Dialog Between Logic, Trees and Circuits

Pierre Bourhis

Dialogue between Logic, Trees and Circuits
(Dialogue entre Logique, Arbres et Circuits)

présentée et soutenue publiquement le 18 Septembre 2024 par

Pierre Bourhis

Chargé de Recherche au CNRS

Ecole Doctorale MADIS

Centre de Recherche en Informatique, Signal et Automatique de Lille

pour obtenir le

Diplôme d'Habilitation à Diriger des Recherches
de l'Université de Lille

Spécialité Informatique

Composition du Jury

Garant	Lionel Seinturier	Professeur, Université de Lille
Président du Jury	Pierre Senellart	Professeur, ENS Université PSL
Rapporteurs	Phokion Kolatis	Distinguished Research Professor, University of California Santa Cruz
	Reinhart Pichtler	Professeur, TU Vienna
	Frank Wolter	Professeur, University of Liverpool
Examinateur	Serge Abiteboul	Directeur de recherche émérite, INRIA Paris
Examinatrice	Sophie Tison	Professeure émérite, Université de Lille

Abstract

Since my post-doctorate, I have mainly worked on the foundation of data and knowledge management. In my habilitation, I focus on some of my main contributions over the following questions: reasoning of logical formula over relational instance, reasoning and evaluating queries over trees, and reasoning over circuits.

My contributions are organised into three parts: a dialogue between logical reasoning and reasoning over trees; a dialogue between tree evaluation and querying circuit and finally a dialogue between the explanation of query evaluation and circuits.

The first part focuses on translating logical problems into reasoning problems over trees and vice versa. First, I present different reductions of reasoning on fixpoint logic into reasoning on tree automata in order to obtain some efficient upper bound over the logical reasoning. Secondly, I explain how the validity problem of a tree automata into a union of conjunctive queries can be used to prove different lower bounds on logical reasoning.

In the second part, I present an approach proposed by my co-authors and me to reduce different problems of enumeration of solutions of tree automata evaluation over trees into the same enumeration problems over a particular kind of circuit allowing to solve this problem efficiently.

Finally, I present different results on the computation of provenance for Datalog, a well-known recursive language for querying databases. First, I present an approach based on techniques presented in previous parts giving an FPT algorithm to compute efficiently the Boolean provenance for a subclass of Datalog programs. Then, I present a discussion about different semantics to compute the provenance for Datalog comparing them together and following some desired properties.

Résumé

Mon habilitation porte principalement sur les résultats fondamentaux que j'ai développé depuis mon post-doctorat. Ceux-ci se sont focalisés sur la relation entre différents questions: les raisonnements sur les formules logiques dans le cadre relationnel, le raisonnement et l'évaluation de requêtes sur les arbres et l'évaluation sur les circuit logiques.

Dans le cadre de mes travaux sur la gestion des connaissances, j'ai étudié le raisonnement sur de formules de logiques avec point fixes. J'ai travaillé sur la réduction des problèmes de raisonnements sur des instances relationnelles vers des problèmes de raisonnements sur les arbres et vice versa. Ces travaux ont porté sur la traduction de formule logique en automate d'arbres permettant d'obtenir des bornes supérieures sur les raisonnements de formules et sur la traduction du problème de validité d'un automate d'arbre au regard d'une requête conjonctive dans un problème de raisonnement sur la logique étudiée permettant d'obtenir des bornes inférieures sur ce dernier problème. En particulier, nous étudions une logique de point fixe appelée GNFP-UP et montrant les bornes inférieures pour l'inclusion de programmes Monadic Datalog qui était une question ouverte dans la communauté.

Le deuxième grand axe de recherche sur lequel j'ai travaillé est la relation entre les circuits logiques et l'évaluation de requêtes sur les arbres et les mots. Nous montrons que la trace de l'évaluation d'une requête MSO sur un arbre, appelée provenance peut être décrite au travers un circuit booléen possédant une structure particulière appelée d-DNNF de même que les réponses de cette requête. Au travers l'étude de ce type de circuit, nous avons pu redémontrer que l'énumération des réponses peut se faire avec un précalcul linéaire et un délai constant et que nous pouvions mettre à jour efficacement le circuit sous-jacent. Nous avons étendu notre approche pour énumérer les réponses suivant un ordre donné par une fonction d'agrégation sur les valeurs des réponses:

Pour finir, j'ai travaillé sur la notion de provenance et sa représentation en circuit. La notion de provenance en base de données est une méthode permettant de représenter et calculer différentes informations autour de l'évaluation des données. Une représentation par circuits permet de représenter efficacement tout un ensemble de provenances. Plusieurs questions se posent dans ce cadre, comment bien définir la notion de provenance, comment l'évaluer efficacement et dans quel contexte. Tout d'abord, nous avons proposé une approche FPT pour calculer la provenance booléenne pour un sous langage de Datalog sur une instance ICG en utilisant des techniques développées dans les autres parties. Nous investiguons aussi différentes sémantiques pour la provenance afin de comprendre les limitations. La définition classique de la provenance pour Datalog impose de raisonner sur un nombre infini de structures limitant son usage. Nous comparons ces différentes sémantiques entre elles mais aussi au regard de propriétés désirées.

Contents

Chapter 1

Introduction

The following document is a synthesis of my research activity since my PhD defence in February 2011.

After my PhD, I did a post-doctorate at Oxford University. I joined CNRS in October 2013 at LIFL which became CRIStAL in 2015. From 2013 to 2017, I was a member of the Links team and since 2017 I have been a member of the SPIRALS team. My research activities have focused on Foundation Data and Knowledge Management. My research was in particular motivated by Knowledge management on the web until 2017. Since I have arrived at SPIRALS, I have also been doing research in the management of knowledge inside applications and not only inside Knowledge management systems. This has led me to study different problems such as the security of knowledge, the explanation of queries as well as the logical reasoning in the development of applications. Moreover, I have started to apply my research into the domain of biocomputing and digital humanities: in literature, law and history.

My Habilitation will mainly summarize some of the theoretical results that I have developed since my postdoctorate. I focus on the study of the relation between the following questions: reasoning on fixpoint logical formula over relational data, reasoning and evaluation of query over trees and the evaluation of logical circuit. In particular, these papers have linked different fields such as Database Theory, Knowledge Representation and Formal Methods. My works were published and presented at the main conferences of these different domains.

1.1 Context

The complexity of data management has raised some peaks over the last decades with the proliferation of different sources of data, in particular through data accessible on the Web.

Due to this proliferation of sources and the amount of data that they generated, different problems have been raised: how to manage the heterogeneity of the data to query it efficiently; how to manage data which is less structured for example used in NoSQL paradigms such as JSON and Graph Databases and text; how to query efficiently data when data is incomplete and uncertain; how to query efficiently data when the number of answers is huge and more generally how to answer complex queries and how to understand how the data does satisfy the query. These questions have been intensively studied in different communities over the last decades in particular in the communities of Database Theory and Knowledge Representation.

A general approach to deal with these problems has been to use logical formulas. Logical formulas have been always at the core of data management in particular for relational databases. Hidden through the standard language SQL, it is the core of query languages for databases. Thanks to the results of Codd,

it is possible to transform any First-order-Logic into an algebraic relational expression that then can be optimized into an efficient execution plan using appropriate algorithms for each algebraic operator. More generally, logic has been at the core of many approaches to solve the problems expressed above.

The heterogeneity of the data has different reasons: (i) in the relational model, the schemas of the data are different; (ii) the data have different formats/representations. In the first case, the data is stored in different relational databases having different schemas. It is a classical problem as companies can have inside their organisation their data stored at different places and when data comes from different sources. A classical approach for dealing with data coming from different schemas is to create a new schema used as a reference. Each other schema is linked to this new schema by explaining how to transfer the data from a schema to a reference schema. The classical setting to describe such links between schema is based on particular First-Order formulas called *Tuple Generated Dependencies (TGDs)*. Thus, there are two manners to query the data over the reference schema: first the data from each source is transformed into data from the reference schema. It is important to notice that this transformation creates new data which were not in the initial source. This problem has been studied in the context of data exchange [KPT06]. It was established that there is a canonical instance that can be created through a procedure called the chase. This procedure has different variants that all create a canonical instance and all are called chase. The other approach for querying efficiently the reference schema is to rewrite the query into different queries over the different source schemas and to combine the answers of these queries.

