y})}^{\nu}$ as follows. The root π_1 is labeled by $(q_{S(\mathbf{y})}^{\nu}, n)$, and for every $2 \leqslant i \leqslant p$, π_i is the unique child of π_{i-1} and is labeled by $(q_{S(\mathbf{y})}^{\nu}, m_i)$. This part is valid because we just proved that for every i, there is no j such that y_j is defined by ν and $\nu(y_j) \notin d_j$. Now from π_m, we continue the run by staying at node m and building up the valuation, until we reach a total valuation ν_{f} such that $\nu_{\mathrm{f}}(\mathbf{y}) = \mathrm{enc}_m(\mathbf{c})$. Then we have $s = S(\nu_{\mathrm{f}}(\mathbf{y}))$ and the transition is $\top$, which completes the definition of the run. □

The preceding lemma concerns the base case of extensional relations. We now prove a similar *equivalence lemma* for all relations (extensional or intensional). This lemma allows us to conclude the correctness proof, by applying it to the Goal() predicate and to the root of the tree-encoding.

Lemma 3.7.6. *For every relation R, node $n \in T$ and partial valuation ν of $\mathbf{x}$, there exists a run ρ of A'_P starting at node n in state $q_{R(\mathbf{x})}^{\nu}$ if and only if there exists $\mathbf{c}$ such that $R(\mathbf{c}) \in P(I)$ and $\mathbf{c}$ is compatible with ν at node n (i.e., we have $\mathrm{dec}_n(\nu(x_j)) = c_j$ for every x_j defined by ν).*

Proof. We will prove this equivalence by induction on the stratum $\zeta(R)$ of the relation R. The base case ($\zeta(R) = 0$, so R is an extensional relation) was shown in Lemma 3.7.5. For the inductive case, where R is an intensional relation, we prove each direction separately.

Forward direction. First, suppose that there exists a run ρ of A'_P starting at node n in state $q_{R(\mathbf{x})}^{\nu}$. We show by induction on the run (from bottom to top) that for every node π of the run the following implications hold:

(i) If π is labeled with $(q_{R'(\mathbf{y})}^{\nu'}, m)$, then there exists $\mathbf{c}$ such that $R'(\mathbf{c}) \in P(I)$ and $\mathbf{c}$ is compatible with ν' at node m.

(ii) If π is labeled with $\neg(q_{R'(\mathbf{y})}^{\nu'}, m)$, then $R'(\mathrm{dec}_m(\nu'(\mathbf{y}))) \notin P(I)$ (remembering that in this case ν' must be total, thanks to the fact that negations are guarded in rule bodies).

(iii) If π is labeled with $(q_r^{\nu', \mathcal{A}}, m)$, then there exists a mapping $\mu : \mathrm{vars}(\mathcal{A}) \to \mathrm{Dom}(I)$ that is compatible with $\nu'_{|\mathrm{vars}(\mathcal{A})}$ at node m and such that:

- For every positive literal $S(\mathbf{z})$ in $\mathcal{A}$, then $S(\mu(\mathbf{z})) \in P(I)$.

- For every negative literal $\neg S(\mathbf{z})$ in $\mathcal{A}$, then $S(\mu(\mathbf{z})) \notin P(I)$.

The base case is when π is a leaf. Notice that in this case, and by construction of A'_P, the node π cannot be labeled by states corresponding to rules of P: indeed, there are no transitions for these states leading to a tautology, and all transitions to such a state are from a state in the same stratum, so π could not be a leaf. Thus, we have three subcases:

- π may be labeled by $(q^{\nu'}_{R'(\mathbf{y})}, m)$, where R' is extensional. We must show (i), but this follows from Lemma 3.7.5.
- π may be labeled by $(q^{\nu'}_{R'(\mathbf{y})}, m)$, where R' is intensional and verifies $\zeta(R') < i$. Again we need to show (i). By definition of the run ρ, this implies that there exists a run of A'_P starting at m in state $q^{\nu'}_{R'(\mathbf{y})}$. But then (i) follows from the induction hypothesis on the strata (using the forward direction of the equivalence lemma).
- π may be labeled by $\neg(q^{\nu'}_{R'(\mathbf{y})}, m)$, where R' is intensional and verifies $\zeta(R') < i$. Observe that by construction of the automaton, ν' is total (because negations are guarded in rule bodies). We need to show (ii). By definition of the run ρ there exists no run of A'_P starting at m in state $q^{\nu'}_{R'(\mathbf{y})}$. Hence by induction on the strata we have (using the backward direction of the equivalence lemma) that $R'(\mathrm{dec}_m(\nu'(\mathbf{y}))) \notin P(I)$, which is what we needed to show.

For the induction case, where π is an internal node, we let (d, s) be $\lambda_E(m)$ in what follows, and we distinguish five subcases:

- π may be labeled by $(q^{\nu'}_{R'(\mathbf{y})}, m)$ with R' intensional. We need to prove (i). We distinguish two subsubcases:
  - ν' is not total. In that case, given the definition of $\Delta(q^{\nu'}_{R'(\mathbf{y})}, (d, s))$ and of the run, there exists a child π' of π labeled by $(q^{\nu''}_{R'(\mathbf{y})}, m')$, where $m' \in \mathrm{Nbh}(m)$ and ν'' is either ν' or is $\nu' \cup \{x_j \mapsto a\}$ for some x_j undefined by ν' and $a \in d$. Hence by induction on the run there exists $\mathbf{c}'$ such that $R'(\mathbf{c}') \in P(I)$ and $\mathbf{c}'$ is compatible with ν'' at node m'. We then take $\mathbf{c}$ to be $\mathbf{c}'$, and one can check that the compatibility condition holds.
  - ν' is total. In that case, given the definition of $\Delta(q^{\nu'}_{R'(\mathbf{y})}, (d, s))$ and of the run, there exists a child π' of π labeled by $(q_r^{\nu'', \mathcal{A}}, m')$, where $m' \in \mathrm{Nbh}(m)$, where r is a rule with head $R'(\mathbf{z})$, where $\nu'' = \mathrm{Hom}_{\mathbf{z},\mathbf{y}}(\nu')$ is a partial valuation which is not null, and where $\mathcal{A}$ is the set of literals of r. Then, by induction on the run, there exists a mapping $\mu : \mathrm{vars}(\mathcal{A}) \to \mathrm{Dom}(I)$ that verifies (iii). Thus by definition of the semantics of P we have that $R'(\mu(\mathbf{z})) \in P(I)$, and we take $\mathbf{c}$ to be $\mu(\mathbf{z})$. What is left to check is that the compatibility condition holds. We need to prove that $\mathrm{dec}_m(\nu'(\mathbf{y})) = \mathbf{c}$, i.e., that $\mathrm{dec}_m(\nu'(\mathbf{y})) = \mu(\mathbf{z})$. We know, by definition of μ, that $\mathrm{dec}_{m'}(\nu''(\mathbf{z})) = \mu(\mathbf{z})$. So our goal is to prove $\mathrm{dec}_m(\nu'(\mathbf{y})) = \mathrm{dec}_{m'}(\nu''(\mathbf{z}))$, i.e., by definition of ν'' we want $\mathrm{dec}_m(\nu'(\mathbf{y})) = \mathrm{dec}_{m'}(\mathrm{Hom}_{\mathbf{z},\mathbf{y}}(\nu')(\mathbf{z}))$. By definition of $\mathrm{Hom}_{\mathbf{z},\mathbf{y}}(\nu')$, we know that $\nu'(\mathbf{y}) = \mathrm{Hom}_{\mathbf{z},\mathbf{y}}(\nu')(\mathbf{z})$, and this implies the desired equality by applying Property 3.7.3 to m and m'.

- π may be labeled by $(q_r^{\nu',\mathcal{A}}, m)$, where $\mathcal{A} = \{\neg R''(\mathbf{y})\}$, where $|\mathbf{y}| = 1$, and where, writing y the one element of $\mathbf{y}$, y is undefined by ν'. We need to prove (iii). By construction we have $\Delta(q_r^{\nu',\mathcal{A}}, (d,s)) = q_r^{\nu',\mathcal{A}} \vee \bigvee_{a \in d} q_r^{\nu' \cup \{y \mapsto a\},\mathcal{A}}$. So by definition of a run there is $m' \in \mathrm{Nbh}(m)$ and a child π' of π such that π' is labeled by $(q_r^{\nu',\mathcal{A}}, m')$ or by $(q_r^{\nu' \cup \{y \mapsto a\},\mathcal{A}}, m')$ for some $a \in d$. In both cases it is easily seen that we can define an appropriate μ from the mapping μ' that we obtain by induction on the run (more details are given in the next bullet point).

- π may be labeled by $(q_r^{\nu',\mathcal{A}}, m)$ with $\mathcal{A} = \{R''(\mathbf{y})\}$. We need to prove (iii). By construction we have $\Delta(q_r^{\nu',\mathcal{A}}, (d,s)) = q_{R''(\mathbf{y})}^{\nu'}$, so that by definition of the run there is $m' \in \mathrm{Nbh}(m)$ and a child π' of π such that π' is labeled by $(q_{R''(\mathbf{y})}^{\nu'}, m')$. Thus by induction on the run there exists $\mathbf{c}$ such that $R''(\mathbf{c}) \in P(I)$ and $\mathbf{c}$ compatible with ν' at node m'. By Property 3.7.3, $\mathbf{c}$ is also compatible with ν' at node m. We define μ by $\mu(\mathbf{y}) := \mathbf{c}$, which effectively defines it because in this case $\mathrm{vars}(\mathcal{A}) = \mathbf{y}$, and this choice satisfies the required properties.

- π may be labeled by $(q_r^{\nu',\mathcal{A}}, m)$, with $\mathcal{A} = \{\neg R''(\mathbf{y})\}$ and ν' total on $\mathbf{y}$. We again need to prove (iii). By construction we have $\Delta(q_r^{\nu',\mathcal{A}}, (d,s)) = \neg q_{R''(\mathbf{y})}^{\nu'}$ and then by definition of the automaton there exists a child π' of π labeled by $\neg(q_{R''(\mathbf{y})}^{\nu'}, m)$ with $\zeta(R'') < i$ and there exists no run starting at node m in state $q_{R''(\mathbf{y})}^{\nu'}$. So by using (ii) of the induction on the strata we have $R''(\mathrm{dec}_m(\nu'(\mathbf{y}))) \notin P(I)$. We define μ by $\mu(\mathbf{y}) = \mathrm{dec}_m(\nu'(\mathbf{y}))$, which effectively defines it because $\mathrm{vars}(\mathcal{A}) = \mathbf{y}$, and the compatibility conditions are satisfied.

- π may be labeled by $(q_r^{\nu',\mathcal{A}}, m)$, with $|\mathcal{A}| \geqslant 2$. We need to prove (iii). Given the definition of $\Delta(q_r^{\nu',\mathcal{A}}, (d,s))$ and by definition of the run, one of the following holds:

  - There exists $m' \in \mathrm{Nbh}(m)$ and a child π' of π such that π' is labeled by $(q_r^{\nu',\mathcal{A}}, m')$. By induction there exists $\mu' : \mathrm{vars}(\mathcal{A}) \to \mathrm{Dom}(I)$ satisfying (iii) for node m'. We can take μ to be μ', which satisfies the required properties.
  - There exist $(m_1, m_2) \in \mathrm{Nbh}(m) \times \mathrm{Nbh}(m)$ and π_1, π_2 children of π and non-empty sets $\mathcal{A}_1, \mathcal{A}_2$ that partition $\mathcal{A}$ and a total valuation ν'' of $\mathrm{vars}(\mathcal{A}_1) \cap \mathrm{vars}(\mathcal{A}_2)$ with values in d such that π_1 is labeled by $(q_r^{\nu' \cup \nu'',\mathcal{A}_1}, m_1)$ and π_2 is labeled by $(q_r^{\nu' \cup \nu'',\mathcal{A}_2}, m_2)$. By induction there exists $\mu_1 : \mathrm{vars}(\mathcal{A}_1) \to \mathrm{Dom}(I)$ and similarly μ_2 that satisfy (iii). Thanks to the compatibility conditions for μ_1 and μ_2 and to Property 3.7.3 applied to m_1 and m_2 via m, we can define $\mu : \mathrm{vars}(\mathcal{A}) \to \mathrm{Dom}(I)$ with $\mu := \mu_1 \cup \mu_2$. One can check that μ satisfies the required properties.

Hence, the forward direction of our equivalence lemma is proven.

Backward direction. We now prove the backward direction of the induction on strata of our main equivalence lemma (Lemma 3.7.6). From the induction hypothesis on strata, we know that, for every relation R with $\zeta(R) \leqslant i-1$, for every node $n \in T$ and partial valuation ν of $\mathbf{x}$, there exists a run ρ of A'_P starting at node n in state $q_{R(\mathbf{x})}^{\nu}$ if and only if there exists $\mathbf{c}$ such that $R(\mathbf{c}) \in P(I)$ and $\mathbf{c}$ is compatible with ν at node n. Let R be a relation with $\zeta(R) = i$, let $n \in T$ be a node

and let ν be a partial valuation of $\mathbf{x}$ such that there exists $\mathbf{c}$ such that $R(\mathbf{c}) \in P(I)$ and $\mathbf{c}$ is compatible with ν at node n. We need to show that there exists a run ρ of A'_P starting at node n in state $q^{\nu}_{R(\mathbf{x})}$. We will prove this by induction on the smallest $j \in \mathbb{N}$ such that $R(\mathbf{c}) \in \Xi^j_P(P_{i-1}(I))$, where Ξ^j_P is the j-th application of the immediate consequence operator for the program P (see [Abiteboul, Hull, and Vianu 1995]) and P_{i-1} is the restriction of P with only the rules up to strata $i-1$. The base case, when $j = 0$, is in fact vacuous since $R(\mathbf{c}) \in \Xi^0_P(P_{i-1}(I)) = P_{i-1}(I)$ implies that $\zeta(R) \leqslant i-1$, whereas we assumed $\zeta(R) = i$. For the inductive case ($j \geqslant 1$), we have $R(\mathbf{c}) \in \Xi^j_P(P_{i-1}(I))$ so by definition of the semantics of P, there is a rule r of the form $R(\mathbf{z}) \leftarrow L_1(\mathbf{y}_1) \dots L_t(\mathbf{y}_t)$ of P and a mapping $\mu : \mathbf{y}_1 \cup \dots \cup \mathbf{y}_t \to \mathrm{Dom}(I)$ such that $\mu(\mathbf{z}) = \mathbf{c}$ and, for every literal L_l in the body of r:

- If $L_l(\mathbf{y}_l) = R_l(\mathbf{y}_l)$ is a positive literal, then $R_l(\mu(\mathbf{y}_l)) \in \Xi^{j-1}_P(P_{i-1}(I))$
- If $L_l(\mathbf{y}_l) = \neg R_l(\mathbf{y}_l)$ is a negative literal, then $R_l(\mu(\mathbf{y}_l)) \notin P_{i-1}(I)$

Hence because $\mathbf{c}$ is clique-guarded and by a well-known property of tree decompositions (see Lemma 1 of [Gavril 1974], Lemma 2 of [Bodlaender and Koster 2010]), there exists a node n' such that $\mathbf{c} \subseteq \mathrm{dom}(n')$. Let $n = n_1, n_2, \dots, n_p = n'$ be the nodes on the path from n to n', and (d_i, s_i) be $\lambda_E(n_i)$ for $1 \leqslant i \leqslant p$. By compatibility, for every x_j defined by ν we have $\mathrm{dec}_n(\nu(x_j)) = c_j$. But $\mathrm{dec}_n(\nu(x_j)) \in \mathrm{dom}(n)$ and $c_j \in \mathrm{dom}(m)$ so by Property 3.7.4, for every $1 \leqslant i \leqslant p$ we have $c_j \in \mathrm{dom}(m_i)$ and $\mathrm{enc}_{m_i}(c_j) = \mathrm{enc}_n(c_j) = \mathrm{enc}_n(\mathrm{dec}_n(\nu(x_j))) = \nu(x_j)$, so that $\nu(x_j) \in d_i$. We can then start to construct the run ρ starting at node n in state $q^{\nu}_{R(\mathbf{x})}$ as follows. The root π_1 is labeled by $(q^{\nu}_{R(\mathbf{x})}, n)$, and for every $2 \leqslant i \leqslant p$, π_i is the unique child of π_{i-1} and is labeled by $(q^{\nu}_{R(\mathbf{x})}, m_i)$. This part is valid because we just proved that for every i, there is no j such that y_j is defined by ν and $\nu(y_j) \notin d_j$. Now from $\pi_{n'}$, we continue the run by staying at node n' and building up the valuation, until we reach a total valuation ν' such that $\nu'(\mathbf{x}) = \mathrm{enc}_{n'}(\mathbf{c})$. Hence we now only need to build a run ρ' starting at node n' in state $q^{\nu'}_{R(\mathbf{x})}$.

To achieve our goal of building a run starting at node n' in state $q^{\nu'}_{R(\mathbf{x})}$, it suffices to construct a run starting at node n' in state $q_r^{\nu'', \{L_1, \dots, L_t\}}$, with $\nu'' = \mathrm{Hom}_{\mathbf{z},\mathbf{x}}(\nu')$. The first step is to take care of the literals of the rule and to prove that:

(i) If $L_l(\mathbf{y}_l) = R_l(\mathbf{y}_l)$ is a positive literal, then there exists a node m_l and a total valuation ν_l of $\mathbf{y}_l$ with $\mathrm{dec}_{m_l}(\nu_l(\mathbf{y}_l)) = \mu(\mathbf{y}_l)$ such that there exists a run ρ_l starting at node m_l in state $q^{\nu_l}_{R_l(\mathbf{y}_l)}$.

(ii) If $L_l(\mathbf{y}_l) = \neg R_l(\mathbf{y}_l)$ is a negative literal, then there exists a node m_l and a total valuation ν_l of $\mathbf{y}_l$ with $\mathrm{dec}_{m_l}(\nu_l(\mathbf{y}_l)) = \mu(\mathbf{y}_l)$ such that there exists a run ρ_l starting at node m_l in state $\neg q^{\nu_l}_{R_l(\mathbf{y}_l)}$.

We first prove (i). We have $R_l(\mu(\mathbf{y}_l)) \in \Xi^{j-1}_P(P_{i-1}(I))$, and because P is clique-frontier-guarded, there exists a node m such that $\mu(\mathbf{y}_l) \subseteq m$. We take m_l to be m and ν_l to be such that $\nu_l(\mathbf{y}_l) = \mathrm{enc}_{m_l}(\mu(\mathbf{y}_l))$. We then directly obtain (i) by induction hypothesis (on j). We then prove (ii). Because the negative literals are guarded in rule bodies, there exists a node m such that $\mu(\mathbf{y}_l) \subseteq m$. We take m_l to be m and ν_l to be such that $\nu_l(\mathbf{y}_l) = \mathrm{enc}_{m_l}(\mu(\mathbf{y}_l))$. We straightforwardly get (ii) using the induction on the strata of our equivalence lemma.

The second step is to use the runs ρ_l that we just constructed and to construct from them a run starting at node n' in state $q_r^{\nu'',\{L_1,\dots,L_t\}}$. We describe in a high-level manner how we build the run. Starting at node n, we partition the literals to prove (i.e., the atoms of the body of the rule that we are applying), in the following way:

- We create one class in the partition for each literal R_l (which can be intentional or extensional) such that m_l is n', which we prove directly at the current node. Specifically, we handle these literals one by one, by splitting the remaining literals in two using the transition formula corresponding to the rule and by staying at node n' and building the valuations according to $\mathrm{dec}_n(\mu)$.

- For the remaining literals, considering all neighbors of n' in the tree encoding, we split the literals into one class per neighbor n'', where each literal L_l is mapped to the neighbor that allows us to reach its node m_l. We ignore the empty classes. If there is only one class, i.e., we must go in the same direction to prove all facts, we simply go to the right neighbor n'', remaining in the same state. If there are multiple classes, we partition the facts and prove each class on the correct neighbor.

  One must then argue that, when we do so, we can indeed choose the image by ν'' of all elements that were shared between literals in two different classes and were not yet defined in ν''. The reason why this is possible is because we are working on a tree encoding: if two facts of the body share a variable x, and the two facts will be proved in two different directions, then the variable x must be mapped to the same element in the two direction (namely, $\mu(x)$), which implies that it must occur in the node where we split. Hence, we can indeed choose the image of x at the moment when we split. □

FPT-linear time construction. Finally, we justify that we can construct in FPT-linear time the automaton A_P which recognizes the same language as A'_P. The size of $\Gamma_\sigma^{k_\mathrm{I}}$ only depends on k_I and on the extensional signature, which are fixed. As the number of states is linear in $|P|$, the number of transitions is linear in $|P|$. Most of the transitions are of constant size, and in fact one can check that the only problematic transitions are those for states of the form $q^\nu_{R(\mathbf{x})}$ with R intensional, specifically the second bullet point. Indeed, we have defined a transition from $q^\nu_{R(\mathbf{x})}$, for each valuation ν of a rule body, to the $q_r^{\nu',\mathcal{A}}$ for linearly many rules, so in general there are quadratically many transitions.

However, it is easy to fix this problem: instead of having one state $q^\nu_{R(\mathbf{x})}$ for every occurrence of an intensional predicate $R(\mathbf{x})$ in a rule body of P and total valuation ν of this rule body, we can instead have a constant number of states $q_{R(\mathbf{a})}$ for $\mathbf{a} \in \mathcal{D}_{k_\mathrm{I}}^{\mathrm{arity}(R)}$. In other words, when we have decided to prove a single intensional atom in the body of a rule, instead of remembering the entire valuation of the rule body (as we remember ν in $q^\nu_{R(\mathbf{x})}$), we can simply forget all other variable values, and just remember the tuple which is the image of $\mathbf{x}$ by ν, as in $q_{R(\mathbf{a})}$. Remember that the number of such states is only a function of k_P and k_I, because bounding k_P implies that we bound the arity of P, and thus the arity of intensional predicates.

We now redefine the transitions for those states :

- If there is a j such that $a_j \notin d$, then $\Delta(q_{R(\mathbf{a})}, (d, s)) = \bot$.

- Else, $\Delta(q_{R(\mathbf{a})}, (d, s))$ is a disjunction of all the $q_r^{\nu', \mathcal{A}}$ for each rule r such that the head of r is $R(\mathbf{y})$, $\nu'(\mathbf{y}) = \mathbf{a}$ and $\mathcal{A}$ is the set of all literals in the body of r.

The key point is that a given $q_r^{\nu', \mathcal{A}}$ will only appear in rules for states of the form $q_{R(\mathbf{a})}$ where R is the predicate of the head of r, and there is a constant number of such states.

We also redefine the transitions that used these states:

- Else, if $\mathcal{A} = \{R'(\mathbf{y})\}$ with R' intensional, then $\Delta(q_r^{\nu, \mathcal{A}}, (d, s)) = q_{R'(\nu(\mathbf{y}))}$.
- Else, if $\mathcal{A} = \{\neg R'(\mathbf{y})\}$ with R' intensional, then $\Delta(q_r^{\nu, \mathcal{A}}, (d, s)) = \neg q_{R'(\nu(\mathbf{y}))}$.

A_P recognizes the same language as A'_P. Indeed, consider a run of A'_P, and replace every state $q^{\nu}_{R(\mathbf{x})}$ with R intensional by the state $q_{R(\nu(\mathbf{x}))}$: we obtain a run of A_P. Conversely, being given a run of A_P, observe that every state $q_{R(\mathbf{a})}$ comes from a state $q_r^{\nu, \{R(\mathbf{y})\}}$ with $\nu(\mathbf{y}) = \mathbf{a}$. We can then replace $q_{R(\mathbf{a})}$ by the state $q^{\nu}_{R(\mathbf{x})}$ to obtain a run of A'_P.

3.7.2 Managing Unguarded Negations

We now explain how the translation can be extended to the full CFG-Datalog fragment. We recall that the difference with CFG$^{\text{GN}}$-Datalog is that negative literals in rule bodies no longer need to be clique-guarded. Remember that clique-frontier-guardedness was used in the translation of CFG$^{\text{GN}}$-Datalog to ensure the following property: when the automaton is trying to prove a rule $r := R(\mathbf{z}) \leftarrow L_1(\mathbf{y}_1) \dots L_t(\mathbf{y}_t)$ at some node n, i.e., when it is in a state $q_r^{\nu, \mathcal{A}}$ at node n for some subset $\mathcal{A}$ of literals of the body of r and partial valuation ν of the variables of $\mathcal{A}$, then, for each literal $L_l(\mathbf{y}_l)$ for $1 \leqslant l \leqslant t$, the images of $\mathbf{y}_l$ all appear together in a bag. More formally, let $\mu : \text{vars}(r) \to \text{dom}(I)$ be a mapping with $\mu(\mathbf{z}) = \text{dec}_n(\nu(\mathbf{z}))$ that witnesses that $R(\mu(\mathbf{z})) \in P(i)$: that is, if $L_l(\mathbf{y}_l)$ is a positive literal $S_l(\mathbf{y}_l)$ then we have $S_l(\mu(\mathbf{y_l})) \in P(i)$ and if $L(\mathbf{y}_l)$ is a negative literal $\neg S_l(\mathbf{y}_l)$ then we have $S_l(\mu(\mathbf{y}_l)) \notin P(I)$. In this case, we know that each $\mu(\mathbf{y}_l)$ must be contained in a bag of the tree decomposition.

This property is still true in CFG-Datalog when $L(\mathbf{y}_l)$ is a positive literal $S_l(\mathbf{y}_l)$. Indeed, if S is an extensional relation then the fact $S(\mu(\mathbf{y}_l))$ is encoded somewhere in the tree encoding, hence $\mu(\mathbf{y}_l)$ is contained in a bag of the tree decomposition. If S is an intensional predicate then, because P is clique-frontier-guarded, $\mu(\mathbf{y}_l)$ is also contained in a bag. However, when $L(\mathbf{y}_l)$ is a negative literal $\neg S_l(\mathbf{y}_l)$, it is now possible that $\mu(\mathbf{y}_l)$ is not contained in any bag of the tree decomposition. This can be equivalently rephrased as follows: there are $y_i, y_j \in \mathbf{y}_l$ with $y_i \neq y_j$ such that the occurrence subtrees of $\mu(y_i)$ and that of $\mu(y_j)$ are disjoint. If this happens, the automaton that we construct in the previous proof no longer works: it cannot assign the correct values to y_i and y_j, because once a value is assigned to y_i, the automaton cannot leave the occurrence subtree of $\mu(y_i)$ until a value is also assigned to y_j, which is not possible if the occurrence subtrees are disjoint.

To circumvent this problem, we will first rewrite the CFG Datalog program P into another program (still of body size bounded) which intuitively distinguishes between two kinds of negations: the negative atoms that will hold as in the case of clique-guarded negations in CFG$^{\text{GN}}$-Datalog, and the ones that will hold because two variables have disjoint occurrence subtrees. First, we create a vacuous unary

fact Adom, we modify the input instance and tree encoding in linear time to add the fact Adom for every element a in the active domain, and we modify P in linear time: for each rule r, for each variable x in the body of r, we add the fact $\text{Adom}(x)$. This ensures that each variable of rule bodies occurs in at least one positive fact.

Second, we rewrite P to a different program P'. Let r be a rule of P, and let $\mathcal{N}$ be the set of negative atoms in the body of r. Let $\mathcal{N}_{\text{G}} \cup \mathcal{N}_{\neg\text{G}}$ be a partition of $\mathcal{N}$ (where the classes in the partition may be empty), intuitively distinguishing the guarded and unguarded negations. For every atom of $\mathcal{N}_{\neg\text{G}}$, we nondeterministically choose a pair (y_i, y_l) of distinct variables of this atom, and consider the undirected graph $\mathcal{G}$ formed by the edges $\{y_i, y_l\}$ (we may choose the same edge for two different atoms). The graph $\mathcal{G}$ intuitively describes the variables that must be mapped to elements having disjoint occurrence subtrees in the tree encoding. For each rule r or P, for each choice of $\mathcal{N}_{\text{G}} \cup \mathcal{N}_{\neg\text{G}}$ and $\mathcal{G}$, we create a rule $r_{\mathcal{N}_{\text{G}},\mathcal{N}_{\neg\text{G}},\mathcal{G}}$ defined as follows: it has the same head as r, and its body contains the positive atoms of the body of r (including the Adom-facts) and the negative atoms of $\mathcal{N}_{\text{G}}$. We call $\mathcal{G}$ the *unguardedness graph of* $r_{\mathcal{N}_{G},\mathcal{N}_{\neg G},\mathcal{G}}$. Note that the semantics of P' will defined relative to the instance and also relative to the tree encoding of the instance that we consider: specifically, a rule can fire if there is a valuation that satisfies it in the sense of CFG^{GN}-Datalog (i.e., for all atoms, including negative atoms, all variables must be mapped to elements that occur together in some node), and which further respects the unguardedness graph, i.e., for any two variables $x \neq y$ with an edge in the graph, the elements to which they are mapped must have disjoint occurrence subtrees in the tree encoding. Note that we can compute P' from P in FPT-linear time parameterized by the body size, because the number of rules created in P' for each rule of P can be bounded by the body size; further, the bound on the body size of P' only depends on that of P, specifically it only increases by the addition of the atoms $\text{Adom}(x)$.

The translation of P' can now be done as in the case of CFG^{GN}-Datalog that we presented before; the only thing to explain is how the automaton can ensure that the semantics of the unguardedness graph is satisfied. To this end, we will first make two general changes to the way that our automaton is defined, and then present the specific tweaks to handle the unguardedness graph. The two general changes can already be applied to the original automaton construction that we presented, without changing its semantics.

The first change is that, instead of isotropic automata, we will use automata that take the directions of the tree into account, as in [Cachat 2002] for example (with stratified negation as we do for SATWAs). Specifically, we change the definition of the transition function. Remember that a SATWA has a transition function $\Delta : \mathcal{Q} \times \Gamma \to \mathcal{B}(\mathcal{Q})$ that maps each pair of a state and a label to a propositional formula on states of $\mathcal{Q}$. To handle directions, Δ will instead map to a propositional formula on pairs of states of $\mathcal{Q}$ and of directions in the tree, in $\{\bullet, \uparrow, \leftarrow, \rightarrow\}$. The intuition is that the corresponding state is only evaluated on the tree node in the specified direction (rather than on any arbitrary neighbor). We will use these directions to ensure that, while the automaton considers a rule application and navigates to find the atoms used in the rule body, then it never visits the same node twice. Specifically, consider two variables y_i and y_j that are connected by an edge in the unguardedness graph, and imagine that we first assign a value a to y_i in some node n. To assign a value to y_j, we must leave the occurrence subtree of the current

a in the tree encoding, and must choose a value outside of this occurrence subtree. Thus, the automaton must "remember" when it has left the subtree of occurrences of a, so that it can choose a value for y_j. However, an isotropic automaton cannot "remember" that it has left the subtree of occurrences of a, because it can always come back on a previously visited node, by going back in the direction from which it came. However, using SATWAs with directions, and changing the automata states to store the last direction that was followed, we can ensure that the automaton cannot come back to a previously visited node (while locating the facts that correspond to the body of a rule application). This ensures that, once the automaton has left the subtree of occurrences of an element, then it cannot come back in this subtree again while it is considering the same rule application. Hence, the first general change is that we use SATWAs with directions, and we use the directions to ensure that the automaton does not go back to a previously visited node while considering the same rule application. In fact, this first general change does not change the semantics of the automaton, as in the case of isotropic automata, we did not really need the ability to turn around when trying to prove a rule.

The second general change that we perform on the automaton is that, when guessing a value for an undefined variable, then we only allow the guess to happen as early as possible. In other words, suppose the automaton is at a node n in the tree encoding while it was previously at node n'. Then it can assign a value $a \in n$ to some variable y only if a was not in n', i.e., a has just been introduced in n. Obviously an automaton can remember which elements have been introduced in this sense, and which elements have not. This change can be performed on our existing construction without changing the semantics of the automaton, by only considering runs where the automaton assigns values to variables at the moment when it enters the occurrence subtree of this element.

Having done these general changes, we will simply reuse the previous automaton construction (not taking the unguardedness graph $\mathcal{G}$ into account) on the program P', and make two tweaks to ensure that the unguardedness graph is respected. The first tweak is that, in states of the form $q_r^{\nu,\mathcal{A}}$, the automaton will also remember, for each undefined variable x (i.e., x is in the domain of ν but $\nu(x)$ is still undefined), a set $\beta(x)$ of *blocking elements* for x, which are elements of the tree encoding. While $\beta(x)$ is non-empty, then the automaton is not allowed to guess a value for x, intuitively because we know that it is still in the occurrence subtree of some previously mapped variable $y \in \beta(x)$ which is adjacent to x in $\mathcal{G}$. Here is how these sets of blocking elements are computed and updated:

- When the automaton starts to consider the application of a rule r, then $\beta(x) := \emptyset$ for each variable x of the body of r.
- When the automaton guesses a value a for a variable x, then for every undefined variable y, if x and y are adjacent in $\mathcal{G}$, then we set $\beta(y) := \beta(y) \cup \{a\}$, intuitively adding a to the set of blocking elements for y. This intuitively means that the automaton is not allowed to guess a value for y until it has exited the subtree of occurrences of a. Note that, if the automaton wishes to guess values for multiple variables while visiting one node (in particular when partitioning the literals of $\mathcal{A}$), then the blocking sets are updated between each guess: this implies in particular that, if there is an edge in $\mathcal{G}$ between two variables x and y, then the automaton can never guess the value for x and for y at the same

node.

- When the automaton navigates to a new node n' of the tree encoding, then for every variable x in the domain of ν which does not have an image yet, we set $\beta(x) := \beta(x) \cap n'$. Intuitively, when an element a was blocking for x but disappears from the current node, then a is no longer blocking. Note that $\beta(x)$ may then become empty, meaning that the automaton is now free to guess a value for x.

The blocking sets ensure that, when the automaton guesses a value for x, then this value is guaranteed not to occur in the occurrence subtree of variables that are adjacent to x in $\mathcal{G}$ and have been guessed before. This also relies on the second general change above: we can only guess values for variables as early as possible, i.e., we can only use elements in guesses when we have just entered their occurrence subtree, so when $\beta(x)$ becomes empty then the possible guesses for x do not include any element whose occurrence subtree intersects that of $\nu(y)$ for any variable y adjacent to x in $\mathcal{G}$.

The second tweak is that, when we partition the set of literals to be proven, then we use the directionality of the automaton to ensure that the remaining literals are split across the various directions (having at most one run for every direction). For instance, considering the rule body $\{\mathrm{Adom}(x), \mathrm{Adom}(y)\}$ and the unguardedness graph $\mathcal{G}$ having an edge between x and y, the automaton may decide at one node to partition $\mathcal{A} = \{\mathrm{Adom}(x), \mathrm{Adom}(y)\}$ into $\{\mathrm{Adom}(x)\}$ and $\{\mathrm{Adom}(y)\}$, and these two subsets of facts will be proven by two independent runs: these two runs are required to go in different directions of the tree. This will ensure that, even though the edge $\{x, y\}$ of $\mathcal{G}$ will not be considered explicitly by either of these runs (because the domain of their valuations will be $\{x\}$ and $\{y\}$ respectively), it will still be the case that x and y will be mapped to elements whose occurrence subtrees do not intersect: this is again using the fact that we map elements as early as possible.

We now summarize how the modified construction works:

1. Assume that the automaton A is at some node n in state $q^{\nu''}_{R(\mathbf{x})}$, with ν'' being total in $\mathbf{x}$.

2. At node n, the automaton chooses a rule $r' : R(\mathbf{z}) \leftarrow L_1(\mathbf{y}_1) \dots L_t(\mathbf{y}_t)$ of P' and goes to state $q^{\nu,\mathcal{A}}_{r'}$ where $\nu := \mathrm{Hom}_{\mathbf{z},\mathbf{x}}(\nu'')$ and $\mathcal{A}$ is the set of literals in the body of r'. That is, it simply chooses a rule to prove $R(\nu''(\mathbf{x}))$. This amounts to choosing a rule of the original program P, and choosing which negative atoms will be guarded (i.e., mapped to variables that occur together in some node of the tree encoding), and choosing the unguardedness graph $\mathcal{G}$ in a way to ensure that each unguarded negated atom has a pair of variables that forms an edge of G. The blocking set $\beta(x)$ of each variable x in the rule body is initialized to the empty set, and whenever the automaton will move to a different node n' then each element a that is no longer present in n' will be removed from $\beta(x)$ for each variable x, formally, $\beta(x) := \beta(x) \cap n'$.

3. From now on, assume the automaton A always remembers (stores in its state) which elements N have just been introduced in the current node of the tree encoding. That is, N is initialized with the elements in n, and when A goes from some node n' to node n'', N becomes $n'' \setminus n'$. When guessing values

for variables, the automaton will only use values in N, so as to respect the condition that we guess the value of variables as early as possible. This is how we implement our second general change.

4. While staying at node n, the automaton chooses some undefined variables x (i.e., variables in the domain of ν that do not have a value yet), and guesses some values in N for them, one after another. For each such variable x, we first verify that $\beta(x) = \emptyset$ (otherwise we fail), we set $\nu(x) := a$ where a is the guessed value, and then, for every edge $\{x, y\}$ in $\mathcal{G}$ such that y is an undefined variable (i.e., it is in the domain of ν but does not have an image by ν yet), we set $\beta(y) := \beta(y) \cup \{a\}$, ensuring that no value will be guessed for y until the automaton has left the subtree of occurrences for a. We call ν' the resulting new valuation.

5. While staying at node n, the automaton guesses a partition of $\mathcal{A}$ as $P_{\text{directions}} = (\mathcal{A}^\bullet, \mathcal{A}^\uparrow, \mathcal{A}^\leftarrow, \mathcal{A}^\rightarrow)$ to decide in which direction each one of the remaining facts is sent. Of course, if there is a direction for which n has no neighbor (e.g., $\leftarrow$ and $\rightarrow$ if n is a leaf, or $\uparrow$ if n is the root), then $\mathcal{A}^d$ in the corresponding direction d must be empty.

6. If $\mathcal{A}^\bullet$ is the only class that is not empty, meaning that all remaining facts will be witnessed at the current node, then go to step 10.

7. While staying at node n, the automaton checks that each variable x that appears in two different classes of $P_{\text{directions}}$ has been assigned a value, i.e., $\nu(x)$ is defined; otherwise, the automaton fails. This is to ensure that the partitioning is consistent (i.e., that a variable x will not be assigned different values in different runs). The automaton also checks that $\nu(x)$ is defined for each variable occurring in $\mathcal{A}^\bullet$, that is, we assume without loss of generality that atoms that will be proven at the current node have all their variables already mapped. This ensures that the undefined variables are partitioned between directions in $\{\uparrow, \leftarrow, \rightarrow\}$.

8. The automaton then launches a run $q_{r'}^{\nu', \mathcal{A}^d}$ for each direction $d \in \{\bullet, \uparrow, \leftarrow, \rightarrow\}$ at the corresponding node (n for $\bullet$, the parent of n for $\uparrow$, the left of right child of n for $\leftarrow$ or $\rightarrow$).

9. For each of these runs, we update the value of N and of the blocking sets, and we go back to step 4, except that now A remembers the direction from which it comes, and does not go back to the previously visited node. For example if the automaton goes from some node n to the parent n' of n such that n is the left child of n', then in the partition that will be guessed at step 5 we will have $\mathcal{A}^\leftarrow = \emptyset$. Further, in each of these runs, of course, the automaton remembers the values of the blocking sets $\beta(x)$ for each undefined variable x.

10. Check that all the variables have been assigned. Launch positive states for each positive intensional literal and negative states for each negative literal, i.e., start from step 1: in this case, when the automaton proves a different rule application, then of course it forgets the values of the blocking sets, and forgets the previous direction (i.e., it can again visit the entire tree from the node

where it starts). For each positive extensional literal, simply check that the atom is indeed encoded in the current node of the tree encoding.

All these modifications can be implemented in FPT-linear time provided that the arity of P is bounded, which is the case because the body size of P is bounded. Moreover, as we pointed out after the proof of Theorem 3.6.11, the construction of provenance cycluits can easily be modified to work for stratified alternating two way automata with directions, so that all our results about CFG-Datalog (evaluation and provenance cycluit computation in FPT linear time) still hold on this modified automaton.

Conclusion. This finishes the proof of translation and the presentation of our work on the combined tractability of the evaluation of CFG-Datalog queries on treelike instances. In the next chapter, we return to the problem of probabilistic query evaluation, and investigate how the provenance of CFG-Datalog programs can be compiled into tractable formalisms that allow for efficient probability computation.

Chapter 4

From Cycluits to d-DNNFs and Lower Bounds

This chapter of my thesis presents my work with Antoine Amarilli and Pierre Senellart on the connections between various circuit classes from knowledge compilation. We also show the consequences of these connections for PQE, and in particular for our language of CFG-Datalog from Chapter 3. Excepted Section 4.5, which is part of [Amarilli, Bourhis, Monet, and Senellart 2017], all results presented here were published at ICDT'2018 [Amarilli, Monet, and Senellart 2018].

4.1 Introduction

As previously discussed, a common technique to evaluate queries on probabilistic databases is the intensional approach: first compute a representation of the lineage of the query on the database, which intuitively describes how the query depends on the possible database facts; then use this lineage to compute probabilities efficiently. Specifically, the lineage can be computed as a Boolean circuit [Jha and Suciu 2012], and efficient probability computation can be achieved by restricting to tractable circuit classes via knowledge compilation. Thus, to evaluate queries on probabilistic databases, we can use knowledge compilation algorithms to translate circuits to tractable classes; conversely, lower bounds in knowledge compilation can identify the limits of the intensional approach.

In this chapter, we study the relationship between two kinds of tractable circuit classes in knowledge compilation: *width-based* classes, specifically, bounded-treewidth and bounded-pathwidth circuits; and *structure-based* classes, specifically, OBDDs (following a *variable order*) and d-SDNNFs (following a *v-tree*), as defined in Section 1.7. Circuits of bounded treewidth can be obtained as a result of practical query evaluation (e.g., in [Jha, Olteanu, and Suciu 2010; Amarilli, Bourhis, and Senellart 2015]), whereas OBDDs and d-DNNFs have been studied to show theoretical characterizations of the query lineages they can represent [Jha and Suciu 2011]. Both classes enjoy tractable probabilistic computation: for width-based classes, using *message passing* [Lauritzen and Spiegelhalter 1988], in time linear in the circuit and exponential in the treewidth; for OBDDs and d-SDNNFs, in linear time by definition of the class [Darwiche 2001]. Hence the question that we study: can we compile width-based classes efficiently into structure-based classes?

We first study how to perform this transformation, and show corresponding *upper*

bounds. Existing work has already studied the compilation of bounded-pathwidth circuits to OBDDs [Jha and Suciu 2012], which can be made constructive [Amarilli, Bourhis, and Senellart 2016, Lemma 6.9]. Accordingly, we focus on compiling *bounded-treewidth circuits* to *d-SDNNF circuits*. Our first contribution, stated in Section 4.3 and proved in Section 4.4, is to show the following:

Result 1 (Theorem 4.3.2 and subsequent remark)**.** *Given as input a Boolean circuit C of treewidth k, we can compute a d-SDNNF equivalent to C in time $O(|C| \times f(k))$ where f is singly exponential.*

The algorithm transforms the input circuit bottom-up, considering all possible valuations of the gates in each bag of the tree decomposition, and keeping track of additional information to remember which guessed values have been substantiated by a corresponding input. Our result relates to a recent theorem of Bova and Szeider in [Bova and Szeider 2017], except that our bound depends on $|C|$ (the circuit size) whereas their bound depends on the number of variables of C. In exchange for this, we improve on their result in two ways. First, our result is constructive, whereas the bound of [Bova and Szeider 2017] only shows a bound on the size of the d-SDNNF, without bounding the complexity of effectively computing it. Second, our bound is singly exponential in k, whereas [Bova and Szeider 2017] is doubly exponential; this allows us to be competitive with message passing (also singly exponential in k), and we believe it can be useful for practical applications. Indeed, beyond probabilistic query evaluation, our result implies that all tractable tasks on d-SDNNFs (e.g., enumeration [Amarilli, Bourhis, Jachiet, and Mengel 2017] and MAP inference [Fierens et al. 2015]) are also tractable on bounded-treewidth circuits.

Second, we show in Section 4.5 how we can then use this result to lift the combined tractability of CFG-Datalog on bounded-treewidth databases (i.e., Theorem 3.4.4 of Chapter 3) to the case of probabilistic query evaluation. Indeed, the treewidth of the constructed provenance cycluit is linear in the size of the Datalog query. However, we cannot directly apply Result 1, because here we have a *cycluit*. Hence the first step is to transform this cycluit into a circuit, while preserving some kind of bound on the treewidth. We use an algorithm from [Amarilli, Bourhis, Monet, and Senellart 2017] that converts a cycluit of treewidth k into an equivalent circuit whose treewidth is singly exponential in k, in FPT-linear time (parameterized by k). We can then apply our transformation of Result 1 to this circuit, which in the end shows that we can compute a d-SDNNF representation of the provenance of a CFG-Datalog program of bounded rule-size on an instance of bounded treewidth with a complexity linear in the data and doubly exponential in the program:

Result 2 (Theorem 4.5.4)**.** *Fix $k_{\mathrm{I}}, k_{\mathrm{P}} \in \mathbb{N}$. There is an $\alpha \in \mathbb{N}$ s.t., given a CFG-Datalog program P of body size k_{P} and a TID instance (I, π) of treewidth k_{I}, we can compute a d-SDNNF representing the provenance of P on I in time $O(2^{2^{|P|^\alpha}} |I|)$.*

This d-SDNNF then allows us to solve PQE for this program on this instance with the same complexity. While the complexity in the query is not polynomial, we consider that it is a reasonable one, given that our language of CFG-Datalog is quite expressive (at least in comparison to the restricted CQ fragments that we studied in Chapter 2). Moreover, the complexity is linear in the data, which is crucial for practical applications. Also, we recall that the previous approach to PQE for

expressive queries on treelike instances (i.e., MSO queries in [Amarilli, Bourhis, and Senellart 2015]), has a complexity nonelementary in the query.

Third, we study *lower bounds* on how efficiently we can convert from width-based to structure-based classes. Our bounds already apply to a weaker formalism of width-based circuits, namely monotone CNFs and DNFs of bounded width, so we focus on them. Our third contribution (in Section 4.6) concerns pathwidth and OBDD representations: we show that, up to factors in the formula arity (maximal size of clauses) and degree (maximal number of variable occurrences), any OBDD for a monotone CNF or DNF must be of width exponential in the pathwidth of the formula. Formally:

Result 3 (Theorem 4.6.2)**.** *Let φ be a monotone DNF or monotone CNF, let $a := \mathrm{arity}(\varphi)$ and $d := \mathrm{degree}(\varphi)$. Then any OBDD for φ has width $2^{\Omega\left(\frac{\mathrm{pw}(\varphi)}{a^3 \times d^2}\right)}$.*

This result generalizes several existing lower bounds in knowledge compilation that exponentially separate CNFs from OBDDs, such as [Devadas 1993] and [Bova and Slivovsky 2017, Theorem 19].

We further show (Section 4.7) an analogue for treewidth and (d-)SDNNFs:

Result 4 (Theorem 4.7.1)**.** *Let φ be a monotone DNF (resp., monotone CNF), let $a := \mathrm{arity}(\varphi)$ and $d := \mathrm{degree}(\varphi)$. Then any d-SDNNF (resp., SDNNF) for φ has size $2^{\Omega\left(\frac{\mathrm{tw}(\varphi)}{a^3 \times d^2}\right)}$.*

Our two lower bounds contribute to a vast landscape of knowledge compilation results giving lower bounds on compiling specific Boolean functions to restricted circuits classes, e.g., [Devadas 1993; Razgon 2014; Bova and Slivovsky 2017] to OBDDs, [Calí, Capelli, and Razgon 2017] to *decision* structured DNNF, [Beame and Liew 2015] to *sentential decision diagrams* (SDDs), [Pipatsrisawat and Darwiche 2010; Bova, Capelli, Mengel, and Slivovsky 2016] to d-SDNNF, [Bova, Capelli, Mengel, and Slivovsky 2016; Capelli 2016; Capelli 2017] to d-DNNFs and DNNFs. However, all those lower bounds (with the exception of some results in [Capelli 2016; Capelli 2017] discussed in Section 4.7) apply to well-chosen families of Boolean functions (usually CNF), whereas Result 3 and 4 apply to *any* monotone CNF and DNF. Together with Result 1, these generic lower bounds point to a strong relationship between width parameters and structure representations, on monotone CNFs and DNFs of constant arity and degree. Specifically, the smallest width of OBDD representations of any such formula φ is in $2^{\Theta(\mathrm{pw}(\varphi))}$, i.e., precisely singly exponential in the pathwidth; and an analogous bound applies to d-SDNNF size and treewidth of DNFs.

Example 4.1.1. We give here a concrete example to illustrate the genericity of our results. Let $h \in \mathbb{N}$, $h \geqslant 1$, and let $T = (V, E)$ be the complete binary tree of height h (i.e., the rooted full binary tree such that the path from any leaf to the root is of length h). Let F be the monotone CNF on variables V defined by $F := \bigwedge_{(v_1,v_2)\in E} v_1 \vee v_2$. Intuitively, for every valuation $\nu : V \to \{0,1\}$, we have $F(\nu) = 1$ if and only if $\{v \in V \mid \nu(v) = 1\}$ is a vertex cover of T. Then, the arity of F is 2, its degree is 3, and its pathwidth is $\lceil \frac{h}{2} \rceil$ [Marshall and Wood 2014]. Hence, by Theorem 4.6.2, letting w be the width of an OBDD for F, we have:

$$w \geqslant 2^{\left\lfloor \frac{h}{2 \times 2^3 \times 3^2} \right\rfloor}$$

Moreover, by [Bova and Slivovsky 2017, Theorem 4 and Lemma 9], there exists an OBDD for F of width $2^{\lceil \frac{h}{2} \rceil + 2}$. ◁

To prove our lower bounds, we rephrase pathwidth and treewidth to new notions of *pathsplitwidth* and *treesplitwidth*, which intuitively measure the performance of a variable ordering or v-tree. We also use the *disjoint non-covering prime implicant sets* (dncpi-sets), a tool introduced in [Amarilli, Bourhis, and Senellart 2016; Amarilli 2016], and generalizing *subfunction width* [Bova and Slivovsky 2017]. These dncpi-sets allow us to derive lower bounds on OBDD width directly using [Amarilli 2016]. We show how they can also imply lower bounds on d-SDNNF size, using the recent communication complexity approach of [Bova, Capelli, Mengel, and Slivovsky 2016].

Fourth, in Section 4.8, we apply our lower bounds to intensional query evaluation on relational databases. We reuse the notion of *intricate* queries of [Amarilli, Bourhis, and Senellart 2016], and show that d-SDNNF representations of the lineage of these queries have size exponential in the treewidth of *any* input instance. Intuitively, a query is intricate if, under some conditions, their lineage never has lower treewidth than the instance. We lift the result of [Amarilli, Bourhis, and Senellart 2016] from OBDDs to d-SDNNFs:

Result 5 (Theorem 4.8.4)**.** *There is a constant $d \in \mathbb{N}$ such that the following is true. Let σ be an arity-2 signature, and Q be a connected* $\mathrm{UCQ}^{\neq}$ *which is intricate on σ. For any instance I on σ, any d-SDNNF representing the lineage of Q on I has size $\geqslant 2^{\Omega(\mathrm{tw}(I)^{1/d})}$.*

As in [Amarilli, Bourhis, and Senellart 2016], this result shows that, on arity-2 signatures and under constructibility assumptions, treewidth is the right parameter on instance families to ensure that all queries (in monadic second-order) have tractable d-SDNNF lineage representations.

4.2 Preliminaries on Tree Decompositions

We start with short preliminaries, which are a local refinement of the general preliminaries from Chapter 1. In particular, Sections 1.5 and 1.7 from Chapter 1 will be of special importance for this chapter.

Hypergraphs, treewidth, pathwidth. Recall the definition of a hypergraph $H = (V, E)$ from Section 1.1. In this chapter we will only consider hypergraphs that do not contain the empty hyperedge (i.e., $\emptyset \notin E$). We will moreover always assume that hypergraphs have at least one edge.

Tree decompositions. Recall the definitions of treewidth and of a tree decomposition of a hypergraph from Section 1.5. For convenience, we will often identify a bag with its domain. We will need in this chapter a slightly more precise version of Theorem 1.5.2, which shows that computing a tree decomposition can be done in FPT-linear time when parameterizing by the treewidth:

Theorem 4.2.1 ([Bodlaender 1996])**.** *There exists a fixed function $g(k)$ that is in $O(2^{(32+\varepsilon)k^3})$ for any $\varepsilon > 0$ such that the following is true. Given a hypergraph H and an integer $k \in \mathbb{N}$ we can check in time $O(|H| \times g(k))$ whether H has treewidth $\leqslant k$, and if yes output a tree decomposition of H of width $\leqslant k$.*

For simplicity, we will often assume that a tree decomposition is *v-friendly*, for a node $v \in V$, meaning that:

1. it is a full binary tree, i.e., each node has exactly zero or two children;
2. for every internal bag b with children b_l, b_r we have $b \subseteq b_l \cup b_r$;
3. for every leaf bag b we have $|b| \leqslant 1$;
4. the root bag of T only contains the node v.

Lemma 4.2.2. *Given a tree decomposition T of a hypergraph H of width k and a node v of H, we can compute in time $O(k \times |T|)$ a v-friendly tree decomposition T' of H of width k.*

Proof. We first create a bag b_{root} containing only the node v, and make this bag the root of T by connecting it to a bag of T that contains v (if there is no such bag then we connect b_{root} to an arbitrary bag of T). Then, we make the tree decomposition binary (but not necessarily full) by replacing each bag b with children $b_1, \ldots, b_n$ with $n > 2$ by a chain of bags with the same label as b to which we attach the children $b_1, \ldots, b_n$. This process is in time $O(|T|)$ and does not change the width.

We then ensure the second and third conditions, by applying a transformation to leaf bags and to internal bags. We first modify every leaf bag b containing more than one vertex by a chain of at most k internal bags with leaves where the vertices are added one after the other. Then, we modify every internal bag b that contains elements $v_1, \ldots, v_n$ not present in the union D of its children: we replace b by a chain of at most k internal bags $b'_1, \ldots, b'_n$ containing respectively $b, b\setminus\{v_n\}, b\setminus\{v_n, v_{n-1}\}, \ldots, D$, each bag having a child introducing the corresponding gate v_i. This is in time $O(k \times |T|)$, and again it does not change the width; further, the result of the process is a tree decomposition that satisfies the second, third and fourth conditions and is still a binary tree.

The only missing part is to ensure that the tree decomposition is full, which we can simply do in linear time by adding bags with an empty label as a second children for internal nodes that have only one child. This is obviously in linear time, does not change the width, and does not affect the other conditions, concluding the proof. □

DNFs and CNFs. For lower bounds, we will study representations of Boolean functions as DNFs and CNFs (defined in Section 1.7). In this chapter we always assume that monotone DNFs and monotone CNFs are *minimized*, i.e., no clause is a subset of another. This ensures that every monotone Boolean function has a unique representation as a monotone DNF (the disjunction of its prime implicants), and likewise for CNF. We will also assume that CNFs and DNFs always contain at least one non-empty clause (in particular, they cannot represent constant functions).

4.3 Upper Bound

Our upper bound result studies how to compile a Boolean circuit to a d-SDNNF, parameterized by the treewidth of the input circuit. To present it, we first review the independent result that was recently shown by Bova and Szeider [Bova and Szeider 2017] about these circuit classes:

Theorem 4.3.1 ([Bova and Szeider 2017, Theorem 3 and Equation (22)])**.** *Given a Boolean circuit C with n variables and of treewidth $\leqslant k$, there exists an equivalent d-SDNNF of size $O(f(k) \times n)$, where f is doubly exponential.*

An advantage of their result is that it depends only on the *number of variables* of the circuit (and on the width parameter), not on the *size* of the circuit. In contrast, we will always measure complexity as a function of the size of the input circuit. In exchange for this advantage, their result has two drawbacks: (i) it has a doubly exponential dependency on the width; and (ii) it is nonconstructive, because [Bova and Szeider 2017] gives no time bound on the computation, leaving open the question of effectively compiling bounded-treewidth circuits to d-SDNNFs.

Our bound. Our main upper bound result addresses both drawbacks and shows that we can compile in time linear in the circuit and singly exponential in the treewidth. Our proof is independent from the techniques of [Bova and Szeider 2017]. Formally, we show:

Theorem 4.3.2. *There exists a function $f(k)$ that is in $O(2^{(4+\varepsilon)k})$ for any $\varepsilon \geqslant 0$ such that the following holds. Given as input a Boolean circuit C and tree decomposition T of width $\leqslant k$ of C, we can compute a d-SDNNF equivalent to C with its v-tree, in time $O(|T| \times f(k))$.*

We prove Theorem 4.3.2 in the next section. Observe how we assume the tree decomposition to be given as part of the input. If it is not, we can compute one with Theorem 4.2.1, but this becomes the bottleneck: the complexity becomes $O\left(|C| \times 2^{(32+\varepsilon)k^3}\right)$ for any $\varepsilon > 0$.

Applications. Theorem 4.3.2 implies several consequences for bounded-treewidth circuits. The first one deals with the Boolean circuit probability computation problem, as defined by Section 1.7. This problem is #P-hard for arbitrary circuits, but it is tractable for d-DNNFs [Darwiche 2001], remembering that we assumed that arithmetic operations (sum and product) take unit time. Hence, our result implies the following, where $|\pi|$ denotes the size of writing the probability valuation π:

Corollary 4.3.3. *Let $f(k)$ be the function from Theorem 4.3.2. Given a Boolean circuit C, a tree decomposition T of width $\leqslant k$ of C, and a probability valuation π of C_{var}, we can compute $\pi(C)$ in time $O(|\pi| + |T| \times f(k))$.*

Proof. Use Theorem 4.3.2 to compute an equivalent d-SDNNF C'; as C and C' are equivalent, it is clear that $\pi(C) = \pi(C')$. Now, compute the probability $\pi(C')$ in linear time in C' and $|\pi|$ by a simple bottom-up pass, using the fact that C' is a d-DNNF [Darwiche 2001]. □

This improves the bound obtained when applying message passing techniques [Lauritzen and Spiegelhalter 1988] directly on the bounded-treewidth input circuit (as presented, e.g., in [Amarilli, Bourhis, and Senellart 2015, Theorem D.2]). Indeed, message passing applies to *moralized* representations of the input: for each gate, the tree decomposition must contain a bag containing all inputs of this gate *simultaneously*, which is problematic for circuits of large fan-in. Indeed, if the original circuit has a tree decomposition of width k, rewriting it to make it moralized will results

in a tree decomposition of width $3k^2$ (see [Amarilli, Bourhis, Monet, and Senellart 2017, Lemmas 53 and 55]), and the bound of [Amarilli, Bourhis, and Senellart 2015, Theorem D.2] then yields an overall complexity of $O\left(|\pi| + |T| \times 2^{3k^2}\right)$ for message passing. Our Corollary 4.3.3 achieves a more favorable bound because Theorem 4.3.2 uses directly the associativity of AND and OR. We note that the connection between message-passing techniques and structured circuits has also been investigated by Darwiche, but his result [Darwiche 2003, Theorem 6] produces arithmetic circuits rather than d-DNNFs, and it also needs the input to be moralized.