The heterogeneity of data is also brought by different representations/formats of data such as trees with XML and JSON formats. The representation of data on a graph has become very popular with Graph Databases and RDF graphs. Query languages over data graphs have introduced or highlighted different new features such as regular path queries and more generally, recursive queries and finally ontologies. Understanding and managing these new representations and queries has been a challenge. In particular, mixing ontologies and recursivity is a complex problem that has brought a lot of attention in the last decades. Finally, managing text has brought a lot of attention through data extraction with a new formalism proposed by IBM: spanners [PFKK19]. Spanner is a query language to extract substrings from a text and to reason on them through (Datalog) rules.

When publishing information on the Web, it often happens that some of the data is missing/unknown. To avoid a complete lack of information about the missing data and to express the fact that data is missing, it has been proposed to express some knowledge about the missing/unknown data. This knowledge is often expressed through logical rules expressed in different languages like TGDs or Description Logic. In the context of missing or unknown data, the classical question is to query data by taking into account these missing data, which is called certain query answering. In the same manner as for the heterogeneity of schema, the missing data can take any possible value and therefore when querying data while taking into account missing data, the approach is to check if the query is satisfied by any instance containing the initial database and satisfying the rules describing the knowledge. Because the domain of the value is usually infinite, it is needed in principle to check an infinite number of instances. Thankfully, if the rules are expressed by TGDs and the query possibly recursive is closed under homomorphism, there exists a notion of canonical instance and there exists a procedure called chase to build this canonical instance. The problem of certain query answering can be reduced to the problem of evaluating the query over this canonical instance. Unfortunately, this canonical instance can be infinite following the properties of the rules and therefore, it cannot be built in practice. There exist different approaches to deal with this problem by representing finitely an infinite instance or rewriting the query into another query such that the evaluation of the last one is equivalent to a certain query answering the first one. Even though certain query answering over TGDs is undecidable, there

exist different dialects for which this task is computable. Different techniques are used to prove the decidability such that the finiteness of the chase the boundedness of the chase's treewidth. This last property leads to reducing the certain query answering problem to some reasoning over MSO formula over trees. The certain query answering problem can also be reduced to a classical query evaluation for another query.

Data on the Web is also uncertain. Different approaches have been proposed to deal with uncertainty: numerical ones assign scores/probabilities to each fact and then some score for the evaluation of the query is computed following the score on the facts. In the context, numerical score like for fuzzy logic, the score is computed by changing the classical operators as follows: $x \wedge y$ to $\min(x, y)$, $x \vee y$ to $\max(x, y)$ and $\neg x$ to $1 - x$. In the context of the probabilistic evaluation, the probabilistic values assigned to the fact of the database express the probability that a tuple appears in a possible world and the probability of the query evaluation is based on the probabilities of the possible worlds on which the query is satisfied. More recently, the numerical score for the contribution proposed in the artificial intelligence community such that Shapley value score. More generally, the problem of uncertainty implies a better understanding of the contribution of the tuples to the result of a query. The database community presented different frameworks under the name of provenance which describe such contribution of tuples through logical formula. Interestingly, [DFKM22, JS13] made some interesting connections between provenance and numerical approaches to deal with uncertainty.

Finally, the appearance of a massive amount of data implies revisiting some approaches to evaluate queries. It is well known that different classes of queries have different data complexity. Remember, data complexity is the complexity only depending on the size of the data and not on the size of the query. For example, general recursive queries such as Datalog queries are PTIME-complete when conjunctive queries are in AC-0 data complexity. Moreover, classical plan optimisation is principally focused on first-order queries and even more on conjunctive queries. Therefore, understanding when a recursive query can be rewritten into a first-order query can lead to an important performance optimisation of the query. In the same manner, the combined complexity of conjunctive query has been deeply studied and different notions of decompositions over the conjunctive formula to obtain a fine-grained combined complexity. For example, acyclic queries can be evaluated in polynomial time in combined complexity as the opposite of NP-complete complexity in general. Therefore, it is interesting to understand when a query can be rewritten into a simpler query.

The query evaluation of an enormous amount of data implies that the number of answers can be too voluminous even though the user does not need all the answers. A classical manner to avoid this is to limit the number to a certain amount for example by using the SQL command LIMIT. Another approach for dealing with a huge amount of answers is to provide a partial set of answers and then deliver new partial sets following the user's wish for results delivered by a search engine.

1.2 Challenges

1.2.1 Reasoning on Recursive Logical Formulae

During the previous section, we noticed that several important questions and tasks can be reduced to some reasoning over logical formulas more particularly the satisfiability of the formula, ie knowing if a model is satisfying this formula. For example, the problem of containment of a query Q_1 into another Q_2 under the constraints Σ can be reduced to the problem of satisfying the formula $Q_1 \wedge \neg Q_2 \wedge \Sigma$, the problem of certain query answering for a set of rules Σ, a query Q and instance I can be reduced to the problem of the satisfiability of $Q_I \wedge \Sigma \wedge \neg Q$ where Q_I is the formula stating that the facts in

I are true. Even though all these problems are undecidable in the general case, there has been a lot of progress in finding subclasses of queries and rules to obtain decidability results. In parallel with these results on specific problems, different decidable fixpoint logics such as GF, GFP, GNF, and GNFP have been proposed. Often the specific cases of decidability problems on certain query answers can be reduced to the satisfiability of these logics.

During the last decade, containment of recursive queries possible under constraints, certain query answering for conjunctive queries or recursive queries have brought a lot of attention. Even though recursive query languages such as Datalog were proposed in the 80s, we have observed a new wave of popularity over the last two decades. Indeed, querying graph databases may require looking for nodes that are connected through unbounded paths having some properties. These kinds of queries cannot be expressed by First Order logic but by Fixpoint logic. More interestingly, the basic queries such that conjunctive regular path queries cannot be captured by classical recursive languages having decidability containment such as Monadic Datalog or Fixpoint Logics having their satisfiability decidable such as GNFP. It is interesting to understand if there is a logic or recursive query language that can express conjunctive regular path queries.

Several classical problems such as the containment of decidable subclasses of datalog programs, and certain query answering for a union of conjunctive queries under different classes of rules do not have their exact complexity known. For example, the question of containment of monadic datalog programs is a problem known to be in 2EXPTIME since the end of the 80s. However, for more than two decades, the lower bound of this problem was not known. In the same manner, several subclasses of TGDs for which certain query answering did not have their precise complexity known.

1.2.2 Efficient Complexity for Enumeration

One of the trendy studied techniques to deal with a massive amount of answers is the framework of enumeration. This framework proposes to operate an efficient preprocessing that computes an efficient representation of the answers to the query and a method to enumerate one by one the answers computed in the preprocessing via an efficient method. The manner to measure the complexity of such an approach is to measure first the complexity of the preprocessing and the complexity to compute the next answer from the previous answer and the preprocessed structure. A naive approach is to compute classically the set of answers of then enumerate them one by one. In this context, the complexity of the preprocessing will be the same as the classical computing evaluation of the query and the delay between two answers is constant in the size of the database. From this remark, we can notice that the enumeration process is in general a trade-off between the preprocessing and the delay. Previous works have focused on subclasses of queries for which the complexity of preprocessing is linear and the complexity of the delay is constant both in the size of the database. This question has been studied for different data representations: words, trees and relational and also for more complex queries such as queries with aggregation associating values to the answers and such as the enumeration following the rank given by the values. However, the understanding of maintaining the representation of the answers when the data is updated was an open question in the context of trees.

1.2.3 Efficient Complexity for dealing with explanation of queries

The problem of uncertain data has been dealt with different approaches some numerical associating confidence score or probabilities to the tuples or a more symbolic approach by using a formula representing the sets of tuples of the database contributing to the answers. This last notion is also called provenance in the database community but it could be named as lineage. There exist different notions of

provenance following the details of contributions that are wanted by the users. Interestingly, [GKT07] made some connections between the numerical evaluations and the symbolic approaches through the notion of semiring by showing that some confidence scores can computed as a specialisation of some more general specifications. Some other results [JS13] explored this connection showing that the notion of probabilistic evaluation can be efficient if the Boolean provenance of a query can as some particular form. This approach has made links between Database Theory and Knowledge Compilation. This notion of provenance through semi-ring and their representation via circuits has brought a lot of attention in the community and different questions of efficient computation and representation have been studied. This problem is in particular important in the context of recursive queries. Indeed, by using classical semi-ring In this context, classical semi-rings can be used but more specific semi-rings should be used. Different properties over the semi-ring have been proposed to obtain a well-defined and finite representation of semi-ring provenance over recursive queries [KNP$^+$24].

1.3 My contributions

1.3.1 Reasonning on Fixpoint Logical Formulae

My main contributions on reasoning on logical formulae are on recursive queries and fixpoint formulas such as containment of recursive queries, certain query answering for recursive queries or with rules having some recursive parts. These contributions can be classified in two parts: one on the increase of expressiveness of subclasses of recursive queries and fixpoint queries which have some decidable properties and another about establishing new lower bounds.

For the first part, [RK13] established a more expressive fragment of Datalog which the containment is decidable because belonging to the intersection of Datalog and MSO. Interestingly, this new fragment captures regular conjunctive queries. Unfortunately, its complexity established is elementary. In [BKR15], we present the exact complexity of the containment problem of queries of this language. We also extend this query language to capture decidable query language outside MSO with a similar complexity. Our results also allow us to capture conjunctive regular conjunctive queries and reproof classic results over them. By using the same approach, we extend GNFP to another decidable logics which capture conjunctive regular path queries. These results use a property called tree decomposition. Intuitively, most of the reasoning tasks over logical formulas are to construct an instance showing that the property over the formula is true. The tree decomposition technique/property is satisfied by a formula iff if there exists an instance satisfying the formula (model) then there exists a model with a tree shape. The tree-shape property allows us to reduce a problem from a general relational structure to an equivalent one over trees which is expressible in MSO. This property allows us to translate the problem of satisfiability over any instance to a satisfiability problem into an MSO formula over (infinite) trees. This problem is well-known to be decidable.

Thanks to this approach, many decidability results have been obtained, however, this approach does not provide in general the exact complexity of the problem. To obtain the exact complexity, I worked on different translation of logic formulas into tree automata

- for the containment of Datalog in the union of conjunctive queries to obtain better complexity for subclasses of Datalog queries [BBV16, BBV19] which are used in the context of querying data accessible through access patterns.
- for understanding the complexity highly expressible recursive queries expressible in Datalog called GQ [BKR15] and its generalisation for highly expressive fixpoint logics [BMMP16], [BL16].

These techniques are based on a new kind of tree automata called *pointed tree automata* which select and evaluate a tree automata starting from a particular node in the tree.

- for rewriting logical formulas into more efficient subclasses. For example, we proposed techniques [BBV17],[BBV19] to check if it is possible to rewrite a logical formula in GNPF using k variables into a GFP formula and propose a rewriting of the initial formula.

In another way, if some previous results had some well-known upper bounds as the containment of monadic datalog in a union of conjunctive queries [CGKV88], the lower bound of its problem was not known. We established a general framework by establishing new results for the question of the validity of a language described by a tree automata into a tree pattern query, i.e. checking that each tree satisfying the tree automata also satisfies the tree pattern query. We study this problem for different schemas that the tree pattern can use: CHILD-OR-SELF ... Thanks to these results, we reduce classical reasoning problems on fixpoint logics and subclasses to related validity problems for different class of schema and thus obtaining lower bounds for the different problems. The establishment of the results over trees is published in [BBGS20] and it was used in different contexts, in particular for closing the exact complexity of the containement of Monadic Datalog queries which was an open problem for 20 years in [BBS12].

Two of these results received prizes, [BBV17] received the best paper award for track B of ICALP 2017 and [BKR15] was a distinguished paper (honorary mention) at IJCAI 2015.

1.3.2 Efficient Compact representation for answers based on circuits applied to Enumeration

The second axe of research has been the link between logical circuits and evaluation of queries over trees and words. As presented before, Enumeration is a framework based on computing a compact representation of the answers of the query and then to enumerate the answers following a possible order or not. We have studied this problem in the context of MSO queries over trees and words. We introduce a new framework which represents the set of solutions into classes of logical circuits which are known thanks to the Knowledge Compilation community. We then studied the different problems on the circuit representation.

This approach started by defining a notion of the provenance of the evaluation of a MSO formula over trees [ABS15a] during the PhD of Antoine Amarilli whom I informally co-advised. These results show that Boolean provenance of MSO over trees can be computed in linear time in the size of the tree and that the computed circuit has a particular form called d-DNNF and therefore, different tasks such that probabilistic evaluation over trees can be done in linear time. This result was known but it showed the interest of looking at our approach.

Based on these results and by noticing that the provenance of the evaluation of the MSO formula Q can be seen as describing a compact representation of the set of answers of another MSO formula Q', we investigated the general problem of computing compact representation. Through the progress in our work, we proposed different equivalent presentations of circuits, d-DNNF, d-DNNF set circuits or d-DNNF multivalued circuits which are all computable in linear time in the size of the tree. We then studied the different problems that we were interested in: enumeration of answers, ranked enumeration and finally updating the compact representation of the study. Thanks to this circuit representation, we propose a novel approach to compute enumeration over MSO trees [ABJM17]. This result was proved before with two completely different approaches [Bag06, KS13]. Thanks to the links between the circuit structure and the tree structure, we were able to prove that we can efficiently update the circuit structure when the tree structure is updated [ABMN19b]. Some of these results rely on an unpublished

work [KMN22]. To finish our work on classical enumeration, we studied the problem of combined complexity: in [FRU$^+$18], an open question was asked about the combined complexity of enumeration on nondeterministic automata. The previous results were established when deterministic automata expressed queries. We proved that for words and trees, the complexity can be done in polynomial time in the size of the query and linear time in the data in [ABMN19a, ABMN21] and [ABMN19b].

Finally, we studied the problem of ranked enumeration over words and trees when different aggregation procedures for assigning values to the answers of the query. In [BGJR21], we study the rank enumeration for words when the value attached to the answers is obtained by aggregating values attached to the edges of the automata giving a value associated with the accepting run of the automata. In [ABCM24], we study aggregations of values associated with the nodes of trees belonging to the answers in the same setting as [DK19, DHK22, TGR22].

The paper [ABMN19a] got a SIGMOD Research Highlights in 2020.

1.3.3 Efficient Complexity for dealing with explanation of recursive queries

In the context of recursive queries, the problem of definition and representation of semi-ring provenance has been heavily studied to find a more compact [DMRT14] representation through circuits and different definitions of semi-ring for finite representations. In this context, we have proposed another representation called cycluit that can construct a more compact representation of a circuit of recursive queries for the boolean provenance. [ABMS19]. These cycluits can be seen as recursive circuits over the internal gates. Secondly, in [BBPT22a], we studied different propositions of computing the provenance based on execution semantics and we compare them together trough different properties that we introduce.

1.4 Other works

My full list of publications can be found in the appendix Section 5. Most of my other works have been on reasoning on different formula logics for different applications such as security and provenance. The line of work not presented in this manuscript is on Data-centric Workflow. This is a line of research that tries to model different applications through their transformation of data. It is in line with the work of active databases [Wid96], relational transducer [AVFY00] and relational machines [AVV97] developed in the 90s and the notion of business/tuple artefact [KV17] developed in 2000. In this context, I have worked on the notion of explanation in the context of different settings, distributed or centralized based on the tuple artefact framework. We established in [BDM16] a notion of provenance for the tuple-artifact approach and used it for different kinds of explanations. In [BDM20], we focus on a compact algebraic representation for tuple artefacts based on particular kinds of updates. Finally in [ABV18], we studied the explanation problem in a distributed setting; we provided a methodology to rewrite the workflow/programs to introduce better explanations for the users.