A second consequence concerns the task of *enumerating* the accepting valuations of circuits, i.e., producing them one after the other, with small *delay* between each accepting valuation. The valuations are concisely represented as *assignments*, i.e., as a set of variables that are set to true, omitting those that are set to false. This task is of course NP-hard on arbitrary circuits (as it implies that we can check whether an accepting valuation exists), but was recently shown in [Amarilli, Bourhis, Jachiet, and Mengel 2017] to be feasible on d-SDNNFs with linear-time preprocessing and delay linear in the Hamming weight of each produced assignment. Hence, we have:

Corollary 4.3.4. *Let $f(k)$ be the function from Theorem 4.3.2. Given a Boolean circuit C and a tree decomposition T of width $\leqslant k$ of C, we can enumerate the accepting assignments of C with preprocessing in $O\left(|T| \times f(k)\right)$ and delay linear in the size of each produced assignment.*

Proof. Use Theorem 4.3.2 to compute an equivalent d-SDNNF C', which has the same accepting valuations, along with a v-tree T' of C'. We now conclude using [Amarilli, Bourhis, Jachiet, and Mengel 2017, Theorem 2.1]. □

Other applications of Theorem 4.3.2 include counting the number of satisfying valuations of the circuit (a special case of probability computation), MAP inference [Fierens et al. 2015] or random sampling of possible worlds (which can be done on the d-SDNNF in an easy manner).

4.4 Proof of the Upper Bound

We first present in Section 4.4.1 the construction used for Theorem 4.3.2, then prove in Section 4.4.2 that this construction is correct and can be done within the prescribed time bound.

4.4.1 Construction

We start with prerequisites, and then describe how to build the d-SDNNF equivalent to the input bounded-treewidth circuit.

Prerequisites. Let C be the input circuit, and T the input tree decomposition of width $\leqslant k$ of C. Let g_{output} be the output gate of C. Thanks to Lemma 4.2.2, we can assume that T is g_{output}-friendly. For every variable gate $x \in C_{\mathsf{var}}$, we chose a leaf bag b_x of T such that $\lambda(b_x) = \{x\}$. Such a leaf bag exists because T is friendly (specifically, thanks to bullet points 2 and 3). We say that b_x is *responsible for the variable gate x*. We can obviously chose such a b_x for every variable gate x in linear time in T.

To abstract away the type of gates and their values in the construction, we will talk of *strong* and *weak* values. Intuitively, a value is *strong* for a gate g if any input g' of g which carries this value determines the value of g; and *weak* otherwise. Formally:

Definition 4.4.1. Let g be a gate and $c \in \{0,1\}$:

- If g is an AND-gate, we say that $c = 0$ is *strong* for g and $c = 1$ is *weak* for g;
- If g is an OR-gate, we say that $c = 1$ is *strong* for g and $c = 0$ is *weak* for g;
- If g is a NOT-gate, $c = 0$ and $c = 1$ are both *strong* for g;
- For technical convenience, if g is a var-gate, $c = 0$ and $c = 1$ are both *weak* for g. ◁

If we take any valuation $\nu : C_{\mathsf{var}} \to \{0,1\}$ of the circuit $C = (G, W, g_{\mathsf{output}}, \mu)$, and extend it to an evaluation $\nu : G \to \{0,1\}$, then ν will respect the semantics of gates. In particular, it will *respect strong values*: for any gate g of C, if g has an input g' for which $\nu(g')$ is a strong value, then $\nu(g)$ is determined by $\nu(g')$, specifically, it is $\nu(g')$ if g is an OR- or an AND-gate, and $1 - \nu(g')$ if g is a NOT-gate. In our construction, we will need to guess how gates of the circuit are evaluated, focusing on a subset of the gates (as given by a bag of T); we will then call *almost-evaluation* an assignment of these gates that respects strong values. Formally:

Definition 4.4.2. Let U be a set of gates of C. We call $\nu : U \to \{0,1\}$ a (C,U)-*almost-evaluation* if it *respects strong values*, i.e., for every gate $g \in U$, if there is an input g' of g in U and $\nu(g')$ is a strong value for g, then $\nu(g)$ is determined from $\nu(g')$ in the sense above. ◁

Respecting strong values is a necessary condition for such an assignment to be extensible to a valuation of the entire circuit. However, it is not sufficient: an almost-evaluation ν may map a gate g to a strong value even though g has no input that can justify this value. This is hard to avoid: when we focus on the set U, we do not know about other inputs of g. For now, let us call *unjustified* the gates of U that carry a strong value that is not justified by ν:

Definition 4.4.3. Let U be a set of gates of a circuit C and ν a (C,U)-almost-evaluation. We call $g \in U$ *unjustified* if $\nu(g)$ is a strong value for g, but, for every input g' of g in U, the value $\nu(g')$ is weak for g; otherwise, g is *justified*. The set of unjustified gates is written $\mathrm{Unj}(\nu)$. ◁

Let us start to explain in a high-level manner how to construct the d-SDNNF D equivalent to the input circuit C (we will later describe the construction formally). We do so by traversing T bottom-up, and for each bag b of T we create gates $G_b^{\nu,S}$ in D, where ν is a (C,b)-almost-evaluation and S is a subset of $\mathrm{Unj}(\nu)$ which we call the *suspicious gates* of $G_b^{\nu,S}$. We will connect the gates of D created for each internal bag b with the gates created for its children in T, in a way that we will explain later. Intuitively, for a gate $G_b^{\nu,S}$ of D, the *suspicious gates* g in the set S are gates of b whose strong value is not justified by ν (i.e., $g \in \mathrm{Unj}(\nu)$), and is not justified either by any of the almost-evaluations at descendant bags of b to which $G_b^{\nu,S}$ is connected. We call *innocent* the other gates of b; they are the gates that are justified in ν (in

particular, those who carry weak values), and the gates that are unjustified in ν but have been justified by an almost-evaluation at a descendant bag b' of b. Crucially, in the latter case, the gate g' justifying the strong value in b' may no longer appear in b, making g unjustified for ν; this is why we remember the set S.

We still have to explain how we connect the gates $G_b^{\nu,S}$ of D to the gates $G_{b_l}^{\nu_l,S_l}$ and $G_{b_r}^{\nu_r,S_r}$ created for the children b_l and b_r of b in T. The first condition is that ν_l and ν_r must *mutually agree*, i.e., $\nu_l(g) = \nu_r(g)$ for all $g \in b_l \cap b_r$, and ν must then be the union of ν_l and ν_r, restricted to b. We impose a second condition to prohibit suspicious gates from escaping before they have been justified, which we formalize as *connectibility* of a pair (ν, S) at bag b to the parent bag of b.

Definition 4.4.4. Let b be a non-root bag, b' its parent bag, and ν a (C, b)-almost-evaluation. For any set $S \subseteq \mathrm{Unj}(\nu)$, we say that (ν, S) is *connectible* to b' if $S \subseteq b'$, i.e., the suspicious gates of ν must still appear in b'. $\lhd$

If a gate $G_b^{\nu,S}$ is such that (ν, S) is not connectible to the parent bag b', then this gate will not be used as input to any other gate (but we do not try to preemptively remove these useless gates in the construction). We are now ready to give the formal definition that will be used to explain how gates are connected:

Definition 4.4.5. Let b be an internal bag with children b_l and b_r, let ν_l and ν_r be respectively (C, b_l) and (C, b_r)-almost-evaluations that mutually agree, and $S_l \subseteq \mathrm{Unj}(\nu_l)$ and $S_r \subseteq \mathrm{Unj}(\nu_r)$ be sets of suspicious gates such that both (ν_l, S_l) and (ν_r, S_r) are connectible to b. The *result* of (ν_l, S_l) and (ν_r, S_r) is then defined as the pair (ν, S) where:

- ν is defined as the restriction of $\nu_l \cup \nu_r$ to b.
- $S \subseteq \mathrm{Unj}(\nu)$ is the new set of suspicious gates, defined as follows. A gate $g \in b$ is innocent (i.e., $g \in b \setminus S$) if it is justified for ν or if it is innocent for some child. Formally, $b \setminus S := (b \setminus \mathrm{Unj}(\nu)) \cup \Big[b \cap [(b_l \setminus S_l) \cup (b_r \setminus S_r)]\Big]$. $\lhd$

We point out that (ν, S) is not necessarily a (C, b)-almost-evaluation.

Construction. We now use these definitions to present the construction formally. For every variable gate g of C, we create a corresponding variable gate $G^{g,1}$ of D, and we create $G^{g,0} := \mathrm{NOT}(G^{g,1})$. For every internal bag b of T, for each (C, b)-almost-evaluation ν and set $S \subseteq \mathrm{Unj}(\nu)$ of suspicious gates of ν, we create one OR-gate $G_b^{\nu,S}$. For every leaf bag b of T, we create one OR-gate $G_b^{\nu,S}$ for every (C, b)-almost-evaluation ν, where we set $S := \mathrm{Unj}(\nu)$; intuitively, in a leaf bag, an unjustified gate is always suspicious (it cannot have been justified at a descendant bag).

Now, for each internal bag b of T with children b_l, b_r, for each pair of gates $G_{b_l}^{\nu_l,S_l}$ and $G_{b_r}^{\nu_r,S_r}$ that are both connectible to b and where ν_l and ν_r mutually agree, letting (ν, S) be the result of (ν_l, S_l) and (ν_r, S_r), if (ν, S) is a (C, b)-almost-evaluation then we create a gate $G_b^{\nu_l,S_l,\nu_r,S_r} = \mathrm{AND}(G_{b_l}^{\nu_l,S_l}, G_{b_r}^{\nu_r,S_r})$ and make it an input of $G_b^{\nu,S}$. We now explain where the variables gates are connected. For every leaf bag b that is responsible for a variable gate x (i.e., b is b_x), for $\nu \in \{\{x \mapsto 1\}, \{x \mapsto 0\}\}$, we set the gate $G^{x,\nu(x)}$ to be the (only) input of the gate $G_b^{\nu,S}$. Last, for every leaf bag b that is not responsible for a variable gate, for every valuation ν of b, we create a

constant 1-gate (i.e., an AND-gate with no inputs), and we make it the (only) input of $G_b^{\nu,S}$. The output gate of D is the gate $G_{b_{\text{root}}}^{\nu,\emptyset}$ where b_{root} is the root of T and ν maps g_{output} to 1 (remember that b_{root} contains only g_{output}).

We now construct a v-tree T' that structures D. T' has the same skeleton than T (in particular, it is a full binary tree). For every node b of T, let us denote by b' the corresponding node of T'. For every leaf bag b of T, b' is either x if b is responsible for the variable x, or unlabelled otherwise. It is then clear that T' structures D. Indeed, the only AND-gates of D are gates of the form $G_b^{\nu_l,S_l,\nu_r,S_r}$, or constant 1-gates (i.e., AND-gates with no inputs). Now, an AND-gate with no input is structured by any node of T', and one can check that any gate of the form $G_b^{\nu_l,S_l,\nu_r,S_r}$ is structured by b'.

4.4.2 Proof of Correctness

We now prove that the circuit D constructed in the main text is indeed a d-SDNNF equivalent to the initial circuit C, and that it can be constructed together with its v-tree in $O(|T| \times f(k))$ for some function $f(k)$ that is in $O(2^{(4+\varepsilon)k})$ for any $\varepsilon \geqslant 0$.

4.4.2.1 D is a Structured DNNF

Negations only apply to the input gates, so D is an NNF. Moreover, we already justified that D is structured by the v-tree T' that we constructed, so D is an SDNNF.

4.4.2.2 D is Equivalent to C

In order to prove that D is equivalent to C, we introduce the standard notion of a *trace* in an NNF:

Definition 4.4.6. Let D be an NNF, χ a valuation of its variable gates, and g a gate that evaluates to 1 under χ. A *trace* of D starting at g according to χ is a set Ξ of gates of D that is minimal by inclusion and such that:

- $g \in \Xi$;
- If $g' \in \Xi$ and g' is an AND gate, then $W(g') \subseteq \Xi$, where $W(g')$ denotes the set of gates that are an input to g';
- If $g' \in \Xi$ and g' is an OR gate, then exactly one input of g' that evaluates to 1 is in Ξ. ◁

The first step is then to prove that traces have exactly one almost-evaluation corresponding to each descendant bag, and that these almost-evaluations mutually agree.

Lemma 4.4.7. *Let χ be a valuation of the variable gates, $G_b^{\nu,S}$ a gate in D that evaluates to 1 under χ and Ξ be a trace of D starting at $G_b^{\nu,S}$ according to χ. Then for any bag $b' \leqslant b$ (meaning that b' is b or a descendant of b), Ξ contains exactly one gate of the form $G_{b'}^{\nu',S'}$. Moreover, over all $b' \leqslant b$, all the almost-evaluations of the gates $G_{b'}^{\nu',S'}$ that are in Ξ mutually agree.*

Proof. The fact that Ξ contains exactly one gate $G_{b'}^{\nu',S'}$ for any bag $b' \leqslant b$ is obvious by construction of D, as OR-gates are assumed to have exactly one input evaluated to 1 in Ξ. For the second claim, suppose by contradiction that not all the almost-evaluations of the gates $G_{b'}^{\nu',S'}$ that are in Ξ mutually agree. We would then have $G_{b_1}^{\nu_1,S_1}$ and $G_{b_2}^{\nu_2,S_2}$ in Ξ and $g \in b_1 \cap b_2$ such that $\nu_1(g) \neq \nu_2(g)$. But because T is a tree decomposition, g appears in all the bags on the path from b_1 and b_2, and by construction the almost-evaluations of the $G_{b'}^{\nu',S'}$ on this path that are in Ξ mutually agree, a contradiction. □

Therefore, Lemma 4.4.7 allows us to define the union of the almost-evaluations in such a trace:

Definition 4.4.8. Let χ be a valuation of the variable gates, G_b^ν a gate in D that evaluates to 1 under χ, and Ξ be a trace of D starting at G_b^ν according to χ. Then $\gamma(\Xi) := \bigcup_{G_{b'}^{\nu',S'} \in \Xi} \nu'$ (the union of the almost-evaluations in Ξ, which is a valuation from $\bigcup_{G_{b'}^{\nu',S'} \in \Xi} b'$ to $\{0,1\}$) is properly defined. ◁

We now need to prove a few lemmas about the behavior of gates that are innocent (i.e., not suspicious).

Lemma 4.4.9. *Let χ be a valuation of the variable gates, $G_b^{\nu,S}$ a gate in D that evaluates to 1 under χ and Ξ be a trace of D starting at $G_b^{\nu,S}$ according to χ. Let $g \in b$ be a gate that is innocent ($g \notin S$). Then the following holds:*

- *If $\nu(g)$ is a weak value of g, then for every input g' of g that is in the domain of $\gamma(\Xi)$ (i.e., g' appears in a bag $b' \leqslant b$), we have that $\gamma(\Xi)$ maps g' to a weak value of g;*
- *If $\nu(g)$ is a strong value of g, then there exists an input g' of g that is in the domain of $\gamma(\Xi)$ such that $\gamma(\Xi)(g')$ is $\nu(g)$ if g is an* AND *or* OR *gate, and $\gamma(\Xi)(g')$ is $1 - \nu(g)$ if g is a* NOT *gate.*

Proof. We prove the claim by bottom-up induction on $b \in T$. One can easily check that the claim is true when b is a leaf bag, remembering that in this case we must have $S = \mathrm{Unj}(\nu)$ by construction (that is, all the gates that are unjustified are suspicious). For the induction case, let b_l, b_r be the children of b. Suppose first that $\nu(g)$ is the weak value of g, and suppose for a contradiction that there is an input g' of g in the domain of $\gamma(\Xi)$ such that $\gamma(\Xi)(g')$ is a strong value of g. By the occurrence and connectedness properties of tree decompositions, there exists a bag $b' \leqslant b$ in which both g and g' occur. Consider the gate $G_{b'}^{\nu',S'}$ that is in Ξ: by Lemma 4.4.7, this gate exists and is unique. By definition of $\gamma(\Xi)$ we have $\nu'(g') = \gamma(\Xi)(g')$. Because ν' is a (C, b')-almost-evaluation that maps g' to a strong value of g, we must have that $\nu'(g)$ is also a strong value of g, thus contradicting our hypothesis that $\nu(g) = \gamma(\Xi)(g) = \nu'(g)$ is a weak value for g.

Suppose now that $\nu(g)$ is a strong value of g. We only treat the case when g is an OR or an AND gate, as the case of a NOT gate is similar. We distinguish two sub-cases:

- g is justified. Then, by definition of ν being a (C, b)-almost-evaluation, there must exist an input g' of g that is also in b such that $\nu(g')$ is a strong value of g, which proves the claim.

- g is unjustified but innocent ($g \notin S$). By construction (precisely, by the second item of Definition 4.4.5), g must then be innocent for a child of b, and the claim clearly holds by induction hypothesis. □

Lemma 4.4.9 allows us to show that for a gate g, letting b be the topmost bag in which g appears (hence, each input of g must occur in some bag $b' \leqslant b$), if g is innocent then for any trace Ξ starting at a gate for bag b, $\gamma(\Xi)$ respects the semantics of g. Formally:

Lemma 4.4.10. *Let χ be a valuation of the variable gates, $G_b^{\nu,S}$ a gate in D that evaluates to 1 under χ and Ξ be a trace of D starting at $G_b^{\nu,S}$ according to χ. Let $g \in b$ be a gate such that b is the topmost bag in which g appears (hence $W(g) \subseteq \mathit{domain}(\gamma(\Xi))$). If g is innocent ($g \notin S$) then $\gamma(\Xi)$ respects the semantics of g, that is, $\gamma(\Xi)(g) = \bigodot \gamma(\Xi)(W(g))$ where $\bigodot$ is the type of g.*

Proof. Clearly implied by Lemma 4.4.9. □

We need one last lemma about the behavior of suspicious gates, which intuitively tells us that if we have already seen all the input gates of a gate g and g is still suspicious, then g can never escape:

Lemma 4.4.11. *Let χ be a valuation of the variable gates, $G_b^{\nu,S}$ a gate in D that evaluates to 1 under χ, and Ξ be a trace of D starting at $G_b^{\nu,S}$ according to χ. Let g be a gate such that the topmost bag b' in which g appears is $\leqslant b$, and consider the unique gate of the form $G_{b'}^{\nu',S'}$ that is in Ξ. If $g \in S'$ then $b' = b$ (hence $G_b^{\nu,S} = G_{b'}^{\nu',S'}$ by uniqueness).*

Proof. Let $g \in S'$. Suppose by contradiction that $b' \neq b$. Let p be the parent of b' (which exists because $b' < b$). It is clear that by construction (ν', S') is connectible to p (recall Definition 4.4.4), hence g must be in p, contradicting the fact that b' should have been the topmost bag in which g occurs. Hence $b' = b$. □

We now have all the results that we need to show that $D \implies C$, i.e., for every valuation χ of the variables of C, if $\chi(D) = 1$ then $\chi(C) = 1$. We prove a stronger result:

Lemma 4.4.12. *Let χ be a valuation of the variable gates, $G_{\mathrm{root}(T)}^{\nu,\emptyset} \in D$ a gate that evaluates to 1 under χ, and Ξ a trace of D starting at $G_{\mathrm{root}(T)}^{\nu,\emptyset}$ according to χ. Then $\gamma(\Xi)$ corresponds to the evaluation χ of C.*

Proof. We prove by induction on C (as its graph is a DAG) that for all $g \in C$, $\gamma(\Xi)(g) = \chi(g)$. When g is a variable gate, consider the leaf bag b_g that is responsible of g, and consider the gate $G_{b_g}^{\nu',S'}$ that is in Ξ: this gate exists and is unique according to Lemma 4.4.7. This gate evaluates to 1 under χ (because it is in the trace), which is only possible if $G^{g,\nu'(g)}$ evaluates to 1 under χ, hence by construction we must have $\nu'(g) = \chi(g)$ and then $\gamma(\Xi)(g) = \chi(g)$. Now suppose that g is an internal gate, and consider the topmost bag b' in which g appears. Consider again the unique $G_{b'}^{\nu',S'}$ that is in Ξ. By induction hypothesis we have that $\gamma(\Xi)(g') = \chi(g')$ for every input g' of g. We now distinguish two cases:

- $b' = \mathrm{root}(T)$. Therefore by Lemma 4.4.10 we know that $\gamma(\Xi)$ respects the semantics of g, which means that $\gamma(\Xi)(g) = \odot\gamma(\Xi)(W(g)) = \odot\chi(W(g)) = \chi(g)$ (where the second equality comes from the induction hypothesis and the third equality is just the definition of the evaluation χ of C), which proves the claim.

- $b' < \mathrm{root}(T)$. But then by Lemma 4.4.11 we must have $g \notin S'$ (because otherwise we should have $b' = \mathrm{root}(T)$ and then $S' = \emptyset$), that is g is innocent for $G_{b'}^{\nu',S'}$. Therefore, again by Lemma 4.4.10, it must be the case that $\gamma(\Xi)$ respects the semantics of g, and we can again show that $\gamma(\Xi)(g) = \chi(g)$, concluding the proof. □

This indeed implies that $D \implies C$: let χ be a valuation of the variable gates and suppose $\chi(D) = 1$. Then by definition of the output of D, it means that the gate $G_{\mathrm{root}(T)}^{\nu,\emptyset}$ such that $\nu(g_{\mathsf{output}}) = 1$ evaluates to 1 under χ. But then, considering a trace Ξ of D starting at $G_{\mathrm{root}(T)}^{\nu,\emptyset}$ according to χ, we have that $\chi(g_{\mathsf{output}}) = \gamma(\Xi)(g_{\mathsf{output}}) = \nu(g_{\mathsf{output}}) = 1$.

To show the converse ($C \implies D$), one can simply observe the following phenomenon:

Lemma 4.4.13. *Let χ be a valuation of the variable gates. Then for every bag $b \in T$, the gate $G_b^{\chi_{|b},S}$ evaluates to 1 under χ, where S is the set of gates $g \in \mathrm{Unj}(\nu)$ such that for every input g' of g that appears in some bag $b' \leqslant b$, then $\chi(g')$ is a weak value of g.*

Proof. Easily proved by bottom-up induction. □

Now suppose $\chi(C) = 1$. By Lemma 4.4.13 we have that $G_{\mathrm{root}(T)}^{\chi_{|\mathrm{root}(T)},\emptyset}$ evaluates to 1 under χ, and because $\chi(g_{\mathsf{output}}) = 1$ we have that $\chi(D) = 1$. This shows that $C \implies D$. Hence, we have proved that D is equivalent to C.

4.4.2.3 D is Deterministic

We now prove that D is deterministic, i.e., that every OR gate in D is deterministic. Recall that the only OR gates in D are the gates of the form $G_b^{\nu,S}$. We will in fact prove that traces starting at OR gates are unique, which clearly implies that all the OR gates are deterministic.

We start by proving the following lemma:

Lemma 4.4.14. *Let χ be a valuation of the variable gates, $G_b^{\nu,S}$ a gate in D that evaluates to 1 under χ and Ξ be a trace of D starting at $G_b^{\nu,S}$ according to χ. Let $g \in b$. Then the following is true:*

- *if g is innocent ($g \notin S$) and $\nu(g)$ is a strong value of g, then there exists an input g' of g such that $\gamma(\Xi)(g)$ is a strong value for g.*

- *if $g \in S$, then for every input g' of g that is in the domain of $\gamma(\Xi)$, we have that $\gamma(\Xi)(g')$ is a weak value for g.*

Proof. We prove the two claims independently:

- Let $g \in b$ such that $g \notin S$ and $\nu(g)$ is a strong value for g. Then the claim directly follows from the second item of Lemma 4.4.9.

- We prove the second claim via a bottom-up induction on T. When b is a leaf then it is trivially true because g has no input g' in b because $|b| \leqslant 1$ because T is friendly. For the induction case, let $G_{b_l}^{\nu_l,S_l}$ and $G_{b_r}^{\nu_r,S_r}$ be the (unique) gates in Ξ corresponding to the children b_l, b_r of b. By hypothesis we have $g \in S$. By definition of a gate being suspicious, we know that $\nu(g)$ is a strong value for g. To reach a contradiction, assume that there is an input g' of g in the domain of $\gamma(\Xi)$ such that $\gamma(\Xi)(g')$ is a strong value for g. Clearly this g' is not in b, because g is unjustified by ν (because $S \subseteq \mathrm{Unj}(\nu)$). Either g' occurs in a bag $b'_l \leqslant b_l$, or it occurs in a bag $b'_r \leqslant b_r$. The two cases are symmetric, so we assume that we are in the former. As $g \in b$ and $g' \in b'_l$, by the properties of tree decompositions and because $g' \notin b$, we must have $g \in b_l$. Hence, by the contrapositive of the induction hypothesis on b_l applied to g, we deduce that $g \notin S_l$. But then by the second item of Definition 4.4.5, g should be innocent for $G_b^{\nu,S}$, that is $g \notin S$, which is a contradiction. □

We are ready to prove that traces starting at OR gates are unique. Let us first introduce some useful notations: Let U, U' be sets of gates, ν, ν' be valuations having the same domain. We write $(\nu, U) = (\nu', U')$ to mean $\nu = \nu'$ and $U = U'$, and for g in the domain of ν we write $(\nu, U)(g) = (\nu', U')(g)$ to mean that $\nu(g) = \nu'(g)$ and that we have $g \in U$ iff $g \in U'$. We show the following:

Lemma 4.4.15. *Let χ be a valuation of the variable gates such that $G_b^{\nu,S}$ evaluates to 1 under χ. Then there is a unique trace of D starting at $G_b^{\nu,S}$ according to χ.*

Proof. We will prove the claim by bottom-up induction on T. The case when b is a leaf is trivial because gates of the form $G_b^{\nu,S}$, for b a leaf of T, have either:

- exactly one input g' that is a constant 1-gate;

- or, exactly one input that is a variable gate $G^{x,1}$, if b is responsible of x and $\nu(g) = 1$;

- or, exactly one input $G^{x,0}$ that is the NOT-gate $\mathrm{NOT}(G^{x,1})$, if b is responsible of x and $\nu(g) = 0$.

In all cases, there can be at most one trace. For the inductive case, let b be an internal bag with children b_l and b_r. By induction hypothesis for every $G_{b_l}^{\nu_l,S_l}$ (resp., $G_{b_r}^{\nu_r,S_r}$) that evaluates to 1 under χ there exists a unique trace Ξ_l (resp., Ξ_r) of D starting at $G_{b_l}^{\nu_l,S_l}$ (resp., $G_{b_r}^{\nu_r,S_r}$). Hence, if by contradiction there are more than two traces of D starting at $G_b^{\nu,S}$, it can only be because $G_b^{\nu,S}$ is not deterministic, i.e., because at least two different inputs of $G_b^{\nu,S}$ evaluate to 1 under χ, say $G_b^{\nu_l,S_l,\nu_r,S_r}$ and $G_b^{\nu'_l,S'_l,\nu'_r,S'_r}$ with $(\nu_l, S_l) \neq (\nu'_l, S'_l)$ or $(\nu_r, S_r) \neq (\nu'_r, S'_r)$. We can suppose that it is $(\nu_l, S_l) \neq (\nu'_l, S'_l)$, since the other case is symmetric. Hence there exists $g_0 \in b_l$ such that $(\nu_l, S_l)(g_0) \neq (\nu'_l, S'_l)(g_0)$. Let Ξ_l be the trace of D starting at $G_{b_l}^{\nu_l,S_l}$ and Ξ'_l be the trace of D starting at $G_{b_l}^{\nu'_l,S'_l}$. We observe the following simple fact about Ξ_l and Ξ'_l:

(*) for any g, if $\gamma(\Xi_l)(g) \neq \gamma(\Xi'_l)(g)$ then $g \notin b$.

Indeed otherwise we should have $\nu(g) = \nu_l(g) = \gamma(\Xi_l)(g)$ and $\nu(g) = \nu'_l(g) = \gamma(\Xi'_l)(g)$, which is impossible.

Now we will define an operator θ that takes as input a gate g such that $(\gamma(\Xi_l), S_l)(g) \neq (\gamma(\Xi'_l), S'_l)(g)$, and outputs another gate $\theta(g)$ which is an input of g and such that again $(\gamma(\Xi_l), S_l)(\theta(g)) \neq (\gamma(\Xi'_l), S'_l)(\theta(g))$. This will lead to a contradiction because for any $n \in \mathbb{N}$, starting with g_0 and applying θ n times consecutively we would obtain a path of n mutually distinct gates (because C is acyclic), but C has a finite number of gates.

Let us now define θ: let g such that $(\gamma(\Xi_l), S_l)(g) \neq (\gamma(\Xi'_l), S'_l)(g)$. We distinguish two cases:

- We have $(\gamma(\Xi_l), S_l)(g) \neq (\gamma(\Xi'_l), S'_l)(g)$ because $\gamma(\Xi_l)(g) \neq \gamma(\Xi'_l)(g)$. Then by (*), we know for sure that $g \notin b$. Therefore the topmost bag b' in which g occurs is $\leqslant b_l$. Let $G_{b'}^{\nu',S'}$ be the gate in Ξ_l and $G_{b'}^{\nu'',S''}$ the gate in Ξ'_l (they exist and are unique by Lemma 4.4.7). Then by Lemma 4.4.11 we must have $g \notin S'$ and $g \notin S''$, because otherwise we should have $b' = b$, which is not true. Hence, by Lemma 4.4.10 we know that both $\gamma(\Xi_l)$ and $\gamma(\Xi'_l)$ respect the semantics of g. But we have $\gamma(\Xi_l)(g) \neq \gamma(\Xi'_l)(g)$, so there must exist an input g' of g such that $\gamma(\Xi_l)(g') \neq \gamma(\Xi'_l)(g')$. We can thus take $\theta(g)$ to be g'.

- We have $(\gamma(\Xi_l), S_l)(g) \neq (\gamma(\Xi'_l), S'_l)(g)$ because (without loss of generality) $g \notin S_l$ and $g \in S'_l$. Observe that this implies that $g \in b_l$, and that $\nu'_l(g)$ is a strong value for g. We can assume that $\nu_l(g) = \nu'_l(g)$, as otherwise we would have $\gamma(\Xi_l)(g) \neq \gamma(\Xi'_l)(g)$, which is a case already covered by the last item. Hence $\nu_l(g)$ is also a strong value for g, but we have $g \notin S_l$, so by the first item of Lemma 4.4.14 we know that there exists an input g' of g that occurs in some bag $\leqslant b_l$ and such that $\gamma(\Xi_l)(g')$ is a strong value for g. We show that $\gamma(\Xi'_l)(g')$ must in contrast be a weak value for g, so that we can take $\theta(g)$ to be g' and conclude the proof. Indeed suppose by way of contradiction that $\gamma(\Xi'_l)(g')$ is a strong value for g. By the contrapositive of the second item of Lemma 4.4.14, we get that $g \notin S'_l$, which contradicts our assumption.

Hence we have constructed θ, which shows a contradiction, which means that in fact we must have $G_b^{\nu_l,S_l,\nu_r,S_r} = G_b^{\nu'_l,S'_l,\nu'_r,S'_r}$, so that $G_b^{\nu,S}$ is deterministic, which proves that there is a unique trace of D starting at $G_b^{\nu,S}$ according to χ, which was our goal. □

This concludes the proof that D is deterministic, and thus that D is a d-SDNNF equivalent to C.

4.4.2.4 Analysis of the Running Time

We last check that the construction can be performed in time $O(|T| \times f(k))$, for some function $f(k)$ that is in $O(2^{(4+\varepsilon)k})$ for any $\varepsilon > 0$:

- From the initial tree decomposition T of C, in time $O(k|T|)$ we computed the g_{output}-friendly tree decomposition T_{friendly} of size $O(k|T|)$;

- In linear time in T_{friendly}, for every varable $x \in C_{\mathsf{var}}$ we selected a leaf bag b_x of T such that $\lambda(b_x) = \{x\}$;

- We can clearly compute the v-tree T' in linear time in T_{friendly};
- For each bag b of T_{friendly} we have $2^{2|b|} \leqslant 2^{2k+2}$ different pairs of a valuation ν of b and of a subset S of b, and checking if ν is a (C, b)-almost-evaluation and if S is a subset of the unjustified gates of ν can be done in polynomial time in $|b| \leqslant k+1$ (we access the inputs and the type of each gate in constant time from C), hence we pay $O(|T_{\text{friendly}}| \times p(k) \times 2^{2k})$ to create the gates of the form $G_b^{\nu,S}$, for some polynomial p;
- We constructed and connected in time $O(|T_{\text{friendly}}| \times p'(k) \times 2^{4k})$ the gates of the form $G_b^{\nu_l,S_l,\nu_r,S_r}$ (the polynomial is for testing if the result of (ν_l, S_l) and of (ν_r, S_r) is an almost-evaluation);
- In time $O(|T_{\text{friendly}}|)$ we connected the gates of the form $G_b^{\nu,S}$ for b a leaf to their inputs.

Hence, the total time is indeed in $O(|T| \times f(k))$, for some function $f(k)$ that is in $O(2^{(4+\varepsilon)k})$ for any $\varepsilon > 0$.

4.5 Application to PQE of CFG-Datalog

In this section we show how to use the algorithm of Theorem 4.3.2 to evaluate CFG-Datalog queries on treelike TID instances. We recall our main provenance result from Chapter 3, which states that we can compute in FPT-bilinear time (parameterized by k_{P} and k_{I}) a stratified cycluit capturing the provenance of a CFG-Datalog program of body size $\leqslant k_{\text{P}}$ on a relational instance of treewidth $\leqslant k_{\text{I}}$:

Theorem 3.6.1. *Given a CFG-Datalog program P of body size k_{P} and a relational instance I of treewidth k_{I}, we can construct in FPT-bilinear time in $|I| \cdot |P|$ (parameterized by k_{P} and k_{I}) a representation of the provenance of P on I as a stratified cycluit.*

Moreover, we can prove that the treewidth of C is linear in the size of the CFG-Datalog program:

Proposition 4.5.1. *The treewidth of the cycluit C constructed in Theorem 3.6.1 is in $O(|P|)$.*

Proof. We recall how C was constructed. Imagine first that P is a CFG$^{\text{GN}}$-Datalog program. From the CFG$^{\text{GN}}$-Datalog program P of body size $\leqslant k_{\text{P}}$ and the treewidth bound k_{I}, we used Theorem 3.5.4 to compute in FPT-linear time in $|P|$ a Γ_σ^k-SATWA A that tests it on tree encodings of width $\leqslant k_{\text{I}}$. We also computed in FPT-linear time a tree encoding E of the instance I, i.e., a Γ_σ^k-tree E. We then lift the Γ_σ^k-SATWA A in linear time using Lemma 3.6.14 to a $\overline{\Gamma_\sigma^k}$-SATWA A', and used Theorem 3.6.11 on A' and E to compute in FPT-bilinear time the stratified cycluit C that captures the provenance of A' on E. Hence, to show that the treewidth of C is linear in $|P|$, it suffices to prove that the treewidth of the cycluit constructed in Theorem 3.6.11 is linear in the size of the SATWA A'. We now refer to the construction of Theorem 3.6.11 and justify that the treewidth of C is indeed linear in the size of the automaton. For every node w of E, we created $O(|A'|)$ gates,

and those gates can only be connected between them or between gates created for the neighbors of w. Hence, the treewidth of C is $O(|A'|)$: we can compute a tree decomposition of C where the underlying tree is the same as that of E and where each bag b corresponding to a node w of E contains the gates defined for the node w and for the neighbors of w. We can then observe that this argument is still true for CFG-Datalog programs and the SATWAs with directions that we used in Section 3.7.2. □

Unfortunately, we cannot directly apply Theorem 4.3.2 because here we have a cycluit, while our transformation takes a circuit as input. Hence, the first step will be to convert this stratified cycluit into a circuit. A first result from existing work is that we can remove cycles in cycluits and convert them to circuits, with a quadratic blowup, by creating linearly many copies to materialize the fixpoint computation:

Proposition 4.5.2 ([Riedel and Bruck 2012], Theorem 2)**.** *For any stratified cycluit C, we can compute in time $O(|C|^2)$ a Boolean circuit C' which is equivalent to C.*

However this approach has two disadvantages for us: first, it is quadratic rather than linear, which would imply a complexity at least quadratic in the database I if we used it. Second, the bound of Proposition 4.5.1 on the treewidth of the cycluit is not preserved on the resulting circuit, while we need such a bound to use Theorem 4.3.2. Therefore, we use another cycle removal result, that proceeds in FPT-linear time when parameterized by the treewidth of the input cycluit:

Theorem 4.5.3 ([Amarilli, Bourhis, Monet, and Senellart 2017, Theorem 34])**.** *There is an $\alpha \in \mathbb{N}$ such that for any stratified cycluit C of treewidth k, we can compute in time $O(2^{k^\alpha}\,|C|)$ a circuit C' which is equivalent to C and has treewidth $O(2^{k^\alpha})$.*

We can then use Theorem 4.5.3 to apply our main upper bound of this chapter (Theorem 4.3.2) on the cycluit constructed in Theorem 3.6.1 (relying on the treewidth bound of Proposition 4.5.1) and show the following:

Theorem 4.5.4. *Fix $k_{\mathrm{I}}, k_{\mathrm{P}} \in \mathbb{N}$ and consider them as constants. There is an $\alpha \in \mathbb{N}$ s.t., given a CFG-Datalog program P of body size k_{P} and a TID instance (I, π) of treewidth k_{I}, we can compute a d-SDNNF representing the provenance of P on I in time $O(2^{2^{|P|^\alpha}}\,|I|)$.*

Proof. Let C be the stratified cycluit constructed from Theorem 3.6.1, whose treewidth is in $O(|P|)$ according to Proposition 4.5.1. Let α'' be the constant from Theorem 4.5.3. We use Theorem 4.5.3 to compute in time $O(2^{|P|^{\alpha''}}\,|C|)$ a circuit C' which is equivalent to C and has treewidth $O(2^{|P|^{\alpha''}})$. We then use Theorem 4.2.1 with $\varepsilon = 1$ to compute a tree decomposition T of C' in time $O(2^{|P|^{\alpha''}}\,|C| \times 2^{33\times 2^{3\times|P|^{\alpha''}}})$ (of width $O(2^{|P|^{\alpha''}})$). If we let $\alpha' := \alpha''+1$, we can do all of this in time $O(2^{2^{|P|^{\alpha'}}} \times |C|)$. Now, we use Theorem 4.3.2 with $\varepsilon = 1$ to compute in time $O(2^{2^{|P|^{\alpha'}}} \times |C| \times 2^{5\times 2^{|P|^{\alpha''}}})$ a d-SDNNF equivalent to C'. Since $|C|$ is of size $O(|I| \times |P|)$, we can do all of this in time $O(2^{2^{|P|^\alpha}} \times |I|)$ for $\alpha := \alpha' + 1$. □

This d-SDNNF then allows us to solve PQE for this program on this instance with the same complexity, since the Boolean circuit probability computation problem restricted to d-DNNFs is in linear time [Darwiche 2001].

4.6 Lower Bounds on OBDDs

We now move to lower bounds on the size of structured representations of Boolean functions, in terms of the width of a circuit for that function. Our end goal is to obtain a lower bound for (d-)SDNNFs, that will form a counterpart to the upper bound of Theorem 4.3.2. We will do so in Section 4.7. For now, in this section, we consider a weaker class of lineage representations than (d-)SDNNFs, namely, *OBDDs* (recall their definition from Section 1.7). Our upper bound in Section 4.3 applied to arbitrary Boolean circuits; however, our lower bounds in this section and the next one will already apply to much weaker formalisms for Boolean functions, namely, monotone DNFs and monotone CNFs. Some lower bounds are already known for the compilation of monotone CNFs into OBDDs: Bova and Slivovsky have constructed a family of CNFs of bounded degree whose OBDD width is exponential in their number of variable occurrences [Bova and Slivovsky 2017, Theorem 19], following an earlier result of this type by Razgon, presented in [Razgon 2014, Corollary 1]. The result is as follows:

Theorem 4.6.1 ([Bova and Slivovsky 2017, Theorem 19])**.** *There is a class of monotone CNF formulas of bounded degree and arity such that every formula φ in this class has OBDD size at least $2^{\Omega(|\varphi|)}$.*

We adapt some of these techniques to show a more general result: our lower bound applies to *any* monotone DNF or monotone CNF, not to one specific family. Specifically, we show:

Theorem 4.6.2. *Let φ be a monotone DNF (or monotone CNF), let $a := \mathrm{arity}(\varphi)$ and $d := \mathrm{degree}(\varphi)$. Then any OBDD for φ has width $\geqslant 2^{\left\lfloor \frac{\mathrm{pw}(\varphi)}{a^3 \times d^2} \right\rfloor}$.*

From our Theorem 4.6.2, we can easily derive Theorem 4.6.1 using the fact (also used in the proof of [Bova and Slivovsky 2017, Theorem 19]) that there exists a family of monotone CNFs of bounded degree and arity whose treewidth (hence pathwidth) is linear in their size, namely, the CNFs built from *expander graphs* (see [Grohe and Marx 2009, Theorem 5 and Proposition 1]). Note that expander graphs can also be used to show lower bounds for *(non-deterministic and non-structured) DNNFs* for a CNF formula [Bova, Capelli, Mengel, and Slivovsky 2015]; our lower bound on SDNNFs of Section 4.7 will not capture this result (because we need structuredness).

We observe that, for a family of formulas with bounded arity and degree, the bound of Theorem 4.6.2 is optimal, up to constant factors in the exponent. Indeed, following earlier work [Ferrara, Pan, and Vardi 2005; Razgon 2014], Bova and Slivovsky have shown that any CNF φ can be compiled to OBDDs of width $2^{\mathrm{pw}(\varphi)+2}$ [Bova and Slivovsky 2017, Theorem 4 and Lemma 9]. (Their upper bound result also applies to DNFs, and does not assume monotonicity nor a bound on the arity or degree.) In other words, for any monotone DNF or monotone CNF of bounded arity and degree, pathwidth *characterizes* the width of an OBDD for the formula, in the following sense:

Corollary 4.6.3. *For any constant c, for any monotone DNF (or monotone CNF) φ with arity and degree bounded by c, the smallest width of an OBDD for φ is $2^{\Theta(\mathrm{pw}(\varphi))}$.*

This corollary talks about the pathwidth of φ measured as that of its hypergraph, but note that the same result would hold when measuring the pathwidth of the

incidence graph of φ (i.e., the undirected graph that has as set of vertices the variables and the clauses of φ, and that connects a variable and a clause iff this variable appears in that clause) or dual hypergraph of φ (i.e., the hypergraph having as nodes the clauses of φ, and having, for every variable v of φ, one hyperedge containing all the clauses in which v appears). Indeed, all these pathwidths are within a constant factor of one another when the degree and arity are bounded by a constant.

We prove Theorem 4.6.2 in the rest of this section. We present the proof in the case of monotone DNFs to reuse existing lower bound techniques from [Amarilli, Bourhis, and Senellart 2016; Amarilli 2016], but explain at the end of this section how the proof adapts to monotone CNFs. We first present *pathsplitwidth*, a new notion which intuitively measures the performance of a variable ordering for an OBDD on the monotone DNF φ, and connect it to the pathwidth of φ. Second, we recall the definition of *dncpi-sets* introduced in [Amarilli, Bourhis, and Senellart 2016; Amarilli 2016] to show lower bounds from the structure of Boolean functions. Last, we conclude the proof by connecting pathsplitwidth to the size of dncpi-sets.

Pathsplitwidth. The first step of the proof is to rephrase the bound on pathwidth, arity, and degree, in terms of a bound on the performance of variable orderings. Intuitively, a good variable ordering is one which does not *split* too many clauses. Formally:

Definition 4.6.4. Let $H = (V, E)$ be a hypergraph, and $\mathbf{v} = v_1, \ldots, v_{|V|}$ be an ordering on the variables of V. For $1 \leqslant i \leqslant |V|$, we define $\mathrm{Split}_i(\mathbf{v}, H)$ as the set of hyperedges e of H that contain both a variable at or before v_i, and a variable strictly after v_i, formally: $\mathrm{Split}_i(\mathbf{v}, H) := \{e \in E \mid \exists l \in \{1, \ldots, i\}$ and $\exists r \in \{i+1, \ldots, |V|\}$ such that $\{v_l, v_r\} \subseteq e\}$. Note that $\mathrm{Split}_{|V|}(\mathbf{v}, H)$ is always empty.

The *pathsplitwidth* of $\mathbf{v}$ relative to H is the maximum size of the split, formally, $\mathrm{psw}(\mathbf{v}, H) := \max_{1 \leqslant i \leqslant |V|} |\mathrm{Split}_i(\mathbf{v}, H)|$. The *pathsplitwidth* $\mathrm{psw}(H)$ of H is then the minimum of $\mathrm{psw}(\mathbf{v}, H)$ over all variable orderings $\mathbf{v}$ of V. ◁

In other words, $\mathrm{psw}(H)$ is the smallest integer $n \in \mathbb{N}$ such that, for any variable ordering $\mathbf{v}$ of the nodes of H, there is a moment at which n hyperedges of H are *split*, i.e., for n hyperedges e, we have begun enumerating the nodes of e but we have not enumerated all of them yet. We note that the pathsplitwidth of H is exactly the *linear branch-width* [Nordstrand 2017] of the dual hypergraph of H, but we introduced pathsplitwidth because it fits our proofs better.

For a monotone DNF φ with hypergraph H, the quantity $\mathrm{psw}(H)$ is intuitively connected to the quantity of information that an OBDD will have to remember when evaluating φ following any variable ordering, which we will formalize via dncpi-sets. This being said, the definition of pathsplitwidth is also reminiscent of that of pathwidth, and we can indeed connect the two (up to a factor of the arity):

Lemma 4.6.5. *For any hypergraph $H = (V, E)$, we have* $\mathrm{pw}(H) \leqslant \mathrm{arity}(H) \times \mathrm{psw}(H)$.

Proof. Let $H = (V, E)$ be a hypergraph, and let $\mathbf{v}$ be an enumeration of the nodes of H witnessing that H has pathsplitwidth $\mathrm{psw}(H)$. We will construct a path decomposition of H of width $\leqslant \mathrm{arity}(H) \times \mathrm{psw}(H)$. Consider the path $P = b_1, \cdots, b_{|V|}$ and the labeling function λ where $\lambda(b_i) := \{v_i\} \cup \bigcup \mathrm{Split}_i(\mathbf{v}, H)$ for

$1 \leqslant i \leqslant |V|$. Let us show that (P, λ) is a path decomposition of H: once this is established, it is clear that its width will be $\leqslant \text{arity}(H) \times \text{psw}(H)$.

First, we verify the occurrence condition. Let $e \in E$. If e is a singleton $\{v_i\}$ then e is included in b_i. Now, if $|e| \geqslant 2$, then let v_i be the first element of e enumerated by $\mathbf{v}$. We have $e \in \text{Split}_i(\mathbf{v}, H)$, and therefore e is included in b_i.

Second, we verify the connectedness condition. Let v be a vertex of H, then by definition $v \in b_i$ iff $v = v_i$ or there exists $e \in \text{Split}_i(\mathbf{v}, H)$ with $v \in e$. We must show that the set T_v of the bags that contain v forms a connected subpath in P. To show this, first observe that for every $e \in E$, letting $\text{Split}(e) = \{v_i \mid 1 \leqslant i < |V| \wedge e \in \text{Split}_i(\mathbf{v}, H)\}$, then $\text{Split}(e)$ is clearly a connected segment of $\mathbf{v}$. Second, note that for every e with $v \in e$, then either $v \in \text{Split}(e)$ or v and the connected subpath $\text{Split}(e)$ are adjacent (in the case where v is the last vertex of e in the enumeration). Now, by definition T_v is the union of the $b_{v'}$ for $v' \in \text{Split}(e)$ with $v \in e$ and of b_i, so it is a union of connected subpaths which all contain b_i or are adjacent to it: this establishes that T_v is a connected subpath, which shows in turn that (T, λ) is a path decomposition, concluding the proof. □

For completeness with the preceding result, we note that the following also holds, although we do not use it in the proof of Theorem 4.6.2:

Lemma 4.6.6. *For any hypergraph $H = (V, E)$, we have* $\text{psw}(H) \leqslant \text{degree}(H) \times (\text{pw}(H) + 1)$.

Proof. Let $P = b_1 - \cdots - b_m$ be a path decomposition of H of width $\text{pw}(H)$. For $1 \leqslant i \leqslant m$ we define *first*(b_i) to be the set of all $v \in b_i$ such that b_i is the first bag containing v (this set can be empty). Let $\mathbf{v}_i$ be any ordering on *first*(b_i). Consider the ordering $\mathbf{v} := \mathbf{v}_1 \dots \mathbf{v}_m \mathbf{v}'$, where $\mathbf{v}'$ is any ordering of the remaining vertices of H (i.e., those that do not appear in P because they are not present in any hyperedge). Let $n = |\mathbf{v}| = |H|$. We claim that for any $1 \leqslant i \leqslant n$, we have $|\text{Split}_i(\mathbf{v}, H)| \leqslant \text{degree}(H) \times (\text{pw}(H) + 1)$, which clearly implies that $\text{psw}(H) \leqslant \text{degree}(H) \times (\text{pw}(H) + 1)$. This is clear for v_i in $\mathbf{v}'$, since then $|\text{Split}_i(\mathbf{v}, H)| = 0$. Now suppose $v_i \in \mathbf{v}_j$ for some $1 \leqslant j \leqslant m$. Let $e \in \text{Split}_i(\mathbf{v}, H)$. We will show that there exists $v' \in b_j$ such that $v' \in e$, which will imply that $|\text{Split}_i(\mathbf{v}, H)| \leqslant \text{degree}(H) \times (\text{pw}(H) + 1)$. Assume by way of contradiction that there is no such v'. We know that $e \in \text{Split}_i(\mathbf{v}, H)$, hence there exist v^-, v^+ with $\{v^-, v^+\} \subseteq e$ and $v^- \in \mathbf{v}_{j^-}$ for some $j^- < j$ and $v^+ \in \mathbf{v}_{j^+}$ for some $j^+ > j$. But, as P is a path decomposition of H and $\{v^-, v^+\} \subseteq e$, we know that v^- and v^+ must appear together in a bag. Now, as b_{j^+} is the first bag in which v^+ appears, it must be the case that $v^- \in b_{j^+}$, and therefore $v^- \in b_j$ (otherwise the connectedness property would be violated), which leads to a contradiction and concludes the proof. □

Thanks to Lemma 4.6.5, to show Theorem 4.6.2 it suffices to show that an OBDD for φ has width $\geqslant 2^{\left\lfloor \frac{\text{psw}(\varphi)}{(a \times d)^2} \right\rfloor}$, which we will do in the rest of this section.

dncpi-sets. To show this lower bound, we use the technical tool of *dncpi-sets* [Amarilli, Bourhis, and Senellart 2016; Amarilli 2016]. We recall the definitions here, adapting the notation slightly. Remember that our monotone DNFs are assumed to be minimized. Note that dncpi-sets are reminiscent of *subfunction width* in [Bova and Slivovsky 2017] (see Theorem 17 in [Bova and Slivovsky 2017]), but the latter notion is only defined for graph CNFs.

Definition 4.6.7 ([Amarilli 2016, Definition 6.4.6])**.** Given a monotone DNF φ on variables V, a *disjoint non-covering prime implicant set* (dncpi-set) of φ is a set S of clauses of φ which:

- are pairwise disjoint: for any $D_1 \neq D_2$ in S, we have $D_1 \cap D_2 = \emptyset$.
- are *non-covering* in the following sense: for any clause D of φ, if $D \subseteq \bigcup S$, then $D \in S$.

The *size* of S is the number of clauses that it contains.

Given a variable ordering $\mathbf{v}$ of V, we say that $\mathbf{v}$ *shatters* a dncpi-set S if there exists $1 \leqslant i \leqslant |V|$ such that $S \subseteq \mathrm{Split}_i(\mathbf{v}, H)$, where H is the hypergraph of φ. $\lhd$

Observe the analogy between shattering and splitting, which we will substantiate below. We recall the main result on dncpi-sets:

Lemma 4.6.8 ([Amarilli 2016, Lemma 6.4.7])**.** *Let φ be a monotone DNF on variables V and $n \in \mathbb{N}$. Assume that, for every variable ordering $\mathbf{v}$ of V, there is some dncpi-set S of φ with $|S| \geqslant n$, such that $\mathbf{v}$ shatters S. Then any OBDD for φ has width $\geqslant 2^n$.*

Proof sketch. Considering the point at which the dncpi-set is shattered, the OBDD must remember exactly the status of each clause of the set: any valuation that satisfies a subset of these clauses gives rise to a different continuation function. This is where we use the fact that the DNF is monotone: it ensures that we can freely choose a valuation of the variables that do not occur in the dncpi-set without making the formula true. □

Concluding the proof. We conclude the proof of Theorem 4.6.2 by showing that any variable ordering of the variables of a monotone DNF φ shatters a dncpi-set of the right size. The formal statement is as follows:

Lemma 4.6.9. *Let φ be a monotone DNF, H its hypergraph, and $\mathbf{v}$ an enumeration of its variables. Then there is a dncpi-set S of φ shattered by $\mathbf{v}$ such that $|S| \geqslant \left\lfloor \frac{\mathrm{psw}(H)}{(\mathrm{arity}(H) \times \mathrm{degree}(H))^2} \right\rfloor$.*

We prove this result in the rest of the section. Our goal is to construct a dncpi-set, which intuitively consists of clauses that are disjoint and which do not cover another clause. We can do so by picking clauses sufficiently "far apart". Let the *exclusion graph* of $H = (V, E)$ be the graph on E where two edges $e \neq e'$ are adjacent if there is an edge e'' of E with which they both share a node: this is in particular the case when e and e' intersect as we can take $e'' := e$. Formally, the exclusion graph is $G_H = (E, \{\{e, e'\} \in E^2 \mid e \neq e' \wedge \exists e'' \in E, (e \cap e'') \neq \emptyset \wedge (e' \cap e'') \neq \emptyset\})$. In other words, two hyperedges are adjacent in G_H iff they are different and are at distance at most 4 in the incidence graph of H.

Remember that an *independent set* in the graph G_H is a subset S of E such that no two elements of S are adjacent in G_H. The definition of G_H then ensures:

Lemma 4.6.10. *For any monotone DNF φ, letting H be its hypergraph, any independent set of the exclusion graph G_H is a dncpi-set of φ.*

Proof. The vertices of G_H are clauses of φ by construction. Now, the elements of an independent set S are pairwise disjoint clauses, because whenever two clauses e and e' intersect, then taking $e'' := e$, we have that e'' intersects both e and e', so there is an edge between e and e' in the exclusion graph, so e and e' cannot both occur in an independent set. Now, to show why S is non-covering, assume by contradiction that there exists a clause e'' of φ which is not in S and such that $e'' \subseteq \bigsqcup S$. Remember that φ has been minimized, so e'' cannot be a strict subset of a single clause of S, and it cannot be a clause of S by hypothesis. Hence, there must be two clauses $e \neq e'$ in S such that e'' intersects both e and e'. Thus, e'' witnesses that there is an edge between e and e' in the exclusion graph, so they cannot be both part of S, a contradiction. This concludes the proof. □

In other words, our goal is to compute a large independent set of the exclusion graph. To do this, we will use the following straightforward lemma about independent sets:

Lemma 4.6.11. *Let $G = (V, E)$ be a graph and let $V' \subseteq V$. Then G has an independent set $S \subseteq V'$ of size at least $\left\lfloor \frac{|V'|}{\text{degree}(G)+1} \right\rfloor$.*

Proof. We construct the independent S set with the following trivial algorithm: start with $S := \emptyset$ and, while V' is non-empty, pick an arbitrary vertex v in V', add it to S, and remove v and all its neighbors from G. It is clear that this algorithm terminates and add the prescribed number of vertices to S, so all that remains is to show that S is an independent set at the end of the algorithm. This is initially true for $S = \emptyset$; let us show that it is preserved throughout the algorithm. Assume by way of contradiction that, at a stage of the algorithm, we add a vertex v to S and that it stops being an independent set. This means that S contains a neighbor v' of v which must have been added earlier; but when we added v' to S we have removed all its neighbors from G, so we have removed v and we cannot add it later, a contradiction. Hence, the algorithm is correct and the claim is shown. □

Moreover, we can bound the degree of G_H using the degree and arity of H:

Lemma 4.6.12. *Let H be a hypergraph. Then we have* $\text{degree}(G_H) \leqslant (\text{arity}(H) \times \text{degree}(H))^2 - 1$.