1.5 Organization of the manuscript

This manuscript highlights some selected results of my research since my PhD defence in February 2011. Detailed proofs and algorithms can be found in the original papers. I have chosen to highlight three research directions:

- The dialogue between logic and trees where I present the main translation approach from logic over databases to automata over trees and how to reduce some problems over trees to logic ones to obtain the main lower results. These results were published in [BBV16, BBGS20].
- The dialogue between trees and circuit where I present the main technique to compute a compact computation of the answers of a MSO formula over trees. Then, the two main algorithms to enumerate and rank enumeration are presented. Finally, I discuss the principles behind maintaining the compact representation of the answers through updates of the tree. The results were published in [ABMN19b, ABCM24].
- The dialogue between logic and circuit. I discuss the problem of evaluating provenance for Datalog queries. First, I give a new efficient representation of circuit called cycluits in order to obtain a FPT algorithm for a subclasse of Datalog program. Then, we explore different definitions of the computation of the provenance for Datalog and their impact. The results were published in [ABMS19] and [BBPT22a].

Chapter 2

Dialog between Logic and Trees

In the context of fixpoint logics, there exists a family of fixpoint having their satisfiability decidable: GFP, UNFP and GNFPUP. The approach to obtain decidability relies on the fact that the satisfiability can be restrained to check for instances that can be encoded through (infinite) trees. It is sufficient to encode the fixpoint formula into an MSO formula over the tree encoding. With the rise of Graph Databases, new kinds of recursive queries such as conjunctive regular path queries are not expressible with previous fixpoint logics. We studied a logics GNFP-UP which captures such queries in a more general manner. We also study query languages based on Datalog in [BKR15] for which the containement problem is decidable. These languages are captured by our fixpoint logic.

We also studied lower bounds of fixpoint logics and containment of recursive queries [BBGS20]. We presented that these questions can be reduced to some relevant problems over trees.

In this chapter, I will focus on two main techniques:(i) the technique developed in [BBV16] and [BKR15] to translate a logical formula in GNFP-UP into a tree automata used to obtain the exact complexity of the satisfiability result and their applications;(ii) the techniques developed in [BBGS20] to reduce a reasoning problem into a validity problem over trees to obtain lower bounds of the complexity of the reasoning problem. I will first introduce the preliminaries.

2.1 GNFP-UP and Automata Translation

The Guarded Fragment (GF) [ANvB98] is a fragment of first-order logic obtained by requiring in existential quantification $\exists \boldsymbol{x}.\phi(\boldsymbol{x})$ that ϕ be of the form $R(\boldsymbol{x}) \wedge \phi'(\boldsymbol{x})$, where $R(\boldsymbol{x})$ is an atom containing all free variables of ϕ', and requiring in universal quantification that ϕ be of the form $R(\boldsymbol{x}) \rightarrow \phi'(\boldsymbol{x})$, where R is as above. The Guarded Negation Fragment (GNF) [BtCS15] is an even more expressive decidable language, allowing unrestricted existential quantification but restricting negation to be of the form $R(\boldsymbol{x}) \wedge \neg\phi'(\boldsymbol{x})$, with R as above.

Given a formula $\phi(x_1 \dots x_m, y_1 \dots y_n)$ over some signature σ in which an m-ary second-order variable X occurs freely and positively in ϕ, and given a σ-structure $\mathfrak{A}$ and some fixed valuation ρ for $\boldsymbol{y}$, we can define a new m-ary relation: the relation is defined as the limit of a monotone sequence $X_0 \dots$, starting with $X_0 = \varnothing$ and then setting X_{i+1} to be the set of $\boldsymbol{a}$ such that ϕ holds when extending ρ with the interpretation of X as X_i and $\boldsymbol{x}$ as $\boldsymbol{a}$. Emphasizing the distinction between $\boldsymbol{x}$ and $\boldsymbol{y}$, we call $\boldsymbol{x}$ the *fixpoint variables* and $\boldsymbol{y}$ the *parameter variables*. Informally, during the fixpoint process, the fixpoint variables change in each iteration, while the parameter variables stay the same. Formulas in LFP can use relations defined using a fixpoint constructor like this, in addition to relations in σ.

Guarded Fixpoint Logic (denoted GFP or μGF) [GW99] extends GF with a fixpoint operator while maintaining decidability. The fixpoint constructor is restricted in two ways: the parameter variables $\boldsymbol{y}$ must be empty, and the fixpoint relation itself cannot be used as a guard. Guarded Negation Fixpoint Logic (GNFP) [BtCS15] adds fixpoint constructors to GNF with similar restrictions. It is known that the second restriction on these fixpoint logics is essential for decidability [GW99]. But what about the first? It certainly seemed important for the proofs of decidability; [GHO02] states that

> "It should be stressed that the presence of extra first-order parameters in fixed-point operations as well as the use of second-order variables and fixed points as guards is disallowed in μGF. These restrictions are essential for keeping the semantics invariant under guarded bisimulation. For instance, with the use of a first-order parameter ... one can define the transitive closure of a binary relation ... However, the transitive closure query is not invariant under guarded bisimulation and it is known that adding transitive closure to GF produces an undecidable logic [Grä99]."

In this section, we show that the parameter restriction can indeed be loosened. We introduce a variation of GNFP, denoted GNFP-UP, where the fixpoint variables of any fixpoint need to be guarded, but the fixpoint can carry additional *unguarded parameters*. One can write a GNFP-UP formula holding on the transitive closure of a binary relation. But such a formula cannot be used as a guard, and thus assertions that a binary relation is transitive (the key to undecidability in [Grä99]) cannot be expressed. GNFP-UP can express many properties related to transitivity, such as assertions of paths with certain properties (see the discussion of conjunctive regular path queries with inverse in Section 2.1.9).

The decidability of GFP is proven using an elegant high-level argument [Grä02]: one shows that satisfiable formulas must have tree-like models, and thus satisfiability can be reduced to satisfiability of a Monadic Second Order Logic sentence over trees, decidable via Rabin's theorem [Rab69]. A finer argument shows that from a GFP formula ϕ one can effectively create a tree automaton $\mathcal{A}_\phi$ which is non-empty exactly when ϕ is satisfiable. By analyzing the complexity of this automaton construction, Grädel and Walukiewicz derived a 2-ExpTime bound on satisfiability [GW99].

We begin by showing that the high-level argument easily extends to give decidability of GNFP-UP. The finer analysis of the complexity of GNFP-UP satisfiability requires more work. Because of negation and quantification in our logic, one might expect that the complexity would be a tower of exponentials based on the quantifier alternation. However, we show that the complexity is controlled by the *parameter depth* of the formula: informally, this is a number that measures the number of times we change parameters while passing from a formula to a subformula. We give elementary bounds for each parameter depth while proving that the complexity is non-elementary (but still primitive recursive) when the depth is not restricted. Each parameter depth includes formulas of arbitrary quantifier alternation; we avoid unnecessary exponential blowups by identifying pieces of the GNFP-UP formulas that behave like GNFP. We also show that some interesting logics fit within low parameter depth.

2.1.1 Preliminaries

We work with finite relational signatures σ. We use $\boldsymbol{x}, \boldsymbol{y}, \ldots$ (respectively, $\boldsymbol{X}, \boldsymbol{Y}, \ldots$) to denote vectors of first-order (respectively, second-order) variables. For a formula ϕ, we write $\mathrm{free}(\phi)$ to denote the free first-order variables of ϕ, and write $\phi(\boldsymbol{x})$ to indicate that these free variables are among $\boldsymbol{x}$. If we want to emphasize that there are also free second-order variables $\boldsymbol{X}$, we write $\phi(\boldsymbol{x}, \boldsymbol{X})$. We often use α to denote atomic formulas, an d if we write $\alpha(\boldsymbol{x})$ then we assume that the free first-order variables in α are precisely $\boldsymbol{x}$. The *width* of ϕ, denoted $\mathrm{width}(\phi)$, is the maximum number of free variables in any subformula of ϕ, and the width of a signature σ is the maximum arity of its relations.