Proof. Any edge e of H contains $\leqslant \text{arity}(H)$ vertices, each of which occurs in $\leqslant \text{degree}(H) - 1$ edges that are different from e, so any edge e of H intersects at most $n := \text{arity}(H) \times (\text{degree}(H) - 1)$ edges different from e. Hence, the degree of G_H is at most $n + n^2$ (counting the edges that intersect e or those at distance 2 from e). Now, we have $n + n^2 = n(n + 1)$, and as $\text{degree}(H) \geqslant 1$ and $\text{arity}(H) \geqslant 1$ (because we assume that hypergraphs contain at least one non-empty edge), the degree of G_H is $< \text{arity}(H) \times \text{degree}(H) \times (1 + \text{arity}(H) \times (\text{degree}(H) - 1))$, i.e., it is indeed $< (\text{arity}(H) \times \text{degree}(H))^2$, which concludes. □

We are now ready to conclude the proof of Lemma 4.6.9:

Proof of Lemma 4.6.9. Let φ be a monotone DNF, $H = (V, E)$ its hypergraph, and $\mathbf{v}$ an enumeration of its variables. By definition of pathsplitwidth, there is $v_i \in V$ such that, for $E' := \text{Split}_i(\mathbf{v}, H)$, we have $|E'| \geqslant \text{psw}(H)$. Now, by Lemma 4.6.11, G_H has an independent set $S \subseteq E'$ of size at least $\left\lfloor \frac{|E'|}{\text{degree}(G_H)+1} \right\rfloor$ which

is $\geqslant \left\lfloor \frac{\mathrm{psw}(H)}{(\mathrm{arity}(H)\times\mathrm{degree}(H))^2} \right\rfloor$ by Lemma 4.6.12. Hence, S is a dncpi-set by Lemma 4.6.10, has the desired size, and is shattered since $S \subseteq E'$. □

Combining this result with Lemma 4.6.5 and Lemma 4.6.8 concludes the proof of Theorem 4.6.2.

From DNFs to CNFs. We now argue that Theorem 4.6.2 also holds for monotone CNFs. Let φ be a monotone CNF, $a := \mathrm{arity}(\varphi)$ and $d := \mathrm{degree}(\varphi)$, and suppose for a contradiction that there is an OBDD O for φ of width $< 2^{\left\lfloor \frac{\mathrm{pw}(\varphi)}{a^3\times d^2} \right\rfloor}$. Consider the monotone DNF φ' built from φ by replacing each $\wedge$ by a $\vee$ and each $\vee$ by a $\wedge$. Now, let O' be the OBDD built from O by replacing the label $b \in \{0,1\}$ of each edge by $1-b$, and replacing the label b of each leaf by $1-b$. It is clear, by De Morgan's laws, that O' is an OBDD for φ' of size $< 2^{\left\lfloor \frac{\mathrm{pw}(\varphi)}{a^3\times d^2} \right\rfloor}$, which contradicts Theorem 4.6.2 applied to monotone DNFs.

4.7 Lower Bounds on (d-)SDNNFs

In the previous section, we have shown that *pathwidth* measures how concisely an OBDD can represent a monotone DNF or CNF formula with bounded degree and arity. In this section, we move from OBDDs to (d-)SDNNFs, and show that *treewidth* plays a similar role to pathwidth in this setting. Formally, we show the following analogue of Theorem 4.6.2:

Theorem 4.7.1. *Let φ be a monotone DNF (resp., monotone CNF), let $a := \mathrm{arity}(\varphi)$ and $d := \mathrm{degree}(\varphi)$. Then any d-SDNNF (resp., SDNNF) for φ has size $\geqslant 2^{\left\lfloor \frac{\mathrm{tw}(\varphi)}{3\times a^3\times d^2} \right\rfloor} - 1$.*

Combined with Theorem 4.3.2 (or with existing results specific to CNF formulas such as [Bova and Slivovsky 2015, Corollary 1]), this yields an analogue of Corollary 4.6.3. However, its statement is less neat: unlike OBDDs, (d-)SDNNFs have no obvious notion of width, so the lower bound above refers to size rather than width, and it does not exactly match our upper bound. We obtain:

Corollary 4.7.2. *For any constant c, for any monotone DNF (resp., monotone CNF) φ with arity and degree bounded by c, there is a d-SDNNF for φ having size $|\varphi| \times 2^{O(\mathrm{tw}(\varphi))}$, and any d-SDNNF (resp., SDNNF) for φ has size $2^{\Omega(\mathrm{tw}(\varphi))}$.*

Our proof of Theorem 4.7.1 will follow the same overall structure as in the previous section. We first present the proof for monotone DNFs and d-DNNFs in Section 4.7.1, then we extend the result to monotone CNFs and SDNNFs in Section 4.7.2.

4.7.1 Monotone DNFs and d-SDNNFs

Recall that d-SDNNFs are structured by *v-trees*, which generalize variable orders. We first introduce *treesplitwidth*, a width notion that measures the performance of a v-tree by counting how many clauses it splits; and we connect treesplitwidth to treewidth. We use again dncpi-sets, and argue that a d-SDNNF structured by a v-tree must shatter a dncpi-set whose size follows the treesplitwidth of the v-tree. We

then show that shattering a dncpi-set forces d-SDNNFs to be large: instead of the easy OBDD result of the previous section (Lemma 4.6.8), we will need a much deeper result of Pipatsrisawat and Darwiche, rephrased in the setting of communication complexity by Bova, Capelli, Mengel, and Slivovsky, presented in [Pipatsrisawat and Darwiche 2010, Theorem 3] and in [Bova, Capelli, Mengel, and Slivovsky 2016]. Note that [Bova, Capelli, Mengel, and Slivovsky 2016], by a similar approach, shows an exponential lower bound on the size of d-SDNNF which is reminiscent of ours. However, their bound again applies to one well-chosen family of Boolean functions; our contribution is to show a general lower bound. In essence, our result is shown by observing that the family of functions used in their lower bound occurs "within" any bounded-degree, bounded-arity monotone DNF. Also note that a result similar to the lower bound of Corollary 4.7.2 is proven by Capelli in his thesis [Capelli 2016, Corollary 6.35] as an auxiliary statement to separate structured DNNFs and FBDDs. The result uses *MIM-width*, but Theorem 4.2.5 of [Vatshelle 2012], as degree and arity are bounded, implies that we could rephrase it to treewidth; further, the result assumes arity-2 formulas, but it could be extended to arbitrary arity as in [Capelli 2017, Theorem 12]. More importantly, the result applies only to monotone CNFs and not to DNFs .

Treesplitwidth. Informally, treesplitwidth is to v-trees what pathsplitwidth is to variable orders: it bounds the "best performance" of any v-tree.

Definition 4.7.3. Let $H = (V, E)$ be a hypergraph, and T be a v-tree over V. For any node n of T, we define $\text{Split}_n(T, H)$ as the set of hyperedges e of H that contain both a variable in T_n and one outside T_n (recall that T_n denotes the subtree of T rooted at n). Formally $\text{Split}_n(T, H)$ is defined as the following set of hyperedges:

$$\{e \in E \mid \exists v_\text{i} \in \mathsf{Leaves}(T_n) \text{ and } \exists v_\text{o} \in \mathsf{Leaves}(T \setminus T_n) \text{ such that } \{v_\text{i}, v_\text{o}\} \subseteq e\}$$

The *treesplitwidth* of T relative to H is $\text{tsw}(T, H) := \max_{n \in T} |\text{Split}_n(T, H)|$. The *treesplitwidth* $\text{tsw}(H)$ of H is then the minimum of $\text{tsw}(T, H)$ over all v-trees T of V. ◁

Again, the treesplitwidth of H is exactly the *branch-width* [Robertson and Seymour 1991] of the dual hypergraph of H, but treesplitwidth is more convenient for our proofs. As with pathsplitwidth and pathwidth (Lemma 4.6.5), we can bound the treewidth of a hypergraph by its treesplitwidth:

Lemma 4.7.4. *For any hypergraph $H = (V, E)$, we have* $\text{tw}(H) \leqslant 3 \times \text{arity}(H) \times \text{tsw}(H)$.

Proof. Let $H = (V, E)$ be a hypergraph, and T a v-tree over V witnessing that H has treesplitwidth $\text{tsw}(H)$. We will construct a tree decomposition T' of H of width $\leqslant 3 \times \text{arity}(H) \times \text{tsw}(H)$. The skeleton of T' is the same as that of T. Now, for each node $n \in T$, we call b_n the corresponding bag of T', and we define the labeling $\lambda(b_n)$ of b_n.

If n is an internal node of T with children n_l, n_r (recall that v-trees are assumed to be binary), then we define $\lambda(b_n) := \bigcup \text{Split}_n(T, H) \cup \bigcup \text{Split}_{n_l}(T, H) \cup \bigcup \text{Split}_{n_r}(T, H)$, and if n is a variable $v \in V$ (i.e., n is a leaf of T) then $\lambda(b_n) := \{v\}$. It is clear that the width of P is $\leqslant \max(3 \times \text{arity}(H) \times \text{tsw}(H), 1) - 1 \leqslant 3 \times \text{arity}(H) \times \text{tsw}(H)$.

The occurrence condition is verified: let e be an edge of H. If e is a singleton edge $\{v\}$ then it is included in b_v. If $|e| \geqslant 2$ then there must exists a node $n \in T$ such that $e \in \mathrm{Split}_n(T, H)$. If n is an internal node of T then $e \subseteq \bigcup \mathrm{Split}_n(T, H) \subseteq b_n$, and if n is a leaf node of T then it must have a parent p (since e is split), and $e \subseteq \bigcup \mathrm{Split}_n(T, H) \subseteq b_p$.

Connectedness is proved in the same way as in the proof of Lemma 4.6.5: for a given vertex $v \in V$, the nodes of T where each edge e containing v is split is a connected subtree of T without its root node: more precisely, they are all the ancestors of a leaf in e strictly lower than their the least common ancestor. Adding the missing root to each such subtree and unioning them all will result in the subtree of all ancestors of a vertex adjacent to v (included v itself) up to their least common ancestor a. Consequently, the set of nodes of T' containing v is a connected subtree of T', rooted in b_a. □

Moreover, using the same techniques that we used in the last section, we can show the analogue of Lemma 4.6.9. Specifically, given a monotone DNF φ on variables V, a v-tree T over V, and a dncpi-set S of φ, we say that T *shatters* S if there is a node n in T such that $S \subseteq \mathrm{Split}_n(T, \varphi)$. We now show that any v-tree over V must shatter a large dncpi-set (depending on the treewidth, degree, and arity):

Lemma 4.7.5. *Let φ be a monotone DNF, H its hypergraph, and T be a v-tree over its variables. Then there is a dncpi-set S of φ shattered by T such that $|S| \geqslant \left\lfloor \frac{\mathrm{tsw}(H)}{(\mathrm{arity}(H) \times \mathrm{degree}(H))^2} \right\rfloor$.*

Proof. The proof is just like that of Lemma 4.6.9, except with the new definition of split on v-trees. In particular, we use Lemmas 4.6.10, 4.6.11, and 4.6.12. □

Hence, to prove Theorem 4.7.1, the only missing ingredient is a lower bound on the size of d-SDNNFs that shatter large dncpi-sets. Specifically, we need an analogue of Lemma 4.6.8:

Lemma 4.7.6. *Let φ be a monotone DNF on variables V and $n \in \mathbb{N}$. Assume that, for every v-tree T over V, there is some dncpi-set S of φ with $|S| \geqslant n$, such that T shatters S. Then any d-SDNNF for φ has size $\geqslant 2^n - 1$.*

We will prove Lemma 4.7.6 in the rest of this section using a recent lower bound in [Bova, Capelli, Mengel, and Slivovsky 2016]. They bound the size of any d-SDNNF for the *set intersection* function, defined as $\mathrm{SINT}_n := (x_1 \wedge y_1) \vee \ldots \vee (x_n \wedge y_n)$. This bound is useful for us: a dncpi-set intuitively isolates some variables on which φ computes exactly SINT_n:

Lemma 4.7.7. *Let φ be a DNF with variables V, and let $S = \{D_1, \ldots, D_n\}$ be a dncpi-set of φ where every clause has size $\geqslant 2$. Pick two variables $x_i \neq y_i$ in D_i for each $1 \leqslant i \leqslant n$, and let $V' := \{x_1, y_1, \ldots, x_n, y_n\}$. Then there is a partial valuation ν of V with domain $V \setminus V'$ such that $\nu(\varphi) = \mathrm{SINT}_n$.*

Proof. Define the following partial valuation $\nu : V \setminus V' \to \{0, 1\}$ that maps all the variables of $\bigcup_{i=1,\ldots,n}(D_i \setminus \{x_i, y_i\})$ to 1 and all the other variables of $V \setminus \bigcup S$ to 0. Let us show that for a clause $D \in \varphi \setminus S$ we have $\nu(D) = 0$. Otherwise, as all the variables that ν maps to 1 are in $\bigcup_{i=1,\ldots,n} X_i \cup Y_i$, we should have $D \subseteq \bigcup S$, but because S is a dncpi-set we should have $D \in S$ which is a contradiction. Now, ν maps all the variables of $D_i \setminus \{x_i, y_i\}$ to 1, hence $\nu(\varphi)$ indeed captures SINT_n. Note that this result relies on monotonicity, and on the fact that φ is a DNF. □

This observation allows us to leverage the bound of [Bova, Capelli, Mengel, and Slivovsky 2016] on the size of d-SDNNFs that compute SINT_n, assuming that they are structured by an "inconvenient" v-tree:

Proposition 4.7.8 ([Bova, Capelli, Mengel, and Slivovsky 2016, Proposition 14])**.** *Let $X_n = \{x_1, \ldots, x_n\}$ and $Y_n = \{y_1, \ldots, y_n\}$ for $n \in \mathbb{N}$, and let T be a v-tree over $X_n \sqcup Y_n$ such that there exists a node $n \in T$ with $X_n \subseteq \mathsf{Leaves}(T_n)$ and $Y_n \subseteq \mathsf{Leaves}(T \setminus T_n)$. Then any d-SDNNF structured by T computing SINT_n has size $\geqslant 2^n - 1$.*

In our setting, an "inconvenient" v-tree for a dncpi-set is one that shatters it: each clause of the dncpi-set is then partitioned in two non-empty subsets where we can pick x_i and y_i for Lemma 4.7.7. We can then prove Lemma 4.7.6:

Proof of Lemma 4.7.6. Let C be a d-SDNNF structured by a v-tree T that captures φ. Consider the dncpi-set $S = \{D_1, \ldots, D_m\}$ of size $\geqslant n$ of φ that is shattered by T (note that this implies in particular that every clause contains at least two variables). Consider the node u of T which witnesses this. We can write each clause D_i of S as $X_i \sqcup Y_i$, where X_i is $D_i \cap \mathsf{Leaves}(T \setminus T_u)$ and Y_i is $D_i \cap \mathsf{Leaves}(T_u)$. Then according to Lemma 4.7.7, there exists a valuation ν of the variables of φ with domain $V \setminus \{x_1, y_1, \ldots, x_m, y_m\}$, where $x_i \in X_i$ and $y_i \in Y_i$ for $1 \leqslant i \leqslant m$, such that $\nu(\varphi)$ captures the Boolean function SINT_m, hence we know that $\nu(C)$ also captures SINT_m. But by Proposition 4.7.8, we have $|\nu(C)| \geqslant 2^m - 1 \geqslant 2^n - 1$, hence $|C| \geqslant 2^n - 1$. □

This concludes the proof of Theorem 4.7.1 (in the DNF case).

4.7.2 From DNFs to CNFs

We now argue that Theorem 4.7.1 also holds for monotone CNFs and SDNNFs. Note that we cannot use a dualization argument as we did in the previous section, as we are now working with DNNFs that are not necessarily deterministic. Observe that Definition 4.6.7 and Lemma 4.7.5 can also apply to monotone CNFs as these only use the hypergraph corresponding to the formula, not the semantics of the formula. Hence, in order to apply the same arguments as in the DNF case and prove an analogue of Lemma 4.7.6 in the CNF case, the only difference is that we would need to consider the function $f_n := (x_1 \vee y_1) \wedge \ldots \wedge (x_n \vee y_n)$ and obtain analogues of Lemma 4.7.7 and Proposition 4.7.8 for that function. This is clear for Lemma 4.7.7, so we only need to check that the analogue of Proposition 4.7.8 holds. To understand why, we need to go deeper into the proof from [Bova, Capelli, Mengel, and Slivovsky 2016]. They paraphrase a result of [Pipatsrisawat and Darwiche 2010] in the following way:

Theorem 4.7.9 ([Bova, Capelli, Mengel, and Slivovsky 2016, Theorem 13] and [Pipatsrisawat and Darwiche 2010, Theorem 3])**.** *Let C be a SDNNF on variables V structured by a v-tree T, and let f be the function that it captures. For every node $n \in T$, the function f has a rectangle cover of size $\leqslant |C|$ with partition $(V \cap \mathsf{Leaves}(T_n), V \cap \mathsf{Leaves}(T \setminus T_n))$.*

Here, a *rectangle cover* of a Boolean function $f : X \sqcup Y \to \{0,1\}$ with partition (X,Y) is a disjunction $\bigvee_{i=1}^{m}(g_i(X) \wedge h_i(Y))$ equivalent to f such that g_i (resp., h_i) is a Boolean function on variables X (resp., on variables Y), and m is its *size*. This notion is a standard tool for showing lower bounds in communication complexity. Therefore, we are interested in the smallest size of a rectangle cover for the function $f_n : X \sqcup Y \mapsto (x_1 \vee y_1) \wedge \ldots \wedge (x_n \vee y_n)$ under partition (X,Y). But it is known from communication complexity that any rectangle cover for the function *set disjunction* $\mathrm{SDISJ}_n : X \sqcup Y \mapsto (\neg x_1 \vee \neg y_1) \wedge \ldots \wedge (\neg x_n \vee \neg y_n)$ has size $\geqslant 2^n$ (see paragraph "Fooling set method", page 5 of [Sherstov 2014]). Moreover, it is easy to see that we can turn any rectangle cover of size m for f_n with partition (X,Y) into a rectangle cover for SDISJ_n of the same size and under the same partition, which implies that any such cover for f_n must be of size at least $\geqslant 2^n$ and concludes the proof of Theorem 4.7.1 for CNFs and SDNNFs. Indeed, let $\bigvee_{i=1}^{m}(g_i(X) \wedge h_i(Y))$ be a rectangle cover for f_n with partition (X,Y). When ν is a Boolean valuation from S to $\{0,1\}$, let us write $\overline{\nu}$ for the Boolean valuation from S to $\{0,1\}$ defined by $\overline{\nu}(s) := 1 - \nu(s)$ for $s \in S$. We then define $\overline{g_i}$ for $1 \leqslant i \leqslant n$ (resp., $\overline{h_i}$) to be the Boolean function from X (resp., Y) to $\{0,1\}$ defined by $\overline{g_i}(\nu) := g_i(\overline{\nu})$ for all valuations $\nu : X \to \{0,1\}$ (resp., $\overline{h_i}(\nu) := h_i(\overline{\nu})$). One can then check that $\bigvee_{i=1}^{m}(\overline{g_i}(X) \wedge \overline{h_i}(Y))$ is a rectangle cover for SDISJ_n of size m with partition (X,Y).

Hence, we have shown that, up to constant factors in the arity and degree of a DNF (resp., CNF) φ, any equivalent d-SDNNF (resp., SDNNF) must have size at least exponential in the treewidth of φ, thus generalizing our lower bound on OBDDs from Section 4.6. We will now use our lower bound on d-SDNNF in the context of lineage representation.

4.8 Application to Query Lineages

In this section, we adapt the lower bound of the previous section to the computation of query lineages on relational instances. Like in [Amarilli, Bourhis, and Senellart 2016], for technical reasons, we must assume a graph signature. We first recall some preliminaries and then state our result.

Preliminaries. We consider σ-instances, where σ is a fixed *arity-2* relational signature, i.e., a relational signature whose relations all have arities in $\{1,2\}$, and with at least one relation of arity 2.

Recall from Section 1.2 the definition of unions of conjunctive queries with disequalities ($\mathrm{UCQ}^{\neq}$). We say that a $\mathrm{UCQ}^{\neq}$ is *connected* if the Gaifman graph of each disjunct (seen as an instance, and ignoring $\neq$-atoms) is connected. For instance, letting σ_R consist of one arity-2 relation R, the following connected $\mathrm{UCQ}^{\neq}$ tests if there are two facts that share one element: $Q_{\mathrm{p}} : \exists xyz\,(R(x,y) \vee R(y,x))) \wedge (R(y,z) \vee R(z,y)) \wedge x \neq z$. While Q_{p} is not given as a disjunction of $\mathrm{CQ}^{\neq}$s, it can be rewritten to one using distributivity, as follows:

$$\begin{aligned} Q_{\mathrm{p}} : (&\exists xyz\, R(x,y) \wedge R(y,z) \wedge x \neq z) \\ \vee(&\exists xyz\, R(x,y) \wedge R(z,y) \wedge x \neq z) \\ \vee(&\exists xyz\, R(y,x) \wedge R(y,z) \wedge x \neq z) \end{aligned}$$

We then see that Q_{p} is connected.

Problem statement. We study when query lineages can be computed efficiently in data complexity. A first question asks which *queries* have tractable lineages on all instances: Jha and Suciu showed that *inversion-free* UCQ$^{\neq}$ queries admit OBDD representations in this sense [Jha and Suciu 2012, Theorem 3.9], and Bova and Szeider have recently shown that UCQ$^{\neq}$ queries with inversions do not even have tractable d-SDNNF lineages [Bova and Szeider 2017, Theorem 5]. A second question asks which *instance classes* ensure that *all queries* have tractable lineages on them. This was studied for OBDD representations in [Amarilli, Bourhis, and Senellart 2016]: bounded-treewidth instances have tractable OBDD lineage representations for any MSO query ([Amarilli, Bourhis, and Senellart 2016, Theorem 6.5], using [Jha and Suciu 2012]); conversely there are *intricate* queries (a class of connected UCQ$^{\neq}$ queries) whose lineages never have tractable OBDD representations in the instance treewidth [Amarilli, Bourhis, and Senellart 2016, Theorem 8.7]. The idea behind intricate queries is that the treewidth of their lineage (as a DNF) cannot be lower than the treewidth of the instances (under some conditions, that we will explain latter in the proof). We recall here the definition of an intricate connected UCQ$^{\neq}$. For this, we will need the notion of a *line instance*:

Definition 4.8.1 ([Amarilli, Bourhis, and Senellart 2016, Definition 8.4])**.** A *line instance* is an instance I of the following form: a domain $a_1, \ldots, a_n$, and, for $1 \leqslant i < n$, one single binary fact between a_i and a_{i+1}: either $R(a_i, a_{i+1})$ or $R(a_{i+1}, a_i)$ for some binary $R \in \sigma$ (Recall that σ includes at least one binary relation.) ◁

Definition 4.8.2 ([Amarilli, Bourhis, and Senellart 2016, Definition 8.5])**.** A connected UCQ$^{\neq}$ Q is *intricate* if, for every line instance I with $|I| = 2|Q| + 2$, letting F and F' be the two facts of I incident to the middle element a_{n+2}, there is a *minimal* match of Q on I that includes both F and F'. ◁

Here, by a *minimal match of a* UCQ$^{\neq}$ Q, we mean a subinstance I' of I that satisfies Q and that is minimal by inclusion. The query Q_{p} above is an example of an intricate query on the signature σ_R. Amarilli, Bourhis, and Senellart then showed the following:

Theorem 4.8.3 ([Amarilli, Bourhis, and Senellart 2016, Lemma 8.2] and [Amarilli, Bourhis, and Senellart 2016, Theorem 8.7])**.** *There is a constant $d \in \mathbb{N}$ such that the following is true. Let σ be an arity-2 signature, and Q a connected* UCQ$^{\neq}$ *which is intricate on σ. For any instance I on σ, any OBDD representing the lineage of Q on I has size $2^{\Omega(\mathrm{tw}(I)^{1/d})}$.*

This result shows that we must bound instance treewidth for intricate queries to have tractable OBDDs, but leaves the question open for more expressive lineage representations.

Result. Our bound in the previous section allows us to extend Theorem 4.8.3 from OBDDs to d-SDNNFs, yielding the following:

Theorem 4.8.4. *There is a constant $d \in \mathbb{N}$ such that the following is true. Let σ be an arity-2 signature, and Q a connected* UCQ$^{\neq}$ *which is intricate on σ. For any instance I on σ, any d-SDNNF representing the lineage of Q on I has size $2^{\Omega(\mathrm{tw}(I)^{1/d})}$.*

Hence, given an instance family $\mathcal{I}$ satisfying the constructibility requirement of Theorem 8.1 of [Amarilli, Bourhis, and Senellart 2016], there are two regimes: (i.) $\mathcal{I}$ has bounded treewidth and then all MSO queries have d-SDNNF lineages on instances of $\mathcal{I}$ that are computable in linear time; or (ii.) the treewidth is unbounded and then there are $\text{UCQ}^{\neq}$ queries (the intricate ones) whose lineages on instances of $\mathcal{I}$ have no d-SDNNF representations polynomial in the instance size.

We prove Theorem 4.8.4 in the rest of this section. We will need the notion of a *topological minor* of an undirected graph:

Definition 4.8.5. An *embedding* of an undirected graph H in a (undirected) graph G is an injective mapping f from the vertices of H to the vertices of G and a mapping g that maps the edges $\{u, v\}$ of H to paths in G from $f(u)$ to $f(v)$, all paths being vertex-disjoint. A graph H is a *topological minor* of a graph G if there is an embedding of H in G. ◁

We also recall that a graph is *degree-3* if each vertex has at most 3 adjacent neighbors. Because the proof of Theorem 4.8.4 is quite technical, we first want to give a high-level intuition of how it works:

Proof sketch. As in [Amarilli, Bourhis, and Senellart 2016], we use a result of [Chekuri and Chuzhoy 2014] to show that the Gaifman graph of I has a degree-3 topological minor S of treewidth $\Omega(\text{tw}(I)^{1/d})$ for some constant $d \in \mathbb{N}$; we also ensure that S has sufficiently high *girth* relative to Q. We focus on a subinstance I' of I that corresponds to S: this suffices to show our lower bound, because we can always compute a tractable representation of $\varphi(Q, I')$ from one of $\varphi(Q, I)$. Now, we can represent $\varphi(Q, I')$ as a minimized DNF ψ by enumerating its minimal matches: ψ has constant arity because the number of atoms of Q is fixed, and it has constant degree because S has constant degree and Q is connected. Further, as Q is intricate and I' has high girth relative to Q, we can ensure that this DNF has treewidth $\Omega(\text{tw}(I'))$. We conclude by Theorem 4.7.1: all d-SDNNFs representing $\varphi(Q, I')$, hence $\varphi(Q, I)$, have size $2^{\Omega(\text{tw}(I)^{1/d})}$. □

We are now ready to formally prove Theorem 4.8.4. We will use the restatement of the main result of [Chekuri and Chuzhoy 2014] given in [Amarilli, Bourhis, and Senellart 2016] (where a *degree-3* graph is one where the maximal degree is 3):

Lemma 4.8.6 ([Chekuri and Chuzhoy 2014], rephrased as [Amarilli, Bourhis, and Senellart 2016, Lemma 4.4])**.** *There is $c \in \mathbb{N}$ such that, for any degree-3 planar graph H, for any graph G of treewidth $\geqslant |V(H)|^c$, H is a topological minor of G.*

We set $d := 2c$, where c is given as in Lemma 4.8.6. Fix the arity-2 signature σ and the intricate query Q. As σ is nonempty, the tautological and vacuous $\text{UCQ}^{\neq}$ queries are not intricate, so we can assume that Q is not trivial in this sense. We denote by $|Q|$ the number of atoms of Q.

We now define the class of subgraphs that we wish to extract. Recall that the *girth* of an undirected graph is defined as the length of the shortest simple cycle in the graph (or ∞ if the graph is acyclic). Let us define an infinite family $\mathcal{S} = S_2, \ldots, S_n, \ldots$ of graphs to extract, such that, for each $i \in \mathbb{N}$, the graph S_i has:

1. maximal degree 3;

2. treewidth i;
3. $\leqslant \alpha \times i^2$ vertices for some constant $\alpha \geqslant 1$ depending only on Q;
4. no vertex of degree 1;
5. girth $> 2\,|Q| + 2$;

We can define each S_i by starting, for instance, with a wall graph [Dragan, Fomin, and Golovach 2011], to satisfy the first three conditions (for some fixed α). We then iteratively remove all vertices of degree 1, which does not impact treewidth since:

- Treewidth cannot increase when we do this;
- The graph cannot become empty (because its initial treewidth is $\geqslant 2$, so it has a cycle, which will never be removed);
- Treewidth cannot decrease either. Indeed, if we consider a graph G and the result G' of removing one vertex v of degree 1 in G, given a tree decomposition T' of G', we can construct a tree decomposition T of G by adding one bag with v and its one incident vertex w, and connecting it to a bag containing w in T' (if one exists; we connect it arbitrarily otherwise); the result is clearly a tree decomposition of G, and the width is unchanged because the maximal bag size in T' is $\geqslant 2$ (since G' is non-empty by the previous bullet point).

Hence, the graph obtained when we have removed all vertices of degree 1 satisfies requirements 1–4. Last, we subdivide each edge into a path of length $2\,|Q| + 3$ to ensure that the girth condition is respected: this satisfies requirement 5, does not affect requirements 1–2 or 4, and requirement 3 is still satisfied up to multiplying α by $3 \times (2\,|Q| + 2) + 1$ (each path replacing an edge introduces $(2\,|Q| + 2)$ new vertices, and since the graph is degree-3, an upper bound on the number of edges is three times the number of vertices).

We now make explicit the function hidden in the Ω-notation in the exponent of the bound that we wish to show in the statement of Theorem 4.8.4. This function will only depend on Q. Define the increasing function $f : k \mapsto \frac{1}{\alpha} k^{1/d}$, and let $k_0 \in \mathbb{N}$ be the smallest value of k such that $f(k) \geqslant 2$. We will show that the size of a d-SDNNF for an input instance I is $\geqslant 2^{\beta f(\mathrm{tw}(I))}$ when $\mathrm{tw}(I)$ is large enough, for some constant $\beta > 0$ to be defined later, depending only on Q. This means indeed that $\beta f(\mathrm{tw}(I))$ is in $\Omega(\mathrm{tw}(I)^{1/d})$. We assume $\mathrm{tw}(I) \geqslant k_0$ (and thus $f(k) \geqslant 2$) in what follows.