2.1.2 FixPoint Logics

We briefly review the semantics of the fixpoint operator. Take some $\alpha(\boldsymbol{x}) \wedge \phi(\boldsymbol{x}, X, \boldsymbol{Y})$ where X appears only positively. Then it induces a monotone operator $U \mapsto \mathcal{O}_{\phi}^{\mathfrak{A},\boldsymbol{V}}(U) := \{\boldsymbol{a} : \mathfrak{A}, U, \boldsymbol{V} \models \alpha(\boldsymbol{a}) \wedge \phi(\boldsymbol{a}, X, \boldsymbol{Y})\}$ on every structure $\mathfrak{A}$ with valuation $\boldsymbol{V}$ for $\boldsymbol{Y}$, and this operator has a unique least fixpoint.

One way to obtain this least fixpoint is based on fixpoint approximants. Given some ordinal β, the *fixpoint approximant* $\phi^{\beta}(\mathfrak{A}, \boldsymbol{V})$ of ϕ on $\mathfrak{A}, \boldsymbol{V}$ is defined such that

$$\begin{aligned}
\phi^{0}(\mathfrak{A}, \boldsymbol{V}) &:= \varnothing \\
\phi^{\beta+1}(\mathfrak{A}, \boldsymbol{V}) &:= \mathcal{O}_{\phi}^{\mathfrak{A},\boldsymbol{V}}(\phi^{\beta}(\mathfrak{A}, \boldsymbol{V})) \\
\phi^{\beta}(\mathfrak{A}, \boldsymbol{V}) &:= \bigcup_{\beta' < \beta} \phi^{\beta'}(\mathfrak{A}, \boldsymbol{V}) \quad \text{where } \beta \text{ is a limit ordinal.}
\end{aligned}$$

We let $\phi^{\infty}(\mathfrak{A}, \boldsymbol{V}) := \bigcup_{\beta} \phi^{\beta}(\mathfrak{A}, \boldsymbol{V})$ denote the least fixpoint based on this operation. Thus, $[\mathbf{lfp}_{X}^{\boldsymbol{x}} .\alpha(\boldsymbol{x}) \wedge \phi(\boldsymbol{x}, X, \boldsymbol{Y})]$ defines a new predicate named X of arity $|\boldsymbol{x}|$, and $\mathfrak{A}, \boldsymbol{V}, \boldsymbol{a} \models [\mathbf{lfp}_{X}^{\boldsymbol{x}} .\alpha(\boldsymbol{x}) \wedge \phi(\boldsymbol{x}, X, \boldsymbol{Y})](\boldsymbol{x})$ iff $\boldsymbol{a} \in \phi^{\infty}(\mathfrak{A}, \boldsymbol{V})$. If $\boldsymbol{V}$ is empty or understood in context, we just write $\phi^{\infty}(\mathfrak{A})$.

It is often convenient to allow *simultaneous fixpoints* (also known as *vectorial fixpoints*). These are fixpoints of the form $[\mathbf{lfp}_{X_i}^{\boldsymbol{x}_i} .S](\boldsymbol{x})$ where S is a system of equations

$$\begin{cases}
X_1, \boldsymbol{x}_1 := \alpha_1(\boldsymbol{x}_1) \wedge \phi_1(\boldsymbol{x}_1, X_1, \ldots, X_j, \boldsymbol{Y}) \\
\vdots \\
X_j, \boldsymbol{x}_j := \alpha_j(\boldsymbol{x}_j) \wedge \phi_j(\boldsymbol{x}_j, X_1, \ldots, X_j, \boldsymbol{Y})
\end{cases}$$

where $X_1, \ldots, X_j$ occur only positively. Such a system can be viewed as defining a monotone operation on a vector of j valuations, where the i-th component in the vector is the set of tuples satisfying X_i (i.e. the i-th component is the valuation for X_i). The formula $[\mathbf{lfp}_{X_i}^{\boldsymbol{x}_i} .S](\boldsymbol{x})$ expresses that $\boldsymbol{x}$ is a tuple in the i-th component of the least fixpoint defined by this operation. Simultaneous points can be eliminated in favour of traditional points using what is known as the Bekic principle [AN01]. This can be done using a recursive procedure that eliminates a component of the simultaneous fixpoint by in-lining this formula in the other expressions. This in-lining process preserves any guardedness properties of the fixpoints, so we can allow simultaneous fixpoints in GNFP, UNFP, and GFP without changing the expressivity of these logics.

2.1.3 Guarded Logics

An atomic formula α is a *guard* for variables $\boldsymbol{x}$ if α uses every variable in $\boldsymbol{x}$. We say α is a guard for a formula ϕ if it is a guard for the free variables in ϕ. This means $\mathrm{free}(\alpha) \supseteq \mathrm{free}(\phi)$. Guards can take the form $\top$ (if ϕ is a sentence) or the form $x = x$ (if ϕ has one free variable x). A *strict guard* for a formula ϕ is a guard such that the free variables of α are identical to the free variables in ϕ; that is $\mathrm{free}(\alpha) = \mathrm{free}(\phi)$. For example, Rxy could serve as a strict guard for $\exists z.(Ryz \wedge Rzx)$.

We can also talk about guardedness within a structure $\mathfrak{A}$. Any set of elements of size at most 1 is considered to be both guarded and strictly guarded. Otherwise, we say a set U of elements in the domain of $\mathfrak{A}$ is *guarded* in $\mathfrak{A}$ if there is some atom $\alpha(\boldsymbol{a})$ such that every element in U appears in $\boldsymbol{a}$. In the special case that this atom uses precisely the elements in U and no more, then we say U is *strictly guarded* in $\mathfrak{A}$.

If we want to emphasize that the guards come from a certain signature σ', then we will say σ'-guarded or strictly σ'-guarded.

2.1.4 Basics of guarded logics

The *Guarded Negation Fragment* of FO [BtCS15] (denoted GNF) is built up inductively according to the grammar $\phi ::= R\boldsymbol{x} \mid \exists x.\phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \alpha(\boldsymbol{x}) \wedge \neg\phi(\boldsymbol{x})$ where R is either a relation symbol or the equality relation, and α is a guard for ϕ. If we restrict α to be an equality, then each negated formula can be rewritten to use at most one free variable; this is the *Unary Negation Fragment*, UNF [StC13]. GNF is also related to the *Guarded Fragment* [Grä02] (GF), typically defined via the grammar $\phi ::= R\boldsymbol{x} \mid \exists \boldsymbol{x}.\big(\alpha(\boldsymbol{xy}) \wedge \phi(\boldsymbol{xy})\big) \mid \phi \vee \phi \mid \phi \wedge \phi \mid \neg\phi(\boldsymbol{x})$ where R is either a relation symbol or the equality relation, and α is a guard for ϕ. Here it is the quantification that is guarded, rather than negation. GNF subsumes GF sentences and UNF formulas. GNF also subsumes GF formulas in which the free variables are guarded.

The fixpoint extensions of these logics (denoted GNFP, UNFP, and GFP) extend the base logic with formulas $[\mathbf{lfp}^{\boldsymbol{x}}_X .\alpha(\boldsymbol{x}) \wedge \phi(\boldsymbol{x}, X, \boldsymbol{Y})](\boldsymbol{x})$ where (i) $\alpha(\boldsymbol{x})$ is a guard for $\boldsymbol{x}$, (ii) X only appears positively in ϕ, (iii) second-order variables like X cannot be used as guards. Some alternative (but equi-expressive) ways to define the fixpoint extension are discussed in [BBtC13]; in all of the definitions, the important feature is that tuples in the fixpoint are guarded by an atom in the original signature. In UNFP, there is an additional requirement that only unary or 0-ary predicates can be defined using the fixpoint operators. GNFP subsumes both GFP sentences and UNFP formulas. These logics are all contained in LFP, the fixpoint extension of FO.

We will be interested in varying the signatures considered, and in distinguishing more finely which relations can be used in guards. If we want to emphasize the relational signature σ being used, then we will write, e.g., GNFP$[\sigma]$. For $\sigma' \subseteq \sigma$, we let GNFP$[\sigma, \sigma']$ denote the logic built up as in GNFP but allowing only equality or relations $R \in \sigma$ at the atomic step and only guards α using equality or relations $R \in \sigma'$. We define GFP$[\sigma, \sigma']$ similarly. Note that UNFP$[\sigma]$ is equivalent to GNFP$[\sigma, \varnothing]$, since if the only guards are equality guards, then the formula can be rewritten to use only unary negation and monadic fixpoints.

2.1.5 GNFP-UP

Free and bound variables The notion of free vs. bound second-order variables is standard. In particular, Y is free in $\phi(\boldsymbol{y}, \boldsymbol{z}, Y, \boldsymbol{Z})$ but bound in $[\mathbf{lfp}^{\boldsymbol{z}}_{Y,\boldsymbol{y}} .\phi](\boldsymbol{t})$. We assume no second-order variable Y is bound by more than one fixpoint operator, so each bound second-order variable Y identifies a unique fixpoint. If Y identifies a fixpoint with parameters $\boldsymbol{z}$, then $\mathrm{params}(Y) := \boldsymbol{z}$, the *parameters associated with the second-order variable* Y.

We use $\mathrm{free}(\phi)$ to denote the *free first-order variables* in ϕ. It is defined recursively. For atoms $R\boldsymbol{t}$ with $R \in \sigma$ and $\boldsymbol{t}$ a tuple consisting of constants and variables, the free first-order variables are just the variables in $\boldsymbol{t}$. For $Y\boldsymbol{t}$ with Y a second-order variable, $\mathrm{free}(Y\,\boldsymbol{t})$ is the union of the variables in $\boldsymbol{t}$ *and* $\mathrm{params}(Y)$. For boolean connectives, $\mathrm{free}(\phi_1 \wedge \phi_2) = \mathrm{free}(\phi_1 \vee \phi_2) = \mathrm{free}(\phi_1) \cup \mathrm{free}(\phi_2)$, and $\mathrm{free}(\neg\phi) = \mathrm{free}(\phi)$. For quantification, $\mathrm{free}(\exists x.\phi) = \mathrm{free}(\phi)\setminus\{x\}$. Finally, for $[\mathbf{lfp}^{\boldsymbol{z}}_{Y,\boldsymbol{y}} .\phi](\boldsymbol{t})$, the free first-order variables consist of the parameter variables $\boldsymbol{z}$ together with the variables in $\boldsymbol{t}$.

The parameters in ϕ consist of the union of $\mathrm{params}(Y)$ for all second-order variables Y occurring in ϕ; we let $\mathrm{params}(\phi)$ denote the *subset of these parameters that occur free in* ϕ.

GNFP-UP Guarded negation fixpoint logic with unguarded parameters (GNFP-UP) is the fragment of LFP that allows unguarded parameter variables in fixpoint definitions, but requires fixpoint variables and negation to be guarded. Formally, a GNFP-UP$[\sigma]$ *formula* ϕ is generated recursively from the following

grammar:

$$\begin{aligned}\phi ::= {} & R\,\boldsymbol{t} \mid Y\,\boldsymbol{t} \mid \phi \wedge \phi \mid \phi \vee \phi \mid \exists y.\phi \mid \\ & \alpha \wedge \neg\phi \text{ where } \mathrm{free}(\phi) \subseteq \mathrm{free}(\alpha) \mid \\ & [\mathbf{lfp}^{\boldsymbol{z}}_{Y,\boldsymbol{y}}\,.\,\mathrm{gdd}(\boldsymbol{y}) \wedge \phi(\boldsymbol{y},\boldsymbol{z},Y,\boldsymbol{Z})](\boldsymbol{t}) \text{ for } \phi \text{ positive in } Y\end{aligned}$$

where $\boldsymbol{t}$ is a tuple of variables or constants, $R\boldsymbol{t}$ and α are atoms using a relation in σ or $=$, and $\mathrm{gdd}(\boldsymbol{y})$ is defined below.

Guardedness The *guardedness predicate* $\mathrm{gdd}(\boldsymbol{y})$ asserts $\boldsymbol{y}$ is guarded by an atom in σ or $=$. It can be understood as an abbreviation for the disjunction of existentially quantified atoms that use a relation from σ or $=$ and involve all of the variables in $\boldsymbol{y}$. Because of this, only guarded relations can be defined using fixpoints in GNFP-UP: i.e. any tuple of elements in the relation defined by the fixpoint formula must already be guarded by an atom in the base signature σ. Note that the relations defined using a fixpoint operator cannot be used as guards.

The parameters $\boldsymbol{z}$ are not required to be guarded in the fixpoint definition. However, for negation, parameters are treated like other variables and must be guarded. For example, if $\alpha(\boldsymbol{x})$ is an atomic formula over σ, and Y identifies a fixpoint with parameters $\boldsymbol{z}$, then $\alpha(\boldsymbol{x}) \wedge \neg Y\,\boldsymbol{x}$ is not permitted since Y implicitly uses parameters $\boldsymbol{z}$ and these parameters are not guarded by α (since the free variables in $Y\,\boldsymbol{x}$ are really $\boldsymbol{x}$ and $\boldsymbol{z}$).

A formula ϕ that includes free first-order variables $\boldsymbol{x}$ is *$\boldsymbol{x}$-guarded* if it is logically equivalent to $\mathrm{gdd}(\boldsymbol{x}) \wedge \phi$. If $\mathrm{free}(\phi) = \boldsymbol{x}$ and ϕ is $\boldsymbol{x}$-guarded, then we say it is *answer-guarded*. Sentences or formulas with one free variable are always answer-guarded since we can use a trivial guard like $x = x$. For readability purposes, we often omit such trivial guards.

Normal form A *conjunctive query* (CQ) is $\exists\boldsymbol{y}.\psi$ for ψ a conjunction of atoms. A *union of conjunctive queries* (UCQ) is a disjunction of CQs. Such queries are expressible in GNF. It is helpful to work with GNFP-UP in a normal form that highlights the fact that GNFP-UP formulas can be built up from UCQ-shaped formulas using guarded negation and guarded fixpoints with parameters.

Formally, A *normal form* GNFP-UP$[\sigma]$ *formula* ϕ or ψ is generated recursively from the following grammars:

$$\begin{aligned}\phi ::= {} & \textstyle\bigvee_i \exists \boldsymbol{y}_i.\, \bigwedge_j \psi_{ij} \\ \psi ::= {} & R\,\boldsymbol{t} \mid Y\,\boldsymbol{t} \mid \alpha \wedge \neg\phi \text{ where } \mathrm{free}(\phi) \subseteq \mathrm{free}(\alpha) \mid \\ & [\mathbf{lfp}^{\boldsymbol{z}}_{Y_m,\boldsymbol{y}_m}.S](\boldsymbol{t})\end{aligned}$$

where $\boldsymbol{t}$ is a tuple of variables or constants, $R\boldsymbol{t}$ and α are atoms using a relation in σ or $=$, and S is a system with equations of the form $Y, \boldsymbol{y} := \mathrm{gdd}(\boldsymbol{y}) \wedge \phi(\boldsymbol{y},\boldsymbol{z},\boldsymbol{Y},\boldsymbol{Z})$, as described earlier.

Any GNFP-UP formula can be converted into normal form in a canonical way. The *width* of a GNFP-UP formula is the maximum number of free variables used in any subformula after the formula is converted into normal form. We write $(\text{GNFP-UP})^k[\sigma]$ for GNFP-UP formulas of width k.

GNFP-UP vs. GNFP A good example to keep in mind is that GNFP-UP can express the transitive closure of a binary relation R.

Example 1 *Suppose R is a binary relation in σ. Consider the following* GNFP-UP$[\sigma]$*-formula:*

$$\phi(x,z) := [\mathbf{lfp}^{z}_{Y,y} \ . \ Ryz \vee \exists y'.(Ryy' \wedge Yy')](x) \ .$$

Observe that ϕ has two free variables, the variable x being tested in the fixpoint and the parameter variable z. The formula $\phi(x,z)$ expresses that there is some R-path from element x to z, i.e. (x,z) is in the transitive closure of R.