Let I be the input instance on σ, let G be the Gaifman graph of I, and let $k := \mathrm{tw}(I) = \mathrm{tw}(G)$. Let $k' := \lfloor f(k) \rfloor$, and consider $S_{k'}$, which is well-defined because k' is an integer which is $\geqslant 2$. We know that the number $n_{k'}$ of vertices of $S_{k'}$ is such that $n_{k'} \leqslant \alpha k'^2$, hence we have $n_{k'} \leqslant k^{1/c}$, so the treewidth k of G is $\geqslant n_{k'}^c$. Hence, we know by Lemma 4.8.6 that $S_{k'}$ is a topological minor of G. Let G' be the subgraph of G corresponding to this topological minor: it is a subgraph of G, and a subdivision of $S_{k'}$.

We will extract a corresponding subinstance I' of I whose Gaifman graph is G'. For simplicity, we will ensure that I' is *Gaifman-tight*. An instance I_0 is *Gaifman-tight* if two conditions hold: first, letting G_0 be the Gaifman graph of I_0, for each edge $\{a, b\}$ of G_0, there is exactly one fact of I_0 containing a and b (hence, of the

form $R(a,b)$ or $R(b,a)$, with $a \neq b$); second, every fact of I_0 is a binary fact with two distinct elements (of the form $R(c,d)$ with $c \neq d$). Intuitively, an instance is Gaifman-tight if it is exactly obtained from its Gaifman graph by choosing one relation name and orientation for each edge of the Gaifman graph.

We define a Gaifman-tight subinstance I' of I with Gaifman graph G' by keeping, for every edge $\{a,b\}$ of G', exactly one binary fact of I containing the two elements a and b (which must exist by definition of the Gaifman graph). By construction, the Gaifman graph of I' is then G'. Hence, we know the following about the subinstance I' of I and its Gaifman graph G' (the numbering of this list follows the list of conditions on $\mathcal{S}$):

1. For every element a of I', there are at most 3 facts where a occurs (because G' has maximal degree 3).
2. The treewidth of I' is k'.
3. (N/A: There is no analogue of the requirement 3 imposed on $\mathcal{S}$)
4. There are no vertices of degree 1 in G'.
5. The girth of G' is $> 2\,|Q| + 2$ (because as a subdivision of $S_{k'}$ its girth is at least that of $S_{k'}$).
6. I' is Gaifman-tight.

We will now define a DNF representation of $\varphi(Q, I')$. Remember that Q is a UCQ$^{\neq}$, so it is monotone, hence we can define $\varphi(Q, I')$ to be a monotone DNF. As Q is not trivial, $\varphi(Q, I')$ will contain at least one nonempty clause. Further, the DNF can be computed as a minimized DNF by taking the disjunction of conjunctions that stand for each minimal match of Q in I'. The following is then easy to see (and this monotone DNF representation is clearly unique):

$$\varphi(Q, I') := \bigvee_{\substack{M \text{ minimal} \\ \text{match of } Q \\ \text{in } I'}} \bigwedge_{F \in M} F$$

Let H be the hypergraph of this DNF. To be able to usefully apply Theorem 4.7.1, we must show that the arity and degree of H are constant, and that $\mathrm{tw}(H)$ is $\Omega(\mathrm{tw}(I'))$. We first show the first claim. The arity of H is clearly bounded from above by the size of a minimal match of Q in I', whose size is clearly bounded from above by $|Q|$, which is constant. As for the degree of H, as Q is a connected query, any minimal match of Q on I' involving some fact F must be contained in the subinstance of I' induced by the ball of radius $|Q|$ centered around the elements of F in G': as the degree of G' is at most 3, this ball has constant size, so, as σ is fixed, F can only occur in constantly many different matches, and the degree is constant. We now show that $\mathrm{tw}(H)$ is $\Omega(\mathrm{tw}(I'))$: we show this in the following lemma, which captures the essence of intricate queries (namely: under some conditions, their lineage never has lower treewidth than the input instance):

Lemma 4.8.7. *Let σ be an arity-2 signature, let Q be a connected* UCQ$^{\neq}$ *which is intricate for σ, and let I' be a Gaifman-tight instance on σ whose Gaifman graph has no degree-1 vertex and has girth $> 2\,|Q| + 2$. Then, letting H be the hypergraph of the monotone DNF representing $\varphi(Q, I')$, we have* $\mathrm{tw}(H) \geqslant \left\lfloor \frac{\mathrm{tw}(I')}{2} \right\rfloor$.

Let us conclude the proof of Theorem 4.8.4 using Lemma 4.8.7, and show Lemma 4.8.7 afterwards. As the arity and degree of H are bounded by constants, by Theorem 4.7.1, we know that any d-SDNNF for $\varphi(Q, I')$ has size $\geqslant 2^{\beta' \mathrm{tw}(H)}$ for some constant $\beta' > 0$ (depending only on the arity and degree bounds on I' given above, which depend only on Q), which by Lemma 4.8.7 is $\geqslant 2^{\beta \mathrm{tw}(I')}$ for a different constant $\beta > 0$ and $\mathrm{tw}(I')$ large enough. By definition of $S_{k'}$, we obtain the lower bound of $2^{\beta f(k)}$ for k large enough. Now, to conclude, we must show that this lower bound also applies to any d-SDNNF for $\varphi(Q, I)$. But it is clear that, from any d-SDNNF C for $\varphi(Q, I)$, we can obtain a d-SDNNF C' for $\varphi(Q, I')$ which is no larger than C (structured by a v-tree obtained from that of C), simply by evaluating to 0 all inputs corresponding to facts of $I \setminus I'$. Hence, the lower bound also applies to a d-SDNNF for $\varphi(Q, I)$, establishing the result of Theorem 4.8.4.

All that remains is to show Lemma 4.8.7. Let us fix the graph signature σ, the connected UCQ$^{\neq}$ Q which is intricate for σ, and the instance I' on σ satisfying the conditions. We say that two different facts $R(a, b)$ and $S(c, d)$ of I' *touch* if they share an element, formally, $|\{a, b\} \cap \{c, d\}| = 1$: as I' is Gaifman-tight, remember that we must have $a \neq b$, $c \neq d$, and $\{a, b\} \neq \{c, d\}$. The key for Lemma 4.8.7 is then captured in the following auxiliary claim:

Claim 4.8.8. *Let F and F' be two facts of I' that touch. Then there is a minimal match M of Q such that $\{F, F'\} \subseteq M$.*

Proof. Let G be the Gaifman graph of I'. Consider the two edges e and e' standing for F and F' in G: these edges are incident in G, so we write without loss of generality $e = \{u, v\}$ and $e' = \{v, w\}$. Fix $n := |Q|$. Define a path $\pi = uu_1 \dots u_n$ in G of $|Q|$ edges by exploring G from u: initially we are at u and call v the *predecessor vertex*, and whenever we reach some vertex x, we visit a neighbor of x which is different from the predecessor of x, and set x to be the new predecessor. Such a path exists, because this exploration can only get stuck on a vertex of degree 1 (i.e., a vertex that we cannot exit except by going back on its predecessor), and this cannot happen by our assumption that G has no vertex of degree 1. We define a path $\pi' = ww_1 \dots w_n$ in G of $|Q|$ edges by exploring from w with predecessor v in the same way. Now, we consider the path ρ obtained by concatenating the reverse of π, e, e', and π', namely: $\rho : u_n, \dots, u_1, u, v, w, w_1, \dots, w_n$. We claim that this path is a simple path, i.e., no two vertices in the path are the same. Indeed, by definition, no two consecutive vertices can be the same in π, in π', or in u, v, w. Further, two vertices separated by one single vertex cannot be the same: this is the case in π and π' because we do not go back to the predecessor vertex in the exploration, and initially we do not go back on v: and for u and w we know that they are different because F and F' touch and I' is Gaifman-tight. Last, two vertices further apart in ρ cannot be equal, because otherwise the path ρ would contain a simple cycle of G, which would contradict the hypothesis on the girth of G.

Hence, ρ is a simple path of the Gaifman graph G of I'. Consider the sequence of facts L of I' that witness the existence of each edge of ρ, which is unique because I' is Gaifman-tight; in particular we choose F and F' as witnesses for e and e'. Recall now the definition of a line instance, and of a UCQ$^{\neq}$ Q being intricate (Definitions 4.8.1 and 4.8.2). The sequence of facts L is a line instance, with $|L| = 2|Q| + 2$, and the two facts incident to the middle element are F and F'. Hence, the definition of

intricate queries ensures that there is a minimal match M of Q on L that includes both F and F'. As L is a subinstance of I', the match M is still a match of Q on I', and it is still minimal, because any match $M' \subseteq M$ would also satisfy $M' \subseteq L$ and contradict the minimality of M on L. Hence, M is the desired minimal match, which concludes the proof. □

We are now ready to prove Lemma 4.8.7 from Claim 4.8.8, which is the only missing part of the proof of Theorem 4.8.4:

Proof of Lemma 4.8.7. Fix σ, Q, and I', consider the monotone DNF representation of $\varphi(Q, I')$ and its hypergraph H. To show the desired inequality, it suffices to show that, from a tree decomposition T of H where the maximal bag size is k, we can construct a tree decomposition T' of I' whose maximal bag size is no greater than $2k$. Let T be a tree decomposition of H, and construct T' to have same skeleton as T. We define the labeling $\lambda(b')$ of every bag b' of T' to be the set of vertices occurring in the label $\lambda(b)$ of the corresponding bag b of T (which consists of variables of $\varphi(Q, I')$, hence of facts of I'): this clearly satisfies the size requirements. We must now show that T' is a tree decomposition of I'.

To show the occurrence requirement, we must show that for every fact F of I', there is a bag of T' containing its two elements. To show this, it suffices to show that there is a bag of T that contains F (as a vertex of H). As the Gaifman graph of I' has no vertex of degree 1, there must be a fact F' of I' that touches F, and we can conclude using a consequence of Claim 4.8.8: F must occur in a minimal match of Q on I' (together with F', but we do not use this), hence it occurs in a clause of $\varphi(Q, I')$, and the occurrence requirement on T ensures that F occurs in a bag of T.

To show the connectedness requirement, pick an element a of I'. Its occurrences in T' are the union of the occurrences in T of the facts that contain a, which are connected subtrees of T by the connectedness requirement of T. Hence, it suffices to show that their union is connected. To do this, let us show that for any two facts F and F' that contain a, then the subtrees T_F and $T_{F'}$ of their occurrences in T necessarily intersect. This is trivial if $F = F'$; now, if $F \neq F'$, since I' is Gaifman-tight, the facts F and F' must touch in I' (they cannot share exactly the same elements). Now, we use Claim 4.8.8 to conclude that F and F' occur together in a minimal match M of Q on I'. Hence, there is a clause of $\varphi(Q, I')$ which contains both F and F', which ensures that F and F' occur together in a bag of T, so T_F and $T_{F'}$ intersect. This shows that T' is indeed a tree decomposition of I', which concludes the proof. □

Conclusion. This concludes the proof of Theorem 4.8.4 and the presentation of our work in Chapter 4 on the connections between width-based and structure-based circuits classes of knowledge compilation. Specifically, we have seen how to efficiently convert bounded-treewidth Boolean circuits to d-SDNNF and were able to use this result for PQE of CFG-Datalog. We have also shown lower bounds on knowledge compilation formalisms and applied these to point to the limits of the intensional approach for PQE.

Conclusion and Perspectives

In this chapter we first give a summary of our contributions, then present a brief overwiew of open questions left open by our work, and finally give general perspectives about our thesis.

Summary

In this thesis, we have investigated questions related to the combined complexity of the probabilistic query evaluation problem. We have explored three major axes: (i) the combined complexity of the *probabilistic graph homomorphism problem*, which can be seen as PQE of conjunctive queries on arity-two tuple independent databases, (ii) the combined complexity of evaluating CFG-Datalog of bounded body size on bounded treewidth (non-probabilistic) databases, (iii) the connections between width-based and structure-based representations of Boolean functions in knowledge compilation, and their applications to PQE. Unrelated to combined complexity and not presented in this thesis, I have also studied the compilation of *safe queries* to deterministic decomposable circuits (this work was published in AMW'2018 [Monet and Olteanu 2018]).

Probabilistic graph homomorphism. In Chapter 2 we have introduced the probabilistic graph homomorphism problem, also known in the database community as probabilistic evaluation of conjunctive queries on TID instances, and studied its combined complexity for various restricted classes of query and instance graphs. Our classes illustrate the impact on PHom of various features: acyclicity, two-wayness, branching, connectedness, and labeling. As we show, the landscape is already quite enigmatic, even for those seemingly restricted classes. In particular, we have identified four incomparable maximal tractable cases, reflecting various trade-offs between the expressiveness that we can allow in the queries and in the instances:

- arbitrary queries on unlabeled downward trees (Proposition 2.3.6);
- one-way path queries on labeled downward trees (Proposition 2.4.9);
- connected queries on two-way labeled path instances (Proposition 2.4.10);
- downward tree queries on unlabeled polytrees (Proposition 2.5.3).

These results all extend to disconnected instances, as shown in Section 2.3.3. The tractability border is described in Tables 2.1 page 39, 2.2 page 43, and 2.3 page 51. In particular, our tractability results are essentially always about paths, i.e., when the instance or the query is a path, or can be converted to a path. Our proofs appeal to various seemingly unrelated notions (e.g., β-acyclic lineages, tree automata techniques, graded DAGS, etc.), and we wonder if one could develop a general theory to better unify our framework.

CFG-Datalog on treelike instances. In Chapter 3, we introduced CFG-Datalog, a stratified Datalog fragment whose evaluation and provenance computation has FPT-linear complexity when parameterized by instance treewidth and program body size. For the case of conjunctive queries, we showed that bounding their treewidth is not the right way to ensure FPT translatability to automata, and we introduced the class of *bounded-simplicial-width* conjunctive queries, which solves this problem. CFG-Datalog can be understood as an extension of this fragment to disjunction, clique-guardedness, stratified negation, and inflationary fixpoints, that preserves tractability.

The complexity result is obtained via compilation to a variant of alternating two-way automata, and via the computation of a provenance representation in the form of stratified cycluits, a generalization of provenance circuits that we hope to be of independent interest. Our results captures the tractability of such query classes as two-way regular path queries and α-acyclic conjunctive queries.

On connecting width and structure in knowledge compilation. In Chapter 4 we have shown tight connections between structured circuit classes and width measures on circuits. Our main upper bound is to constructively rewrite bounded-treewidth circuits to d-SDNNFs in time linear in the circuit and singly exponential in the treewidth. We show lower bounds for arbitrary monotone CNFs or DNFs under degree and arity assumptions; we also show a lower bound for pathwidth and OBDDs. Our results have applications to tasks that extend query evaluation: probabilistic query evaluation, computation of lineages, enumeration, etc. In particular, we were able to transform provenance cycluits of CFG-Datalog on treelike instances into d-SDNNF, allowing us to solve PQE for these classes in linear time in the data and doubly exponential in the query. This contrasts with the nonelementary bound of [Amarilli, Bourhis, and Senellart 2015]. We also used our lower bound on d-SDNNF size to show that any d-SDNNF representing the provenance of an *intricate query* must have size that is exponential in the treewidth of the database.

Open Questions and Directions for Future Work

Our work raises a number of open questions and directions for future work, some of which we present here.

First, the query and instance features studied in Chapter 2 could be completed by other dimensions: e.g., studying an *unweighted* case inspired by counting CSP where all probabilities are $\frac{1}{2}$ (as our hardness proofs seem to heavily rely on some edges being certain); imposing *symmetry* in the sense of [Beame, Van den Broeck, Gribkoff, and Suciu 2015]; or alternatively restricting the *degree* of graphs (though all our hardness proofs on polytrees and lower classes can seemingly be modified to work on bounded-degree). Another option would be to modify some of the existing dimensions: first, polytrees could be generalized to *bounded-treewidth instances*, as we believe that the relevant tractability result (Proposition 2.5.3) adapts to this setting; second, non-branching instances could be generalized to *bounded-pathwidth instances*, or *caterpillar graphs* (graphs in which all nodes are at distance at most one from a central path [Harary and Schwenk 1973]) or maybe general instances with the $\underline{X}$-property (recall Definition 2.4.11).

Of course, another natural direction would be to lift the arity-two restriction, although it is not immediate to generalize the definition of our classes to work in higher arity signatures. We could also extend the query language: in particular, allow *unions* of conjunctive queries as in [Dalvi and Suciu 2012]; allow a *descendant* axis in the spirit of XML query languages [Benedikt and Koch 2009]; or more generally allow fixpoint constructs as done in Chapter 3 in the non-probabilistic case. An interesting question is whether an extended query language could capture the tractability results obtained in the context of probabilistic XML by [Cohen, Kimelfeld, and Sagiv 2009] (remembering, however, that such results crucially depend on having an order relation on node children [Amarilli 2014]). Another possibility would be to search for extensions of the β-acyclicity approach, and investigate which restrictions on the queries and instances ensure that the lineages are β-acyclic. The connection to CSP would seem to warrant further investigation. In particular, we do not know whether one could show a general dichotomy result on the combined complexity of query evaluation on TID instances, to provide a probabilistic analogue to the Feder–Vardi conjecture [Feder and Vardi 1998].

Concerning our work on CFG-Datalog, a careful inspection of the proofs seems to indicate that our results can be used to derive PTIME combined complexity results on arbitrary instances, e.g., XP membership when parameterizing only by program size; this would recapture in particular the tractability of some query languages on arbitrary instances, such as α-acyclic queries or SAC2RPQs. For conjunctive queries of bounded simplicial width, we could investigate if computing a simplicial tree decomposition is FPT (parameterized by the simplicial width), as in Theorem 1.5.2 for the case of treewidth. We also wonder if we could easily extend our cycluit framework to support more expressive provenance semirings than Boolean provenance (e.g., formal power series [Green, Karvounarakis, and Tannen 2007]).

Last, our work in Chapter 4 also raises a number of open questions. The d-SDNNF obtained in the proof of Theorem 4.3.2 does *not* respect the definition of a *sentential decision diagram* (SDD) [Darwiche 2011]. Can this be fixed, and Theorem 4.3.2 extended to SDDs? Or is it impossible, which could solve the open question [Bova 2016] of separating SDDs and d-SDNNFs? Can we weaken the hypotheses of bounded degree and arity in Corollaries 4.6.3 and 4.7.2, and can we rephrase the latter to a notion of (d-)SDNNF width to match more closely the statement of the former? More importantly, can we merge the acyclification procedure of Theorem 4.5.3 with our upper bound of Theorem 4.3.2 into one algorithm, in order to have only one exponential in the treewidth? This would imply a combined complexity of PQE for bounded body-size CFG-Datalog programs P on treelike instances I in time linear in $|I|$ and singly exponential in $|P|$, instead of the double exponential in $|P|$ that we currently have in Theorem 4.5.4. Eventually, Section 4.8 shows that d-SDNNF representations of the lineages of intricate queries are exponential in the treewidth; we conjecture a similar result for pathwidth and OBDDs, but this would require a pathwidth analogue of the minor extraction results of [Chekuri and Chuzhoy 2014].

Perspectives

On using the TID model. One general limitation of our work is the use of the TID model. First, in a vast majority of practical applications, performing *exact* probability computation is not crucial. Indeed, the sheer meaning of the probabilities

of tuples is often not very clear in practice: these are usually obtained from statistical methods, or roughly estimated by humans (e.g., in the example from Table 1.2). In this regard, *approximate probability computation* seems to be closer to real-world applications than exact probability computation. However, our point of view was that one first has to understand the difficulties of exact probability computation in order to determine if approximating methods are needed.

Second, as we have observed in Section 1.3, the TID model cannot represent arbitrary probability distributions, whereas in practice correlations naturally arise. We could have used more expressive formalisms for probabilistic databases, such as the pc-tables model, or the block independent model (BID). For instance, [Amarilli, Bourhis, and Senellart 2015] shows that probabilistic evaluation of a fixed MSO query can be done in linear time on bounded-treewidth pc-tables, provided that the correlations are themselves of bounded treewidth.

Third, another weakness of the TID model is that it uses the *closed world assumption*, which in this case amounts to saying that tuples not present in the database have probability zero. However, in many applications, the fact that a tuple is not present does not mean that it cannot be true, but rather that our confidence about this tuple is below a certain threshold. Hence, a probabilistic model that would be closer these applications could be a model where absent tuples have in fact a low threshold probability; this resembles the *symmetric* model used in [Beame, Van den Broeck, Gribkoff, and Suciu 2015], for instance. In some other applications, the fact that a tuple is not present does not mean anything about its probability to be true. For instance in the context of knowledge bases, it is known that many true facts are not present, because the knowledge base is not necessarily *complete* [Galárraga, Razniewski, Amarilli, and Suchanek 2017]: the missing facts can include very likely facts, even facts that can be logically deduced from the existing ones.

Practical implementations. In this thesis we have not investigated practical implementations of the upper bounds we have derived. We discuss this matter here. First, it is debatable whether the tractable classes we have identified in Chapter 2 yield interesting tractable cases for practical applications. The settings of Propositions 2.3.6, 2.5.3, and 2.4.10 may look restrictive, as both labels and branching are important features of real-world database instances, though some situations may involve unlabeled treelike instances, or labeled words. The setting of Proposition 2.4.9 may be richer, and is reminiscent of probabilistic XML [Kimelfeld and Senellart 2013]: the instance is a labeled (downward) tree, while the query is a path evaluated on that tree.

Concerning our work on CFG-Datalog, we have good hopes that this approach can give efficient results in practice, in part from our experiments with a preliminary provenance prototype [Monet 2016; Amarilli, Maniu, and Monet 2016]. Optimizations are possible, for instance by not representing the full automata but building them on the fly when needed in query evaluation. However, we think that implementing our algorithms for CFG-Datalog would be quite difficult, given how complex the automata construction is. Another promising direction supported by experiments, to deal with real-world datasets that are not treelike, is to use partial tree decompositions [Maniu, Cheng, and Senellart 2017; Amarilli, Maniu, and Monet 2016].

Finally, we think our algorithm from Theorem 4.3.2 could easily be implemented and could be useful to compute the probability of Boolean circuits of low treewidth.

Appendix: résumé en français

This appendix is a summary in French of my PhD thesis. It is a translation of the General Introduction chapter and contains no added information.

Cet appendice est un résumé en français de ma thèse. Il s'agit d'une traduction du chapitre General Introduction qui n'apporte pas de nouvelles informations.

Le besoin de stocker, accéder et interroger des données a donné naissance au domaine de la recherche en bases de données. Les fondations de ce domaine sont sans doute à chercher dans le modèle relationnel, introduit par Edgar Codd à la fin des années soixante [Codd 1970] et qui se base sur le formalisme alors bien établi de la logique du premier ordre. Le modèle relationnel doit son succès à sa généricité : il offre une manière abstraite d'envisager les données, ce qui permet son utilisation dans de nombreux contextes. Ce succès peut s'observer à travers l'utilisation quotidienne de systèmes de gestion de bases de données (SGBDs) tels qu'Oracle, MySQL, PostgreSQL, et bien d'autres encore, partout dans le monde.

La recherche traditionnelle en bases de données fait souvent l'hypothèse que les données sont fiables et complètes. Pourtant, dans de nombreuses situations de la vie réelle, les données sont par nature incertaines. Cette incertitude peut prendre des formes diverses. Par exemple, quand les données sont extraites automatiquement à partir de pages web arbitraires, de l'incertitude est générée à cause de l'ambigüité inhérente aux méthodes d'analyse du langage naturel. Dans les systèmes de surveillance routière, l'incertitude peut venir du manque d'informations récentes sur le trafic [Hua et Pei 2010] : y a-t-il un embouteillage ? Y a-t-il une déviation ? En sciences expérimentales, des erreurs de mesures peuvent s'introduire dans des données, qui sont souvent collectées par des capteurs à la précision imparfaite [Asthana, King, Gibbons et Roth 2004]. Même quand des données nettes peuvent être obtenues, il se pourrait que nous ne fassions pas confiance à ceux qui les ont récoltées, ou à la manière dont elles nous ont été envoyées. Interroger ces données sans prendre en compte cette incertitude peut mener à des réponses incorrectes. Pour certains de ces scénarios, des algorithmes spécifiques ont été développés pour gérer cette incertitude, mais ces algorithmes dépendent en grande partie du domaine d'application et ne forment ainsi pas un cadre unifié. Pour tenter de capturer l'incertitude des données de manière générique, les SGBDs utilisent souvent la notion de NULLs. Cependant, les NULLs se limitent à représenter des données manquantes ou inconnues, et ne peuvent certainement pas représenter de l'incertitude *quantitative* à propos des données.

Pour répondre à ce besoin, la notion de base de données probabilistes a été développée [Suciu, Olteanu, Ré et Koch 2011]. L'objectif est ici de capturer de manière abstraite l'incertitude des données et d'être capable de raisonner sur des données incertaines, à l'instar du modèle relationnel, qui avait été introduit pour travailler de manière générique sur des données non probabilistes. Une première idée pour capturer l'incertitude des données serait de représenter explicitement tous les états possibles de l'information, c'est à dire toutes les bases de données possibles (appelées *mondes possibles*), et d'associer à chaque monde une certaine probabilité. La probabilité qu'une base de donnée satisfasse une requête booléenne donnée est

alors la somme des probabilités des mondes possibles qui satisfont la requête. C'est le problème d'*évaluation probabiliste de requêtes*, ou PQE : étant donné une requête booléenne Q et une base de données probabilistes D, calculer la probabilité que D satisfasse Q. La problème avec cette approche est qu'il y a généralement un trop grand nombre de mondes possibles, et donc qu'il n'est pas possible en pratique de représenter l'information et de l'interroger de la sorte. Néanmoins, nous pouvons *représenter* efficacement des données incertaines si nous faisons des hypothèses d'indépendance : par exemple, chaque fait (tuple) pourrait être annoté par une probabilité d'être présent ou absent, indépendamment des autres faits. Il s'agit là du modèle de base de données probabilistes dit *à tuples indépendants* (modèle TID). Des systèmes plus élaborés de représentation de données probabilistes existent [Barbará, Garcia-Molina et Porter 1992 ; Ré et Suciu 2007 ; Green et Tannen 2006 ; Huang, Antova, Koch et Olteanu 2009 ; Suciu, Olteanu, Ré et Koch 2011], mais nous ne travaillerons dans cette thèse qu'avec le modèle TID. En effet comme nous allons le voir, ce modèle à première vue simple pose déjà des problèmes difficiles à résoudre.

Et de fait : être capable de représenter efficacement des données incertaines (via l'hypothèse d'indépendance des tuples) ne permet pas pour autant *d'interroger* ces données efficacement. Par exemple, considérons la requête conjonctive suivante $q_{\text{hard}} : \exists xy\, R(x) \wedge S(x, y) \wedge T(y)$. Calculer la probabilité de cette requête sur des bases de données TID arbitraires est un problème intractable [Dalvi et Suciu 2007]. Spécifiquement, ce problème est #P-difficile, où #P est la classe de complexité des problèmes de comptage dont la solution peut s'exprimer comme le nombre de chemin acceptants d'une machine de Turing non déterministe qui termine en temps polynomial. Un exemple typique de problème #P-complet est #SAT, qui consiste à compter le nombre de valuations satisfaisant une formule propositionnelle donnée [Valiant 1979].

Pour s'attaquer à l'intractabilité de PQE sur des bases de données TID, trois approches générales ont été proposées. La première approche consiste à relâcher légèrement le problème, en demandant une *approximation* de la probabilité qu'on souhaiterait calculer. Il est en effet toujours possible de faire appel à la méthode de Monte Carlo [Fishman 1986 ; Ré, Dalvi et Suciu 2007 ; Jampani et al. 2008], qui fournit une *approximation additive* de la probabilité de la requête. Cette méthode n'est en revanche pas très utile quand les probabilités sont basses, puisque le temps de calcul de la méthode de Monte-Carlo est quadratique en la précision désirée. Comme nous ne nous intéresserons pas dans cette thèse aux méthodes d'approximation, concentrons nous sur les deux autres approches. Notre point de vue est qu'il faut d'abord comprendre les difficultés des méthodes exactes afin de déterminer si des méthodes d'approximation sont requises.

Les deux autres approches se basent sur une idée assez générale en informatique : lorsqu'on est face à un problème intractable, imposer des restrictions sur les entrées peut parfois ramener la complexité à la raison. Au delà de leur intérêt théorique, de telles restrictions peuvent aussi être utiles en pratique. En effet dans des problèmes concrets l'entrée n'est généralement pas complètement arbitraire et est au contraire souvent structurée d'une manière bien spécifique, ce qui peut ainsi permettre la conception d'algorithmes efficaces. Dans notre cas, la première approche exploite la structure de la *requête*, tandis que la seconde approche exploite celle des *données* :

- **Restreindre les requêtes.** Dalvi et Suciu [Dalvi et Suciu 2012] ont réussi à caractériser entièrement les unions de requêtes conjonctives (UCQs) pour lesquelles PQE est tractable sur des bases de données TID : une UCQ est soit *prudente* et PQE est en temps polynomial (PTIME), soit n'est pas prudente et PQE est #P-difficile. Malheureusement, il se trouve que de nombreuses requêtes, parfois mêmes très simple en apparence, ne sont pas prudentes ; c'est le cas par exemple de la requête q_{hard} vue plus haut.