We can express that x participates in an R-cycle by using the formula $\phi(x,x)$.

We cannot express that the structure is strongly R-connected, since this would require unguarded negation, but we can say every pair of guarded elements is R-connected: $\neg\exists xz.(\mathrm{gdd}(x,z) \wedge \neg\phi(x,z)) \in$ GNFP-UP.

The sentence $\neg\exists xz.(\phi(x,z) \wedge \neg Rxz) \vee (Rxz \wedge \neg\phi(x,z))$ that says R is transitively closed is not in GNFP-UP*, since we cannot use the fixpoint relation defined by ϕ as a guard for $\neg Rxz$.*

In LFP, it is always possible to eliminate the use of parameters by increasing the arity of the defined fixpoint predicates and passing the parameters explicitly in the fixpoint. This is not usually possible in our context, because the fixpoint variables are required to be guarded. Indeed, it can be shown that the transitive closure of a binary relation R cannot be expressed in GNFP (the fragment of GNFP-UP in which fixpoints do not use any parameters).

Proposition 1 GNFP-UP *is strictly more expressive than* GNFP*, even over finite structures.*

2.1.6 Tree-Like Property

Our results rely heavily on a different model theoretic property, called the tree-like model property. We review now what it means for a relational structure to be "tree-like". Roughly speaking, these are structures that can be decomposed into a tree form. Formally, a *tree decomposition* of a structure $\mathfrak{M}$ consists of a tree (V,E) and a function λ assigning to each vertex $v \in V$ a subset $\lambda(v)$ of elements in the domain of $\mathfrak{M}$, so that the following hold:

- For each atom $Rc_1 \dots c_n$ that holds in $\mathfrak{M}$, there is a v such that $\lambda(v)$ includes each element of $c_1 \dots c_n$.
- For each domain element e in the domain of $\mathfrak{M}$, the set of nodes

  $$\{v \in V : e \in \lambda(v)\}$$

  is a connected subset of the tree. In other words, for any two vertices v_1, v_2 such that $e \in \lambda(v_1)$ and $e \in \lambda(v_2)$, there is a path between v_1 and v_2 such that $e \in \lambda(u)$ for every node u on this path.

The *width* of a decomposition is one less than the maximum size of $\lambda(v)$ over any element $v \in V$. The subsets $\lambda(v)$ of $\mathfrak{M}$ are called *bags* of the decomposition, so structures of tree-width $k-1$ have bags of size at most k.

GNFP (and hence UNFP and GFP) has the *tree-like model property* [BtCS15]: if ϕ is satisfiable, then ϕ is satisfiable over structures with tree decompositions of some bounded tree-width. In fact satisfiable GNFP^k formulas have satisfying structures of tree-width $k-1$. Satisfiable GFP sentences have an even stronger property: each bag in the decomposition describes a guarded set of elements, so the width of the tree decomposition is bounded by the maximum arity of the relations.

It is well-known that structures of tree-width $k-1$ can be encoded by labelled trees over an alphabet that depends only on the signature σ of the structure and k. Our encoding scheme will make use of

trees with both node and edge labels, i.e. trees over a transition system signature σ_k. Each node in a tree code represents atomic information over at most k elements, so the signature σ_k includes unary predicates to indicate the number of elements represented at that node, and the atomic relations that hold those elements. The signature includes binary predicates that indicate the overlap and relationship between the names of elements encoded in neighbouring nodes of the tree. Formally, σ_k contains the following relations:

- There are unary relations $D_n \in \sigma_k$ for $n \in \{0, \dots, k\}$, to indicate the number of elements represented at each node. We call these *domain predicates* since they are used to specify the number of domain elements encoded at a given node.
- For every relation $R \in \sigma$ of arity n and every sequence $\boldsymbol{i} = i_1 \dots i_n$ over $\{1, \dots, k\}$, there is a unary relation $R_{\boldsymbol{i}} \in \sigma_k$ to indicate that the tuple of elements coded by $\boldsymbol{i}$ is a tuple of elements in R. For example, if T is a ternary relation in σ and a_i is the element coded by name i in some node, then $T_{3,1,3}$ indicates that $T(a_3, a_1, a_3)$ holds.
- For every partial 1-1 map ρ from $\{1, \dots, k\}$ to $\{1, \dots, k\}$, there is a binary relation $E_\rho \in \sigma_k$ to indicate the relationship between the names of elements in neighbouring nodes. For example, if $(u, v) \in E_\rho$ and $\rho(3) = 1$, then the element with name 3 in u is the same as the element with name 1 in v.

For a unary relation $R_{\boldsymbol{i}}$, we write $\text{indices}(R_{\boldsymbol{i}})$ to denote the set of elements from $\{1, \dots, k\}$ appearing in $\boldsymbol{i}$. We will refer to the elements of $\{1, \dots, k\}$ as *indices* or *names*.

For nodes u, v in a σ_k-tree $\mathcal{T}$ and names i, j, we will say (u, i) is *equivalent* to (v, j) if there is a simple undirected path $u = u_1 u_2 \dots u_n = v$ in $\mathcal{T}$, and $\rho_1, \dots, \rho_{n-1}$ such that $(u_i, u_{i+1}) \in E^{\mathcal{T}}_{\rho_i}$ or $(u_{i+1}, u_i) \in E^{\mathcal{T}}_{\rho_i^{-1}}$, and $(\rho_{n-1} \circ \dots \circ \rho_1)(i) = j$. In words, the i-th element in node u corresponds to the j-th element in node v, based on the composition of edge labels (or their inverses) on the simple path between u and v. We write $[u, i]$ for the equivalence class based on this equivalence relation.

Given some sub signature $\sigma' \subseteq \sigma$ and some set of indices $I \subseteq \{1, \dots, k\}$, we say that $R_{\boldsymbol{i}} \in \sigma_k$ is a *σ'-guard* for I if $\text{indices}(R_{\boldsymbol{i}}) \supseteq I$ and $R \in \sigma'$. Likewise, $R_{\boldsymbol{i}} \in \sigma_k$ is a *strict σ'-guard* for I if $\text{indices}(R_{\boldsymbol{i}}) = I$ and $R \in \sigma'$. Given a set τ of unary relations from σ_k we say I *is σ'-guarded in* τ if $|I| \leqslant 1$ or there is some $R_{\boldsymbol{i}} \in \tau$ that is a σ'-guard for I. Similarly, we say I *is strictly σ'-guarded in* τ if $|I| \leqslant 1$ or there is some $R_{\boldsymbol{i}} \in \tau$ that is a strict σ'-guard for I.

Given some σ_k-tree $\mathcal{T}$, we say $\mathcal{T}$ is *consistent* if it satisfies certain natural conditions that ensure that the tree corresponds to a code of some tree decomposition of a σ-structure:

1. there is exactly one domain predicate D_i that holds at each node, and the root v_0 is in $D_0^{\mathcal{T}}$;
2. edge labels respect the domain predicates: if $u \in D_m^{\mathcal{T}}$, $v \in D_n^{\mathcal{T}}$, and $(u, v) \in E_\rho^{\mathcal{T}}$, then $\text{dom}(\rho) \subseteq \{1, \dots, m\}$ and $\text{rng}(\rho) \subseteq \{1, \dots, n\}$;
3. node labels respect the domain predicates: if $v \in D_n^{\mathcal{T}}$ and $v \in R_{\boldsymbol{i}}^{\mathcal{T}}$, then $\text{indices}(R_{\boldsymbol{i}}) \subseteq \{1, \dots, n\}$;
4. neighbouring node labels agree on shared names: if $u \in R_{\boldsymbol{i}}^{\mathcal{T}}$, $(u, v) \in E_\rho^{\mathcal{T}}$, and $\text{indices}(R_{\boldsymbol{i}}) \subseteq \text{dom}(\rho)$, then $v \in R^{\mathcal{T}}_{\rho(\boldsymbol{i})}$; similarly, if $v \in R_{\boldsymbol{i}}^{\mathcal{T}}$, $(u, v) \in E_\rho^{\mathcal{T}}$, and $\text{indices}(R_{\boldsymbol{i}}) \subseteq \text{rng}(\rho)$, then $u \in R^{\mathcal{T}}_{\rho^{-1}(\boldsymbol{i})}$;

where $P^{\mathcal{T}}$ denotes the interpretation of relation P in $\mathcal{T}$.