- **Restreindre les données.** Des travaux récents de notre groupe [Amarilli, Bourhis et Senellart 2015] ont montré que lorsque les bases de données TID sont restreintes à être de *largeur d'arbre bornée*, il devient possible d'évaluer en temps linéaire n'importe quelle requête fixée Q exprimable en logique monadique du second ordre (MSO). MSO est une extension de la logique du premier ordre où la quantification d'ensembles d'éléments est autorisée (MSO peut donc entre autres exprimer n'importe quelle UCQ).

La largeur d'arbre est une mesure de théorie des graphes qui quantifie à quel point un graphe ressemble à un arbre. Borner la largeur d'arbre des bases de données est une technique connue pour assurer la faisabilité de nombreux problèmes qui sont NP-difficiles sur des bases de données arbitraires.

Ces deux approches s'intéressent à ce qui est communément appelé la *complexité en données*, c'est à dire la complexité du problème en tant que fonction de la taille de la base de données d'entrée, en considérant que la requête est fixée à l'avance. Les algorithmes ainsi conçus peuvent donc avoir un coût arbitrairement grand en la requête. Pourtant, en pratique les requêtes ne sont pas fixées mais sont aussi fournies en entrée par l'utilisateur, donc la complexité en la requête devrait aussi être raisonnable. Une meilleure mesure de la complexité de PQE est donc la *complexité combinée*, c'est à dire la complexité du problème en tant que fonction de la taille des données *et* de la requête. Pour cette mesure, l'algorithme de [Dalvi et Suciu 2012] est superexponentiel [Suciu, Olteanu, Ré et Koch 2011], et celui de [Amarilli, Bourhis et Senellart 2015] n'est même pas élémentaire élémentaire [Thatcher et Wright 1968 ; Meyer 1975]. Ainsi, même quand PQE est tractable en complexité en données, la tâche peut quand même être infaisable à cause de constances prohibitivement grandes qui dépendent de la requête.

À première vue, il semble déraisonnable de vouloir une complexité combinée tractable pour PQE puisque déjà dans le cas non probabiliste la complexité combinée n'est généralement pas tractable. Par exemple, évaluer une requête conjonctive booléenne (CQ) sur une base de données arbitraire est un problème NP-complet. Néanmoins, dans le cadre non probabiliste des recherches ont déjà été menées pour isoler des langages de requêtes pour lesquels l'évaluation est faisable en complexité combinée. Par exemple l'algorithme de Yannakakis peut évaluer des CQs α-acycliques sur des bases de données non probabilistes avec une complexité combinée tractable [Yannakakis 1981]. Pour ces raisons je pense qu'il est également important de mieux comprendre la complexité combinée de PQE, et si possible d'isoler des cas où le problème se comporte bien en complexité combinée. Cela motive la principale question qui sous-tend cette thèse : *pour quelles classes de requêtes et de bases de données le problème PQE a-t-il une complexité combinée raisonnable ?*

Afin de parvenir à une complexité combinée raisonnable, l'idée de cette thèse est d'imposer des restrictions sur à la fois les requêtes *et* sur les données. Nous soulignons

ici brièvement les contributions principales de cette thèse, avant de les présenter plus en détails dans le reste de ce résumé :

1. J'ai étudié PQE de requêtes conjonctives sur des signatures binaires, ce qui peut aussi se formuler comme un problème d'homomorphisme de graphes probabilistes. Nous restreignons les graphes des requêtes et des données à être des arbres et montrons l'impact sur la complexité combinée de nombreuses caractéristiques telles que la présence d'étiquettes sur les arrêtes, la capacité de branchement ou la connectivité. Ce travail débouche sur un paysage de résultats surprenants, mais où les cas tractables sont malheureusement très limités.

2. J'ai montré que, dans le cadre non probabiliste, l'évaluation d'un fragment spécifique de requêtes Datalog sur des bases de données de largeur d'arbre bornée peut se faire en temps linéaire en le produit de la taille des données et de la taille de la requête. Pour montrer ce résultat nous avons utilisé des techniques d'automates d'arbres et de calcul de provenance, et avons introduit un nouveau formalisme de représentation de la provenance, à base de circuits booléens cycliques. Notre résultat capture la tractabilité de langages de requêtes tels que les chemins d'expression régulière bidirectionnels (2RPQs) ou les CQs α-acycliques. De plus, nous pouvons connecter ces résultats de calcul de provenance au problème PQE, ainsi qu'expliqué dans le prochain point.

3. La troisième contribution est de montrer comment appliquer les résultats de la seconde contribution au problème PQE, en transformant les circuits cycliques construits en des circuits booléens ayant de fortes propriété sémantiques venant du domaine de la *compilation de connaissances*, comme par exemple les *circuits déterministes et décomposables en forme normale* (d-DNNFs). En utilisant les propriété sémantiques de ces circuits il devient alors facile de faire de l'évaluation probabiliste. Cela nous permet alors de résoudre PQE pour notre classe de requêtes Datalog (du point précédent) sur des données de largeur d'arbre bornée avec une complexité linéaire en les données et doublement exponentiel en la requête. Ce résultat contraste ainsi avec la complexité non élémentaire de [Amarilli, Bourhis et Senellart 2015] pour les requêtes MSO. Plus généralement, nous étudions les connections entre différentes classes de circuits habituellement considérés en compilation de connaissances, et nous montrons les limites de certaines de ces techniques en prouvant des bornes inférieures générales sur ces formalismes.

4. Sans lien avec la complexité combinée de PQE, j'ai aussi travaillé sur le compilation des requêtes *prudentes* de [Dalvi et Suciu 2012] en d-DNNFs et mené des expériences qui suggèrent qu'un certain sous-ensemble des requêtes prudentes peut en effet être efficacement compilé vers de tels circuits. Ayant décidé de ne pas incorporer ce travail dans ma thèse, ne ne le présenterai pas plus avant dans le reste du résumé.

Bien que j'aie décidé de présenter mon travail de recherche sous l'angle des bases de données probabilistes, l'étendue des contributions ne se limite pas à PQE. Par exemple, notre travail sur l'évaluation de Datalog sur des données arborescentes implique de nouveaux résultats sur la complexité combinée de l'évaluation non probabiliste et sur

le calcul de *provenance*. La troisième contribution a des applications dans le domaine de la compilation de connaissance, et la première contribution a des liens avec les *problèmes de satisfaction de contraintes*.

Nous présentons maintenant plus en détails nos contributions dans les parties suivantes de ce résumé.

Limites de la tractabilité combinée de PQE

Dans le chapitre 2 de la thèse nous étudions la complexité combinée de PQE pour des requêtes conjonctives (CQs) sur des bases de données TID. Ce langage de requêtes est l'un des plus simples qui soit habituellement utilisé sur des données relationnelles : ces requêtes correspondent en effet aux requêtes SQL construites avec les opérateurs SELECT, FROM, et WHERE. Notre objectif est d'isoler des cas fournissant des garanties de tractabilité assez fortes, à savoir, une complexité combinée en temps polynomial. Rappelons nous que cela n'est pas possible en général puisque la complexité en données de PQE peut être #P-difficile sur des bases de données TID arbitraires, et ce déjà à requête fixée. Nous allons donc devoir imposer des restrictions sur le problème.

La première restriction générale que nous allons imposer est de restreindre notre attention à des signatures binaires, c'est à dire où chaque relation est d'arité deux. Il s'agit d'une limitation assez naturelle qui est commune au monde des *bases de données orientées graphe* ou des *bases de connaissances*. Par exemple une base de connaissances enregistre de l'information sous forme de *triplets* tels que $(\text{Elvis}, \text{Aime}, \text{donuts})$. Ce triplet peut se voir comme un fait $\text{Aime}(\text{Elvis}, \text{donuts})$ sur le prédicat d'arité deux Aime. Cette manière de stocker l'information peut également se voir comme un graphe orienté où les arrêtes portent des étiquettes ; par exemple ici nous aurions deux nœuds, "Elvis" et "donuts", et une arrête allant du premier au deuxième nœud, portant l'étiquette "Aime". Par simplicité de présentation, nous définirons donc le problème en termes de graphes. Étant donnés un *graphe requête* G et un *graphe données* H, où chaque arrête est annotée par une étiquette, nous disons que H *satisfait* G lorsqu'il existe un *homomorphisme* de G vers H (c'est à dire intuitivement, une fonction des nœuds de G à ceux de H qui respecte la structure de G). Le problème devient ainsi le suivant : étant donnés un graphe requête et un graphe données probabiliste, où chaque arrête est annotée par une probabilité (et par une étiquette), nous devons déterminer la probabilité que le graphe données satisfasse le graphe requête, à savoir la somme des sous graphes de H qui satisfont G, en supposant l'indépendance entre les arrêtes probabilistes.

La seconde restriction générale sera d'imposer le graphes des requêtes et des données à avoir la forme d'arbres. Plus formellement, nous les restreindrons à être des *polyarbres*, c'est à dire des graphes dirigés dont les graphes non dirigés sous-jacent sont des arbres. Malheureusement comme nous allons le voir, cette restriction ne suffira pas à assurer la tractabilité, et nous étudierons alors l'effet de plusieurs caractéristiques additionnelles :

- *Étiquettes*, c'est à dire, si les arrêtes de la requête et des données peuvent être étiquetées par un alphabet fini comportant plus de deux éléments.
- *Déconnexion*, c'est à dire, permettre aux graphes d'être déconnectés ou non.

- *Branchement*, c'est à dire permettre aux graphes de brancher, ou au contraire leur imposer la forme de chemins.
- *Bidirectionnalité*, c'est à dire autoriser aux arrêtes une orientation arbitraire, plutôt que d'imposer à toutes les arrêtes d'être dans la même direction.

En conséquence, nous étudierons la complexité combinée de PQE pour des graphes *étiquetés* et pour des graphes *non étiquetés*, et quand la requête et les données font partie des classes suivantes, qui couvrent toutes les combinaisons possibles des caractéristiques ci-dessus : chemins unidirectionnels et bidirectionnels, arbres vers le bas et polyarbres, et unions de ces classes. Les figures 1 et 2 montrent des exemples de chemin unidirectionnel et bidirectionnel avec étiquettes, et d'arbre vers le bas et de polyarbre sans étiquettes.

Résultats. Nos résultats classifient entièrement la complexité combinée de PQE pour toutes les combinaisons considérées de restrictions de requêtes et données, dans le cas étiqueté et dans le cas non étiqueté. En particulier, nous avons identifié quatre cas maximaux incomparables de tractabilité, qui reflètent les divers compromis dans l'expressivité que l'on autorise dans la requête et dans les données :

- dans le cas non étiqueté, requêtes arbitraires sur des arbres vers le bas ;
- dans le cas étiqueté, requêtes chemins unidirectionnels sur des arbres vers le bas ;
- dans le cas étiqueté, requêtes connectées sur des chemins bidirectionnels ;
- dans le cas non étiqueté, requêtes arbres vers le bas sur des polyarbres ;

Ces résultats de tractabilité s'étendent tous à des graphes données déconnectés, et nous montrons que tous les autres cas non capturés par ces restrictions sont #P-difficiles. La (sinueuse) frontière de tractabilité peut être observée dans les tableaux 2.1, 2.2 et 2.3 aux pages 39, 43 et 51.

Malheureusement, ces résultats indiquent que la complexité combinée de PQE devient très rapidement intractable, pour la notion forte de tractabilité que nous avions considérée. Cependant cela n'exclue pas des résultats de tractabilité pour PQE sur des requêtes/données plus générales, si on considère une notion moins restrictive de tractabilité.

Tractabilité à paramètre fixé du calcul de provenance

Ayant réalisé que les cas où PQE est tractable en complexité combinée sont très limités, nous diminuons maintenant nos attentes vis-à-vis de la complexité en les requêtes. Nous aimerions trouver un langage de requêtes plus expressif et des classes de données plus générales pour lesquels PQE soit tractable en les données et "raisonnable" en la requête, cette asymétrie étant justifiée par le fait qu'en pratique la taille des données est beaucoup plus grande que celle de la requête. Pour atteindre notre but, nous nous concentrerons d'abord sur la complexité combinée du *calcul de*

provenance sur des données relationnelles non probabilistes. En effet, une approche bien connue pour évaluer des requêtes sur des bases de données probabilistes est *l'approche intensionnelle*. Cette approche consiste à calculer dans un premier temps une représentation du *lignage* (ou de la *provenance*) de la requête sur les données, qui intuitivement décrit comment le résultat de la requête dépend des tuples possibles de la base de donnée ; puis d'utiliser ce lignage pour calculer la probabilité de la requête, ce qui dans certains cas peut se faire efficacement. Ainsi dans le chapitre 3 de la thèse nous nous intéresserons au calcul de provenance, dans l'espoir de pouvoir utiliser plus tard les lignages obtenus pour faire du calcul de probabilités.

Notre point de départ est le travail de Courcelle, initialement présenté dans [Courcelle 1990] et résumé dans [Flum, Frick et Grohe 2002], qui prouve que pour n'importe quelle requête MSO fixée Q et constante k, il est possble d'évaluer Q en temps linéaire sur des bases de données de *largeur d'arbre* bornée par k. Intuitivement, les données de largeur d'arbre bornée peuvent être décomposées en plusieurs petits "morceaux" qui sont connectés entre eux en suivant une structure d'arbre (on parle alors de *décomposition arborescente*). Nous l'avions déjà mentionné : borner la largeur d'arbre des données est une méthode connue pour s'assurer que la complexité de nombreux problèmes (qui sont habituellement NP-difficiles) soit en temps polynomial. La preuve du résultat de Courcelle suit la structure suivante : d'abord, la requête Q est traduite en un *automate d'arbres* A, cette traduction ne dépendant que de la requête et de la constante k, mais pas des données. La base de données D de largeur d'arbre $\leqslant k$ est transformée en ce qu'on appelle un *encodage d'arbre* T, qui est simplement une décomposition arborescente encodée sur un alphabet fini de sorte à pouvoir être lue par un automate d'arbres. L'automate A est conçu de telle sorte que A accepte T si et seulement si D satisfait Q (ce qui s'écrit $D \models Q$), ce qui permet alors d'évaluer Q sur D en temps linéaire, en complexité en données. De plus, [Amarilli, Bourhis et Senellart 2015] montre comment il est possible d'utiliser cet automate pour construire des *circuits de provenance*, encore avec une complexité en données linéaire. Cependant, ces résultats ne nous disent rien sur la complexité *combinée*. En effet la complexité de calculer l'automate est non élémentaire en la requête MSO, ce qui fait que le procédé complet est limité par cette étape de traduction. D'où les questions que nous étudierons dans le chapitre 3 : *quelles sont les requêtes qui peuvent être traduites efficacement en automates ? Pour ces requêtes, peut-on efficacement faire du calcul de provenance ?*

Résultats. Pour tenter de mieux comprendre ce qui fait que la traduction en automates est inefficace, nous nous placerons dans cadre de la *complexité paramétrée* [Flum et Grohe 2006]. La théorie de la complexité paramétrée est une branche de la théorie de la complexité qui étudie la dépendance d'un paramètre d'entrée sur la complexité d'un problème. Cette théorie a été introduite dans les années quatre-vingt dix par Downey et Fellows (voir, par exemple, [Downey et Fellows 1992]). Dans notre cas, plutôt que de se restreindre à une classe fixe de requêtes "efficaces", nous étudierons des classes de requêtes dites *paramétrées*, c'est à dire que nous définirons une classe de requêtes pour chaque valeur distincte d'un certain paramètre. Nous ferons de plus l'hypothèse que la signature est fixée ; en particulier son arité sera constante. Ceci nous permettra de viser une complexité combinée raisonnable pour l'évaluation de nos requêtes, à savoir *tractable à paramètre fixé* avec un temps linéaire en le produit de la requête et des données, ce que nous appellerons

complexité FPT-bilinéaire.

La traduction de langages restreints de requêtes vers des automates d'arbres pour des données arborescentes a déjà été utilisé dans le contexte des *logiques gardées*, pour résoudre des problèmes de satisfiabilité [Benedikt, Bourhis et Vanden Boom 2016] et d'*implication* [Barceló, Romero et Vardi 2014]. Une technique pour la satisfiabilité consiste à montrer une propriété de *modèles arborescents*, ce qui permet de limiter la recherche de modèles à des modèles de faible largeur d'arbre, et ensuite de traduire la formule en un automate d'arbre et de tester si le langage reconnu par cet automate est vide ou non. Inspirés par ces fragments logiques, nous définissons le langage de *Datalog à frontière clique-gardée* (CFG Datalog), et mettons au point un algorithme FPT-linéaire qui effectue la traduction en automates, paramétrisé par la taille maximale des règles du programme CFG-Datalog. Ceci implique une complexité combinée FPT-bilinéaire de l'évaluation de notre langage sur des données arborescentes (paramétrisé par la largeur d'arbre des données et par la taille maximale des règles du programme). Nous montrons comment ce résultat capture la tractabilité de langages de requêtes tels que les 2RPQs α-acycliques [Barceló 2013] ou encore les requêtes conjonctives α-acycliques. Les RPQs sont au cœur de la plupart des langages de requêtes pour les bases de données orientées graphe, tandis que les CQs α-acycliques [Yannakakis 1981] sont la classe principale de CQs pour laquelle la complexité combinée de l'évaluation sur des données arbitraires est tractable.

À la place des automates d'arbres *descendant* utilisés dans [Courcelle 1990 ; Flum, Frick et Grohe 2002], et plus tard dans [Amarilli, Bourhis et Senellart 2015] (qui étend le résultat de Courcelle à PQE, mais toujours en complexité en données), nous utilisons le formalisme des automates d'arbres *alternants bidirectionnels*. Ces automates sont plus succincts que les automates d'arbres descendants, ce qui permet une traduction plus efficace ; et de fait, nous prouvons que les automates d'arbres descendants ne sont déjà pas assez succincts pour traduire efficacement les CQs α-acycliques.

Nous pouvons maintenant utiliser notre langage de CFG-Datalog pour comprendre ce qui rend la traduction des requêtes conjonctives en automates inefficace. Pour les requêtes conjonctives, nous montrons que la largeur d'arbre de la requête n'est pas le bon paramètre à borner pour assurer une traduction efficace, même en utilisant les automates d'arbres les plus expressifs. En revanche, la traduction efficace de CFG-Datalog implique la tractabilité combinée de l'évaluation pour les CQs de largeur d'arbre bornée qui satisfont une propriété additionnelle (dans les décompositions arborescentes, les interfaces entre les "sacs" doivent être clique-gardés). Cette notion de décomposition arborescente se trouve en fait être la notion de *décomposition simpliciale*, précédemment étudiée par Tarjan dans [Tarjan 1985]. CFG-Datalog peut se comprendre comme une extension de ce fragment où l'on aurait rajouté de la disjonction, de la négation stratifiée et des points fixes inflationaires, tout en préservant la tractabilité combinée de l'évaluation.

Nous nous intéressons ensuite au calcul de provenance pour notre langage CFG-Datalog, toujours sur des données de largeur d'arbre bornée. Nous rappelons ici que [Amarilli, Bourhis et Senellart 2015] a montré comment utiliser les automates descendants de Courcelle pour calculer des circuits booléens de provenance. Malheureusement cette technique ne fonctionne pas telle quelle sur des automates d'arbres alternants bidirectionnels. Pour cette raison, nous introduisons la notion de *circuit cyclique de provenance*, que nous baptisons *cycluits*. Ces cycluits sont

facilement utilisables pour représenter la provenance d'automates alternants bidirectionnels, puisqu'ils traitent de façon naturelle à la fois la récursion et la traversée d'un encodage d'arbre dans plusieurs directions. De plus, nous pensons que cette généralisation naturelle des circuits booléens peut avoir un intérêt en soi, et il semblerait que ces circuits booléens cycliques n'aient jamais été étudiés en détail. Nous montrons ainsi comment ces cycluits peuvent être évalués en temps linéaire. Nous montrons ensuite que la provenance d'un automate d'arbres alternant bidirectionnel sur un arbre peut être représenté par un cycluit, et que la construction peut se faire en temps FPT-bilinéaire. Ceci généralise ainsi le résultat de [Amarilli, Bourhis et Senellart 2015] qui calculait des circuits (non cycliques) pour des automates d'arbres descendants

Cela implique qu'il est possible de calculer en temps FPT-bilinéaire la provenance d'un programme CFG-Datalog sur des données de l'argeur d'arbre bornée, sous la forme d'un cycluit (le tout étant paramétrisé par la largeur d'arbre des données et par la taille maximale des règles du programme). Nous montrerons ensuite comment utiliser ces cycluits pour faire de l'évaluation probabiliste, en suivant l'approche intensionnelle pour PQE.

Des cycluits aux d-DNNFs et bornes inférieures

Dans le chapitre 4 de la thèse nous étudions les connexions entre divers formalismes de représentation qui sont habituellement utilisés dans l'approche intensionnelle de PQE, et en montrerons les conséquences pour notre langage de CFG-Datalog du chapitre 3. Rappelons que l'approche intensionnelle consiste en deux étape. La première étape est de calculer une représentation du lignage (provenance) de la requête sur les données. La provenance d'une requête booléenne sur une base de donnée est une fonction booléenne ayant pour variables les tuples de la base de données et qui décrit comment la réponse à la requête dépend de ces tuples. Cette fonction booléenne peut être représentée par n'importe quel formalisme servant à représenter des fonctions booléennes : formules booléennes, circuits booléens, diagrammes de décision binaires, etc. La seconde étape est de calculer la probabilité de cette fonction booléenne, qui est par définition précisément la probabilité que la base de données probabilistes satisfasse la requête. On peut voir cette étape comme une version pondérée du *comptage de modèles* (ici, on compte le nombre de valuations qui satisfont la fonction booléenne). Afin que cette étape soit efficace, le formalisme de représentation utilisé ne peut pas être arbitraire. Le domaine de la *compilation de connaissances* étudie (entre autres) quels sont les formalismes qui permettent ce calcul de probabilité de manière efficace, quelles sont les propriétés entre ces formalismes et les liens qu'ils entretiennent. Ainsi pour évaluer des requêtes sur les bases de données probabilistes, nous pouvons utiliser des algorithmes de compilation de connaissance pour transformer des circuits (ou même les cycluits que nous avons calculé au chapitre 3) en des formalismes tractables ; inversement, des bornes inférieure en compilation de connaissances peuvent aider à identifier les limites de l'approche intensionnelle.

Dans cette partie de la thèse, nous étudions les liens entre deux sortes de classes de circuits tractables en compilation de connaissances : les *classes de largeur bornée*, spécifiquement, les circuits de largeur d'arbre ou de largeur de chemin bornée ; et les *classes structurées*, spécifiquement, les OBDDs (diagrammes de décision binaires ordonnés[Bryant 1992], qui suivent un ordre sur les variables) et les d-SDNNFs

(circuits structurés déterministes décomposables en forme normale [Pipatsrisawat et Darwiche 2008], qui suivent un *v-arbre*). Les circuits de largeur d'arbre bornée peuvent s'obtenir lorsque l'on fait de l'évaluation de requêtes [Jha, Olteanu et Suciu 2010 ; Amarilli, Bourhis et Senellart 2015], tandis que les OBDDs et d-DNNFs ont été étudiés pour trouver des caractérisations théoriques des langages de requêtes dont ils peuvent représenter la provenance [Jha et Suciu 2011]. Ces deux types de classes de circuits permettent de faire du calcul probabiliste : pour les classes de largeur bornée, en utilisant du *passage de messages* [Lauritzen et Spiegelhalter 1988], en temps linéaire en le circuit et exponentiel en la largeur d'arbre ; pour les circuits structurés, en temps linéaire de par la définition de ces classes [Darwiche 2001]. D'où la question que nous étudierons dans le chapitre 4 : *peut-on compiler efficacement les classes de largeur bornée vers des classes structurées ?*

Résultats. Nous étudions d'abord comment effectuer cette transformation, et montrons donc des *bornes supérieures*. Des travaux existants ayant déjà étudié la compilation de circuits de largeur d'arbre bornée vers des OBDDs [Jha et Suciu 2012 ; Amarilli, Bourhis et Senellart 2016], nous nous tournons vers la compilation de ces circuits en d-SDNNFs. Nous montrons comment transformer un circuit booléen C de largeur d'arbre bornée par k en une d-SDNNF équivalent à C en temps $O(|C| \times f(k))$, où f est simplement exponentiel. Ceci nous permet d'être compétitifs avec la technique de passage de messages (qui elle aussi est simplement exponentielle en k), tout en proposant une technique plus modulaire. Au delà de l'évaluation probabiliste, notre résultat implique que toutes les tâches qui sont efficaces sur des d-SDNNFs le sont aussi sur des circuits de largeur d'arbre bornée (par exemple, l'énumération [Amarilli, Bourhis, Jachiet et Mengel 2017] ou l'inférence de la valuation satisfaisante la plus probable [Fierens et al. 2015]).

Nous pouvons alors utiliser cette construction pour étendre la tractabilité combinée de l'évaluation de CFG-Datalog sur des bases de donnée (non probabilistes) arborescentes au cas probabiliste (PQE). En effet, la largeur d'arbre du cycluit de provenance construit est linéaire en la taille du programme Datalog. Seulement, nous ne pouvons pas appliquer directement notre transformation (qui va des circuits de largeur d'arbre bornée aux d-SDNNFs) puisqu'ici nous avons à faire à des cycluits. La première étape est donc de transformer ce cycluit en un circuit, tout en préservant une borne sur la largeur d'arbre du circuit résultant. Nous montrons comment un cycluit de largeur d'arbre k peut être converti en un circuit équivalent et dont la largeur d'arbre est simplement exponentielle en k. Nous pouvons alors appliquer ces deux constructions successivement, ce qui permet de calculer une d-SDNNF qui représente la provenance d'un programme CFG-Datalog Q de taille de règles bornée sur une base de donnée D de largeur d'arbre bornée, le tout avec une complexité linéaire en la base de données et doublement exponentielle en la requête Q. Cette d-SDNNF nous permet ensuite de résoudre PQE pour Q sur D, et ce avec la même complexité. Bien que non polynomiale, nous considérons que la complexité en la requête est assez raisonnable, étant donné que notre langage CFG-Datalog est plutôt expressif (en tous cas, bien plus que les langages très limités de requêtes conjonctives que l'on a étudié au chapitre 2).

Nous montrons ensuite des *bornes inférieures* sur la compilation des classes de largeur bornée vers les classes structurées. Nos bornes fonctionnent déjà sur des formalismes plus faibles que les circuits, à savoir les *formules en forme normale*

conjonctive (CNF) monotones et en *forme normale disjonctive* (DNF) monotones de largeur d'arbre bornée. Nous connectons la largeur de chemin des CNFs/DNFs à la taille minimale de leur représentation sous forme d'OBDDs en prouvant que n'importe quel OBDD représentant une CNF ou DNF monotone doit être de taille au moins exponentielle en la largeur d'arbre de la formule (à arité (taille maximale des clauses de la formule) et degré (nombre maximal d'occurrences d'une variable) bornés). Parce qu'il s'applique à *n'importe quelle* CNF/DNF monotone, notre résultat généralise plusieurs bornes inférieures existantes en compilation de connaissances qui séparent exponentiellement les CNFs des OBDDs, par exemple [Devadas 1993] et [Bova et Slivovsky 2017, Theorem 19]. Nous prouvons de même une borne analogue pour la largeur d'arbre et les (d)-SDNNFs (encore en supposant que l'arité et le degré des formules sont bornés) : n'importe quelle d-SDNNF (resp., SDNNF) représentant une DNF (resp., CNF) monotone doit être de taille au moins exponentielle en la largeur d'arbre de la formule.

Pour prouver nos bornes inférieures, nous reformulons la largeur d'arbre et de chemin en des nouvelles notions, que nous nommons *pathsplitwidth* et *treesplitwidth*, et qui intuitivement mesurent la performance d'un ordre sur les variables ou d'un v-arbre. Nous utilisons également la notion de *dncpi-sets* introduite dans [Amarilli, Bourhis et Senellart 2016 ; Amarilli 2016], ainsi que des techniques récentes de complexité de communication développées dans [Bova, Capelli, Mengel et Slivovsky 2016].

Nous appliquons ensuite nos bornes inférieures à l'approche intensionnelle de PQE. Nous réutilisons la notion de *requêtes intriquées* de [Amarilli, Bourhis et Senellart 2016] et montrons qu'une d-SDNNF représentant la provenance d'une requête intriquée sur *n'importe* quelle base de données D doit être de taille au moins exponentielle en la largeur d'arbre de D. Ce résultat généralise celui de [Amarilli, Bourhis et Senellart 2016], qui lui était montré pour des OBDDs. Ce résultat montre que, en arité-deux et sous des hypothèses de constructibilité, la largeur d'arbre est *la* bonne notion sur les données pour s'assurer que n'importe quelle requête MSO ait des lignages représentables succinctement par des d-SDNNFs.

Structure de la thèse

Nous commençons par des préliminaires techniques dans le chapitre 1. Nous nous attaquons ensuite au contenu propre de la thèse, en exposant dans le chapitre 2 notre travail sur la complexité combinée du problème d'homomorphisme de graphes probabilistes. Nous continuons, dans le chapitre 3, avec l'évaluation de programmes CFG-Datalog sur des bases de données non probabilistes de largeur d'arbre bornée. Enfin, nous montrons dans le chapitre 4 comment étendre ces résultats à de l'évaluation probabiliste, et y prouvons des bornes inférieures sur des formalismes de compilation de connaissances. Puis nous concluons.

Self-References

Peer-Reviewed Journal Articles

Antoine Amarilli, Pierre Bourhis, Mikaël Monet, and Pierre Senellart (2018). "Evaluating Datalog via Tree Automata and Cycluits". In: *Theory of Computing Systems* (cit. on p. 57).

Peer-Reviewed Conference Articles

Antoine Amarilli, Pierre Bourhis, Mikaël Monet, and Pierre Senellart (2017). "Combined Tractability of Query Evaluation via Tree Automata and Cycluits". In: *ICDT* (cit. on pp. 57, 64, 107, 108, 113, 123).

Antoine Amarilli, Silviu Maniu, and Mikaël Monet (2016). "Challenges for Efficient Query Evaluation on Structured Probabilistic Data". In: *SUM* (cit. on p. 144).

Antoine Amarilli, Mikaël Monet, and Pierre Senellart (2017). "Conjunctive Queries on Probabilistic Graphs: Combined Complexity". In: *PODS* (cit. on p. 31).

Antoine Amarilli, Mikaël Monet, and Pierre Senellart (2018). "Connecting Width and Structure in Knowledge Compilation". In: *ICDT* (cit. on p. 107).

Peer-Reviewed Workshop Articles

Mikaël Monet (2016). "Probabilistic Evaluation of Expressive Queries on Bounded-Treewidth Instances". In: *SIGMOD/PODS PhD Symposium* (cit. on p. 144).

Mikaël Monet and Dan Olteanu (2018). "Towards Deterministic Decomposable Circuits for Safe Queries". In: *AMW* (cit. on pp. 12, 141).

Other References

Serge Abiteboul, Richard Hull, and Victor Vianu (1995). *Foundations of Databases*. Addison-Wesley (cit. on pp. 15, 16, 31, 57, 74, 98).

Alfred V. Aho and John E. Hopcroft (1974). *The Design and Analysis of Computer Algorithms*. Addison-Wesley (cit. on p. 19).

Noga Alon, Raphael Yuster, and Uri Zwick (1997). "Finding and Counting Given Length Cycles". In: *Algorithmica* (cit. on p. 59).

Antoine Amarilli (2014). "The Possibility Problem for Probabilistic XML". In: *AMW* (cit. on p. 143).

Antoine Amarilli (2016). "Leveraging the Structure of Uncertain Data". PhD thesis. Télécom ParisTech (cit. on pp. 10, 25, 89, 110, 125–127, 155).

Antoine Amarilli, Pierre Bourhis, Louis Jachiet, and Stefan Mengel (2017). "A Circuit-Based Approach to Efficient Enumeration". In: *ICALP* (cit. on pp. 9, 28, 108, 113, 154).

Antoine Amarilli, Pierre Bourhis, and Pierre Senellart (2015). "Provenance Circuits for Trees and Treelike Instances". In: *ICALP* (cit. on pp. 2–4, 6–9, 24, 25, 27, 31, 33, 51–54, 59, 61, 80, 85, 89, 107, 109, 112, 113, 142, 144, 147, 148, 151–154).

Antoine Amarilli, Pierre Bourhis, and Pierre Senellart (2016). "Tractable Lineages on Treelike Instances: Limits and Extensions". In: *PODS* (cit. on pp. 9, 10, 31, 51, 52, 54, 108, 110, 125, 126, 133–135, 154, 155).

Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski (1987). "Complexity of Finding Embeddings in a k-tree". In: *SIAM Journal on Algebraic and Discrete Methods* (cit. on p. 23).

Chandra K. Ashok and Moshe Y. Vardi (1985). "The Implication Problem for Functional and Inclusion Dependencies is Undecidable". In: *SIAM Journal on Computing* (cit. on p. 74).

Saurabh Asthana, Oliver D. King, Francis D. Gibbons, and Frederick P. Roth (2004). "Predicting Protein Complex Membership Using Probabilistic Network Reliability". In: *Genome Research* (cit. on pp. 1, 145).

Gilles Audemard and Laurent Simon (2009). "Predicting Learnt Clauses Quality in Modern SAT Solvers". In: *IJCAI* (cit. on p. 11).

Vince Bárány, Balder ten Cate, and Martin Otto (2012). "Queries with Guarded Negation". In: *PVLDB* (cit. on p. 74).

Vince Bárány, Balder ten Cate, and Luc Segoufin (2015). "Guarded Negation". In: *Journal of the ACM* (cit. on pp. 73, 74).

Daniel Barbará, Hector Garcia-Molina, and Daryl Porter (1992). "The Management of Probabilistic Data". In: *IEEE Transactions on Knowledge and Data Engineering* (cit. on pp. 1, 18, 146).

Pablo Barceló (2013). "Querying Graph Databases". In: *PODS* (cit. on pp. 7, 17, 58, 60, 76, 152).

Pablo Barceló, Miguel Romero, and Moshe Y. Vardi (2014). "Does Query Evaluation Tractability Help Query Containment?" In: *PODS* (cit. on pp. 7, 58, 61, 76, 152).

Paul Beame, Jerry Li, Sudeepa Roy, and Dan Suciu (2017). "Exact Model Counting of Query Expressions: Limitations of Propositional Methods". In: *ACM Transactions on Database Systems* (cit. on p. 11).

Paul Beame and Vincent Liew (2015). "New Limits for Knowledge Compilation and Applications to Exact Model Counting". In: *UAI* (cit. on p. 109).

Paul Beame, Guy Van den Broeck, Eric Gribkoff, and Dan Suciu (2015). "Symmetric Weighted First-order Model Counting". In: *PODS* (cit. on pp. 142, 144).

Michael Benedikt, Pierre Bourhis, and Pierre Senellart (2012). "Monadic Datalog Containment". In: *ICALP* (cit. on p. 64).

Michael Benedikt, Pierre Bourhis, and Michael Vanden Boom (2016). "A Step Up in Expressiveness of Decidable Fixpoint Logics". In: *LICS* (cit. on pp. 7, 58, 74, 152).

Michael Benedikt, Balder ten Cate, and Michael Vanden Boom (2014). "Effective Interpolation and Preservation in Guarded Logics". In: *LICS* (cit. on pp. 73, 74).

Michael Benedikt and Georg Gottlob (2010). "The Impact of Virtual Views on Containment". In: *PVLDB* (cit. on p. 69).

Michael Benedikt and Christoph Koch (2009). "XPath Leashed". In: *ACM Computing Surveys* (cit. on p. 143).

Anne Berry, Romain Pogorelcnik, and Geneviève Simonet (2010). "An Introduction to Clique Minimal Separator Decomposition". In: *Algorithms* (cit. on p. 65).

Dietmar Berwanger and Erich Grädel (2001). "Games and Model Checking for Guarded Logics". In: *LPAR* (cit. on pp. 60, 61, 70, 74).

Jean-Camille Birget (1993). "State-Complexity of Finite-State Devices, State Compressibility and Incompressibility"". In: *Mathematical systems theory* (cit. on p. 63).

Hans L Bodlaender (1996). "A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth". In: *SIAM Journal of Computing* (cit. on pp. 23, 110).

Hans L. Bodlaender and Arie M. C. A. Koster (2010). "Treewidth Computations I. Upper Bounds". In: *Information and Computation* (cit. on p. 98).

Simone Bova (2016). "SDDs are Exponentially More Succinct than OBDDs". In: *AAAI* (cit. on p. 143).

Simone Bova, Florent Capelli, Stefan Mengel, and Friedrich Slivovsky (2015). "A Strongly Exponential Separation of DNNFs from CNF Formulas". In: *CoRR* (cit. on p. 124).

Simone Bova, Florent Capelli, Stefan Mengel, and Friedrich Slivovsky (2016). "Knowledge Compilation Meets Communication Complexity". In: *IJCAI* (cit. on pp. 10, 109, 110, 130–132, 155).

Simone Bova and Friedrich Slivovsky (2015). "On Compiling Structured CNFs to OBDDs". In: *CSR* (cit. on p. 129).

Simone Bova and Friedrich Slivovsky (2017). "On Compiling Structured CNFs to OBDDs". In: *Theoretical Computer Science* (cit. on pp. 10, 109, 110, 124, 126, 155).

Simone Bova and Stefan Szeider (2017). "Circuit Treewidth, Sentential Decision, and Query Compilation". In: *PODS* (cit. on pp. 11, 108, 111, 112, 134).

Johann Brault-Baron (2014). "Hypergraph Acyclicity Revisited". In: *ArXiv e-prints* (cit. on p. 47).

Johann Brault-Baron, Florent Capelli, and Stefan Mengel (2015). "Understanding Model Counting for β-Acyclic CNF-Formulas". In: *STACS* (cit. on pp. 6, 33, 47–49).

Randal E. Bryant (1992). "Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams". In: *ACM Computing Surveys* (cit. on pp. 9, 153).

Andrei A. Bulatov (2013). "The Complexity of the Counting Constraint Satisfaction Problem". In: *Journal of the ACM* (cit. on p. 32).

Thierry Cachat (2002). "Two-Way Tree Automata Solving Pushdown Games". In: *Automata Logics, and Infinite Games* (cit. on pp. 63, 79, 101).

Andrea Calí, Florent Capelli, and Igor Razgon (2017). "Non-FPT Lower Bounds for Structural Restrictions of Decision DNNF". In: *CoRR* (cit. on p. 109).

Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzeniri, and Moshe Y. Vardi (2000). "Containment of Conjunctive Regular Path Queries with Inverse". In: *KR* (cit. on pp. 17, 76).

Florent Capelli (2016). "Structural Restrictions of CNF-Formulas: Applications to Model Counting and Knowledge Compilation". PhD thesis. Université Paris-Diderot (cit. on pp. 109, 130).

Florent Capelli (2017). "Understanding the Complexity of #SAT Using Knowledge Compilation". In: *LICS* (cit. on pp. 109, 130).

Romain Daniel Cazé, Mark Humphries, and Boris Gutkin (2013). "Passive Dendrites Enable Single Neurons to Compute Linearly Non-separable Functions". In: *PLOS Computational Biology*. Code available at https://github.com/rcaze/PlosCB2013 (cit. on p. 11).

Ismail Ilkan Ceylan, Adnan Darwiche, and Guy Van den Broeck (2016). "Open-World Probabilistic Databases". In: *KR* (cit. on p. 31).

Chandra Chekuri and Julia Chuzhoy (2014). "Polynomial Bounds for the Grid-Minor Theorem". In: *STOC* (cit. on pp. 135, 143).

Edward F. Codd (1970). "A Relational Model of Data for Large Shared Data Banks". In: *Communications of the ACM* (cit. on pp. 1, 145).

Sara Cohen, Benny Kimelfeld, and Yehoshua Sagiv (2009). "Running Tree Automata on Probabilistic XML". In: *PODS* (cit. on pp. 32, 52, 143).

H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi (2007). *Tree Automata: Techniques and Applications* (cit. on pp. 63, 66, 78).

Bruno Courcelle (1990). "The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs". In: *Information and Computation* (cit. on pp. 6, 7, 57, 60, 151, 152).

Radu Curticapean and Daniel Marx (2014). "Complexity of Counting Subgraphs: Only the Boundedness of the Vertex-Cover-Number Counts". In: *FOCS* (cit. on p. 32).

Nilesh N. Dalvi and Dan Suciu (2007). "Efficient Query Evaluation on Probabilistic Databases". In: *VLDB Journal* (cit. on pp. 1, 2, 31, 35, 146).

Nilesh N. Dalvi and Dan Suciu (2012). "The Dichotomy of Probabilistic Inference for Unions of Conjunctive Queries". In: *Journal of the ACM* (cit. on pp. 2–4, 10, 11, 31, 143, 147, 148).

Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov (2001). "Complexity and expressive power of logic programming". In: *ACM Comput. Surv.* (Cit. on p. 83).

Adnan Darwiche (2001). "On the Tractable Counting of Theory Models and its Application to Truth Maintenance and Belief Revision". In: *Journal of Applied Non-Classical Logics* (cit. on pp. 9, 54, 107, 112, 123, 154).

Adnan Darwiche (2003). "A Differential Approach to Inference in Bayesian Networks". In: *Journal of the ACM* (cit. on p. 113).

Adnan Darwiche (2011). "SDD: A New Canonical Representation of Propositional Knowledge Bases". In: *IJCAI* (cit. on p. 143).

Daniel Deutch, Tova Milo, Sudeepa Roy, and Val Tannen (2014). "Circuits for Datalog Provenance." In: *ICDT* (cit. on pp. 27, 58, 80).

Srinivas Devadas (1993). "Comparing Two-Level and Ordered Binary Decision Diagram Representations of Logic Functions". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (cit. on pp. 10, 109, 155).

Reinhard Diestel (1989). "Simplicial Decompositions of Graphs: a Survey of Applications". In: *Discrete Mathematics* 75.1 (cit. on p. 64).

William F. Dowling and Jean H. Gallier (1984). "Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae". In: *J. Log. Program.* (Cit. on p. 83).

Rodney G. Downey and Michael R. Fellows (1992). "Fixed Parameter Tractability and Completeness". In: *Complexity Theory: Current Research* (cit. on pp. 7, 20, 151).

Feodor F. Dragan, Fedor V. Fomin, and Petr A. Golovach (2011). "Spanners in Sparse Graphs". In: *Journal of Computer and System Sciences* (cit. on p. 136).

Ronald Fagin (1983). "Degrees of Acyclicity for Hypergraphs and Relational Database Schemes". In: *Journal of the ACM* (cit. on p. 59).

Tomás Feder and Moshe Y. Vardi (1998). "The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study Through Datalog and Group Theory". In: *SIAM Journal on Computing* (cit. on p. 143).

Andrea Ferrara, Guoqiang Pan, and Moshe Y. Vardi (2005). "Treewidth in Verification: Local vs. Global". In: *LPAR* (cit. on p. 124).

Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt (2015). "Inference and Learning in Probabilistic Logic Programs Using Weighted Boolean Formulas". In: *Theory and Practice of Logic Programming* (cit. on pp. 9, 108, 113, 154).

George S. Fishman (1986). "A Comparison of Four Monte Carlo Methods for Estimating the Probability of s-t Connectedness". In: *IEEE Transactions on Reliability* 35.2 (cit. on pp. 2, 146).

Jörg Flum, Markus Frick, and Martin Grohe (2002). "Query Evaluation via Tree-Decompositions". In: *Journal of the ACM* (cit. on pp. 6, 7, 25, 59, 60, 63, 69, 151, 152).

Jörg Flum and M. Grohe (2006). *Parameterized Complexity Theory*. Springer (cit. on pp. 7, 20, 21, 151).

Luis Galárraga, Simon Razniewski, Antoine Amarilli, and Fabian M. Suchanek (2017). "Predicting Completeness in Knowledge Bases". In: *WSDM* (cit. on p. 144).

Fănică Gavril (1974). "The Intersection Graphs of Subtrees in Trees are Exactly the Chordal Graphs". In: *Journal of Combinatorial Theory* (cit. on pp. 65, 98).

Georg Gottlob, Erich Grädel, and Helmut Veith (2002). "Datalog LITE: a Deductive Query Language with Linear Time Model Checking". In: *ACM Transactions on Computational Logic* (cit. on p. 60).

Georg Gottlob, Gianluigi Greco, and Francesco Scarcello (2014). "Treewidth and Hypertree Width". In: *Tractability: Practical Approaches to Hard Problems*. Cambridge University Press. Chap. 1 (cit. on p. 59).

Georg Gottlob, Christoph Koch, and Klaus U. Schulz (2006). "Conjunctive Queries Over Trees". In: *Journal of the ACM* (cit. on pp. 33, 50).

Georg Gottlob, Nicola Leone, and Francesco Scarcello (2002). "Hypertree Decompositions and Tractable Queries". In: *Journal of Computer and System Sciences* (cit. on p. 59).

Georg Gottlob, Nicola Leone, and Francesco Scarcello (2003). "Robbers, Marshals, and Guards: Game Theoretic and Logical Characterizations of Hypertree Width". In: *Journal of Computer and System Sciences* (cit. on p. 60).

Georg Gottlob, Reinhard Pichler, and Fang Wei (2010). "Monadic Datalog Over Finite Structures of Bounded Treewidth". In: *ACM Transactions on Computational Logic* (cit. on pp. 57, 61, 76).

Erich Grädel (2002). "Guarded Fixed Point Logics and the Monadic Theory of Countable Trees". In: *Theoretical of Computer Science* (cit. on p. 74).

Todd J. Green and Val Tannen (2006). "Models for Incomplete and Probabilistic Information". In: *IIDB* (cit. on pp. 2, 18, 146).

Todd J Green, Grigoris Karvounarakis, and Val Tannen (2007). "Provenance Semirings". In: *PODS* (cit. on pp. 27, 58, 80, 143).

Martin Grohe (2007). "The Complexity of Homomorphism and Constraint Satisfaction Problems Seen from the Other Side". In: *Journal of the ACM* (cit. on p. 32).

Martin Grohe and Dániel Marx (2009). "On Tree Width, Bramble Size, and Expansion". In: *Journal of Combinatorial Theory* (cit. on p. 124).

Martin Grohe and Dániel Marx (2014). "Constraint Solving via Fractional Edge Covers". In: *TALG* (cit. on p. 59).

Wolfgang Gutjahr, Emo Welzl, and Gerhard Woeginger (1992). "Polynomial Graph-Colorings". In: *Discrete Applied Mathematics* (cit. on pp. 6, 33, 50).

Frank Harary and Allen J. Schwenk (1973). "The Number of Caterpillars". In: *Discrete Mathematics* (cit. on p. 142).

Ming Hua and Jian Pei (2010). "Probabilistic Path Queries in Road Networks: Traffic Uncertainty Aware Path Selection". In: *EDBT* (cit. on pp. 1, 145).

Jiewen Huang, Lyublena Antova, Christoph Koch, and Dan Olteanu (2009). "MayBMS: a Probabilistic Database Management System". In: *SIGMOD* (cit. on pp. 2, 18, 146).

Tomasz Imielinski and Witold Lipski Jr. (1984). "Incomplete Information in Relational Databases". In: *Journal of the ACM* (cit. on pp. 27, 80).

Ravi Jampani, Fei Xu, Mingxi Wu, Luis Leopoldo Perez, Christopher M. Jermaine, and Peter J. Haas (2008). "MCDB: a Monte Carlo Approach to Managing Uncertain Data". In: *SIGMOD* (cit. on pp. 2, 146).

Abhay Kumar Jha, Dan Olteanu, and Dan Suciu (2010). "Bridging the Gap Between Intensional and Extensional Query Evaluation in Probabilistic Databases". In: *EDBT* (cit. on pp. 9, 107, 154).

Abhay Kumar Jha and Dan Suciu (2011). "Knowledge Compilation Meets Database Theory: Compiling Queries to Decision Diagrams". In: *ICDT* (cit. on pp. 9, 107, 154).

Abhay Kumar Jha and Dan Suciu (2012). "On the Tractability of Query Compilation and Bounded Treewidth". In: *ICDT* (cit. on pp. 9, 107, 108, 134, 154).

Abhay Jha and Dan Suciu (2013). "Knowledge Compilation Meets Database Theory: Compiling Queries to Decision Diagrams". In: *Theory of Computing Systems* (cit. on p. 11).

Sanjeev Khanna, Sudeepa Roy, and Val Tannen (2011). "Queries with Difference on Probabilistic Databases". In: *PVLDB* (cit. on p. 37).

Benny Kimelfeld and Pierre Senellart (2013). "Probabilistic XML: Models and Complexity". In: *Advances in Probabilistic Databases for Uncertain Information Management* (cit. on pp. 32, 144).

Laks V. S. Lakshmanan, Nicola Leone, Robert B. Ross, and V. S. Subrahmanian (1997). "ProbView: A Flexible Probabilistic Database System". In: *ACM Transactions on Database Systems* (cit. on p. 1).

Steffen L. Lauritzen and David J. Spiegelhalter (1988). "Local Computations with Probabilities on Graphical Structures and their Application to Expert Systems". In: *Journal of the Royal Statistical Society* (cit. on pp. 9, 107, 112, 154).

Hanns-Georg Leimer (1993). "Optimal Decomposition by Clique Separators". In: *Discrete Mathematics* (cit. on p. 65).

Dirk Leinders, Maarten Marx, Jerzy Tyszkiewicz, and Jan Van den Bussche (2005). "The Semijoin Algebra and the Guarded Fragment". In: *Journal of Logic, Language and Information* (cit. on p. 59).

Sharad Malik (1993). "Analysis of Cyclic Combinational Circuits". In: *ICCAD* (cit. on p. 59).

Silviu Maniu, Reynold Cheng, and Pierre Senellart (2017). "An Indexing Framework for Queries on Probabilistic Graphs". In: *ACM Transactions on Database Systems* (cit. on p. 144).

Emily A. Marshall and David R. Wood (2014). "Circumference and Pathwidth of Highly Connected Graphs". In: *Journal of Graph Theory* (cit. on p. 109).

Alberto O. Mendelzon and Peter T. Wood (1989). "Finding Regular Simple Paths in Graph Databases". In: *VLDB* (cit. on p. 76).

Albert R. Meyer (1975). "Weak Monadic Second Order Theory of Succesor is not Elementary-Recursive". In: *Logic Colloquium* (cit. on pp. 3, 57, 60, 147).

John C. Mitchell (1983). "The Implication Problem for Functional and Inclusion Dependencies". In: *Information and Control* (cit. on p. 74).

Joakim Alme Nordstrand (2017). "Exploring Graph Parameters Similar to Tree-Width and Path-Width". MA thesis. University of Bergen (cit. on p. 125).

Shinsuke Odagiri and Hiroyuki Goto (2014). "On the Greatest Number of Paths and Maximal Paths for a Class of Directed Acyclic Graphs". In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* (cit. on pp. 6, 33, 40).

Knot Pipatsrisawat and Adnan Darwiche (2008). "New Compilation Languages Based on Structured Decomposability". In: *AAAI* (cit. on pp. 9, 28, 154).

Knot Pipatsrisawat and Adnan Darwiche (2010). "A Lower Bound on the Size of Decomposable Negation Normal Form". In: *AAAI* (cit. on pp. 109, 130, 132).

J. Scott Provan and Michael O. Ball (1983). "The Complexity of Counting Cuts and of Computing the Probability That a Graph is Connected". In: *SIAM Journal of Computing* (cit. on p. 43).

Igor Razgon (2014). "On OBDDs for CNFs of Bounded Treewidth". In: *KR* (cit. on pp. 109, 124).

Chris Ré, Nilesh N. Dalvi, and Dan Suciu (2007). "Efficient Top-k Query Evaluation on Probabilistic Data". In: *ICDE* (cit. on pp. 2, 146).

Christopher Ré and Dan Suciu (2007). "Materialized Views in Probabilistic Databases: For Information Exchange and Query Optimization". In: *PVLDB* (cit. on pp. 2, 18, 146).

Marc D. Riedel and Jehoshua Bruck (2012). "Cyclic Boolean Circuits". In: *Discrete Applied Mathematics* (cit. on pp. 59, 81, 123).

Neil Robertson and Paul D. Seymour (1984). "Graph Minors. III. Planar Tree-Width". In: *Journal of Combinatorial Theory* (cit. on p. 21).

Neil Robertson and Paul D. Seymour (1986). "Graph Minors. II. Algorithmic Aspects of Tree-Width". In: *Journal of Algorithms* (cit. on p. 57).

Neil Robertson and Paul D. Seymour (1991). "Graph Minors. X. Obstructions to Tree-Decomposition". In: *Journal of Combinatorial Theory* (cit. on p. 130).

Bernd Schröder (2016). *Ordered Sets. An Introduction with Connections from Combinatorics to Topology*. Birkhäuser (cit. on pp. 33, 40).

Alexander A. Sherstov (2014). "Communication Complexity Theory: Thirty-Five Years of Set Disjointness". In: *MFCS* (cit. on p. 133).

Richard Stanley (1997). *Enumerative Combinatorics*. Cambridge University Press (cit. on p. 40).

Tamon Stephen and Timothy Yusun (2014). "Counting Inequivalent Monotone Boolean Functions". In: *Discrete Applied Mathematics* (cit. on p. 11).

Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch (2011). *Probabilistic Databases*. Morgan & Claypool (cit. on pp. 1–3, 18, 31, 43, 46, 50, 51, 145–147).

Robert E. Tarjan (1972). "Depth-First Search and Linear Graph Algorithms". In: *SIAM Journal on Computing* (cit. on p. 84).

Robert E. Tarjan (1985). "Decomposition by Clique Separators". In: *Discrete Mathematics* 55.2 (cit. on pp. 8, 58, 152).

Robert E Tarjan and Mihalis Yannakakis (1984). "Simple Linear-Time Algorithms to Test Chordality of Graphs, Test Acyclicity of Hypergraphs, and Selectively Reduce Acyclic Hypergraphs". In: *SIAM Journal on computing* (cit. on p. 63).

Alfred Tarski (1955). "A Lattice-Theoretical Fixpoint Theorem and its Applications". In: *Pacific Journal of Mathematics* (cit. on p. 81).

James W. Thatcher and Jesse B. Wright (1968). "Generalized Finite Automata Theory with an Application to a Decision Problem of Second-Order Logic". In: *Mathematical Systems Theory* (cit. on pp. 3, 147).

Ken Thompson (1968). "Programming Techniques: Regular Expression Search Algorithm". In: *Communications of the ACM* (cit. on p. 76).

Leslie G. Valiant (1979). "The Complexity of Enumeration and Reliability Problems". In: *SIAM Journal on Computing* (cit. on pp. 2, 19, 146).

Moshe Y. Vardi (1982). "The Complexity of Relational Query Languages". In: *STOC* (cit. on p. 57).

Moshe Y. Vardi (1995). "On the Complexity of Bounded-Variable Queries". In: *PODS* (cit. on pp. 35, 60).

Martin Vatshelle (2012). "New Width Parameters of Graphs". PhD thesis. University of Bergen (cit. on p. 130).

Mihalis Yannakakis (1981). "Algorithms for Acyclic Database Schemes". In: *VLDB* (cit. on pp. 3, 7, 31, 57, 59, 147, 152).

université PARIS-SACLAY | ÉCOLE DOCTORALE Sciences et technologies de l'information et de la communication (STIC)

Titre: Complexité combinée d'évaluation de requêtes sur des données probabilistes

Mots clés: Bases de données, Probabilités, Complexité combinée

Résumé : L'évaluation de requêtes sur des données probabilistes (*probabilistic query evaluation* ou PQE) est généralement très coûteuse en ressources et ce même à requête fixée. Bien que certaines restrictions sur les requêtes et les données aient été proposées pour en diminuer la complexité, les résultats existants ne s'appliquent pas à la complexité *combinée*, c'est-à-dire quand la requête n'est pas fixe. Ma thèse s'intéresse à la question de déterminer pour quelles requêtes et données l'évaluation probabiliste est faisable en complexité combinée.

La première contribution de cette thèse est d'étudier PQE pour des requêtes conjonctives sur des schémas d'arité 2. Nous imposons que les requêtes et les données aient la forme d'arbres et montrons l'importance de diverses caractéristiques telles que la présence d'étiquettes sur les arêtes, les bifurcations ou la connectivité. Les restrictions imposées dans ce cadre sont assez sévères, mais la deuxième contribution de cette thèse montre que si l'on est prêts à augmenter la complexité en la requête, alors il devient possible d'évaluer un langage de requête plus expressif sur des données plus générales. Plus précisément, nous montrons que l'évaluation probabiliste d'un fragment particulier de Datalog sur des données de *largeur d'arbre* bornée peut s'effectuer en temps linéaire en les données et doublement exponentiel en la requête. Ce résultat est prouvé en utilisant des techniques d'automates d'arbres et de compilation de connaissances. La troisième contribution de ce travail est de montrer les limites de certaines de ces techniques, en prouvant des bornes inférieures générales sur la taille de formalismes de représentation utilisés en compilation de connaissances et en théorie des automates.

Title: Combined Complexity of Probabilistic Query Evaluation

Keywords: Databases, Probabilities, Combined Complexity

Abstract: Query evaluation over probabilistic databases (*probabilistic query evaluation* or PQE) is known to be intractable in many cases, even in data complexity, i.e., when the query is fixed. Although some restrictions of the queries and instances have been proposed to lower the complexity, these known tractable cases usually do not apply to combined complexity, i.e., when the query is not fixed. My thesis investigates the question of which queries and instances ensure the tractability of PQE in combined complexity.

My first contribution is to study PQE of conjunctive queries on binary signatures, which we rephrase as a probabilistic graph homomorphism problem. We restrict the query and instance graphs to be trees and show the impact on the combined complexity of diverse features such as edge labels, branching, or connectedness. While the restrictions imposed in this setting are quite severe, my second contribution shows that, if we are ready to increase the complexity in the query, then we can evaluate a much more expressive language on more general instances. Specifically, we show that PQE for a particular class of Datalog queries on instances of bounded *treewidth* can be solved with linear complexity in the instance and doubly exponential complexity in the query. To prove this result, we use techniques from tree automata and knowledge compilation. The third contribution is to show the limits of some of these techniques by proving general lower bounds on knowledge compilation and tree automata formalisms.