It is now easy to verify the fact mentioned at the beginning of this subsection: tree decompositions of every σ-structure of tree-width $k - 1$ can be encoded in consistent σ_k-trees.

The next step is to describe how a consistent σ_k-tree can be decoded to an actual σ-structure. The *decoding* of $\mathcal{T}$ is the σ-structure $\mathfrak{D}(\mathcal{T})$ where the universe is the set

$$\{[v, i] : v \in \text{dom}(\mathcal{T}) \text{ and } i \in \{1, \dots, k\}\}$$

and a tuple $([v_1, i_1], \ldots, [v_r, i_r])$ is in $R^{\mathfrak{D}(\mathcal{T})}$ iff there is some node $w \in \mathrm{dom}(\mathcal{T})$ such that $w \in R_{j_1 \ldots j_r}$ and $[w, j_m] = [v_m, i_m]$ for all $m \in \{1, \ldots, r\}$.

Finally, we introduce some notation related to σ_k. We often use τ to denote a node label, and $\tau(v)$ to denote the label at some node v in a tree. We write $\mathrm{EdgeLabels}$ for the set of functions ρ such that the binary predicate E_ρ is in σ_k. We write $\mathrm{NodeLabels}$ for the set of *internally consistent* node labels, i.e. the set consisting of sets of unary predicates from σ_k that satisfy properties (1) and (3) in the definition of consistency above.

2.1.7 Automata

We define two automaton models that can function on trees that have an unbounded (possibly infinite) branching degree. This is because the tree codes derived from the unravellings described earlier may have this unbounded branching. We describe these automata below but will assume familiarity with standard automata theory over infinite structures (see, e.g., [Tho97]).

Fix a transition system signature Σ consisting of unary relations Σ_p and binary relations Σ_a (for the node labels and edge labels, respectively).

A *2-way alternating μ-automaton* $\mathcal{A}$ is a tuple $\langle \Sigma, Q_E, Q_A, q_0, \delta, \Omega \rangle$ where $Q := Q_E \cup Q_A$ is a finite set of states partitioned into states Q_E controlled by Eve and states Q_A controlled by Adam, and $q_0 \in Q$ is the initial state. The transition function has the form

$$\delta : Q \times \mathcal{P}(\Sigma_p) \to \mathcal{P}(\mathrm{Dir} \times \Sigma_a \times Q)$$

where $\mathrm{Dir} = \{\uparrow, 0, \downarrow\}$ is the set of possible directions (up $\uparrow$, stay 0, down $\downarrow$). The acceptance condition is a parity condition specified by $\Omega : Q \to \mathrm{Pri}$, which maps each state to a priority in a finite set of priorities Pri.

Let $\mathcal{T}$ be a tree over Σ, and let $\mathcal{T}(v)$ denote the set of unary propositions in Σ_p that hold at v.

The notion of acceptance of $\mathcal{T}$ by $\mathcal{A}$ starting at node $v_0 \in \mathrm{dom}(\mathcal{T})$ is defined in terms of a game $\mathcal{G}(\mathcal{A}, \mathcal{T}, v_0)$. The arena is $Q \times \mathrm{dom}(\mathcal{T})$, and the initial position is (q_0, v_0). From a position (q, v) with $q \in Q_E$ (respectively, $q \in Q_A$), Eve (respectively Adam) selects $(d, a, r) \in \delta(q, \mathcal{T}(v))$, and an a-neighbor w of v in direction d (note if $d = 0$, then v is considered the only option, and we sometimes write just $(0, r)$ instead of $(0, a, r)$). The game continues from position (r, w).

A *play* in $\mathcal{G}(\mathcal{A}, \mathcal{T}, v_0)$ is a sequence $(q_0, v_0), (q_1, v_1), (q_2, v_2), \ldots$ of moves in the game. Such a *play is winning* for Eve if the *parity condition* is satisfied: the maximum priority that occurs infinitely often in $\Omega(q_0), \Omega(q_1), \ldots$ is even.

A *strategy* for one of the players is a function that returns the next choice for that player given the history of the play. If the function depends only on the current position (rather than the full history), then it is *positional*. Choosing a strategy for both players fixes a play in $\mathcal{G}(\mathcal{A}, \mathcal{T}, v_0)$. A play π is *compatible* with a strategy ζ if there is a strategy for the other player such that ζ and ζ' yield π. A *strategy is winning* for Eve if every play compatible with it is winning.

We write $L_{v_0}(\mathcal{A})$ for the set of trees $\mathcal{T}$ such that Eve has a winning strategy in $\mathcal{G}(\mathcal{A}, \mathcal{T}, v_0)$. If v_0 is the root of $\mathcal{T}$, then we just write $L(\mathcal{A})$ to denote the *language* of $\mathcal{A}$.

The *dual* of a 2-way alternating μ-automaton $\mathcal{A}$ is the automaton $\mathcal{A}'$ obtained from $\mathcal{A}$ by switching Q_A and Q_E, and incrementing each priority by 1 (i.e. $\Omega'(q) := \Omega(q) + 1$). This has the effect of switching the roles of the two players, so the resulting automaton accepts the complement of $L(\mathcal{A})$.

These 2-way alternating μ-automata are essentially the same as the automata used in [Grä02]; we use slightly different notation here and allow directions stay, up, and down, rather than just stay and 'move to neighbour'.

We are also interested in a type of automaton on trees with arbitrary branching that operates in a 1-way, nondeterministic fashion. These automata were introduced by Janin-Walukiewicz [JW95, JW96]; we follow the presentation given in [DH00]. A *μ-automaton* $\mathcal{M}$ is a tuple $\langle \Sigma_p, \Sigma_a, Q, q_0, \delta, \Omega \rangle$. where the transition function now has the form

$$\delta : Q \times \mathcal{P}(\Sigma_p) \to \mathcal{P}(\mathcal{P}(\Sigma_a \times Q)).$$

Again, the acceptance condition is a parity condition specified by Ω. As before, we define acceptance of $\mathcal{T}$ from a node $v_0 \in \mathrm{dom}(\mathcal{T})$ based on a game $\mathcal{G}(\mathcal{A}, \mathcal{T}, v_0)$. The arena is $Q \times \mathrm{dom}(\mathcal{T})$, and the initial position is (q_0, v_0). From a position (q, v), Eve selects some $S \in \delta(q, \mathcal{T}(v))$, and a marking of every successor of v with a set of states such that (i) for all $(a, r) \in S$, there is some a-successor whose marking includes r, and (ii) for all a-successors w of v, if r is in the marking of w, then there is some $(a, r) \in S$. Adam then selects some successor w of v and a state r in the marking of w chosen by Eve, and the game continues from position (r, w). A winning play and strategy is defined as above.

Properties of μ-automata These automata are bisimulation invariant on trees.

Proposition 1 (JaninW95) *Let $\mathcal{A}$ be a 2-way alternating μ-automaton or a μ-automaton. For all trees $\mathcal{T}$, if $\mathcal{T} \in L(\mathcal{A})$ and $\mathcal{T}'$ is bisimilar to $\mathcal{T}$, then $\mathcal{T}' \in L(\mathcal{A})$.*

These automata models also have nice closure properties.

Proposition 2 *2-way alternating μ-automata are closed under:*

- *Intersection: Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be 2-way alternating μ-automata. Then we can construct a 2-way alternating μ-automaton $\mathcal{A}$ such that $L(\mathcal{A}) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$, and the size of $\mathcal{A}$ is linear in $|\mathcal{A}_1| + |\mathcal{A}_2|$.*
- *Union: Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be 2-way alternating μ-automata. Then we can construct a 2-way alternating μ-automaton $\mathcal{A}$ such that $L(\mathcal{A}) = L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$, and the size of $\mathcal{A}$ is linear in $|\mathcal{A}_1| + |\mathcal{A}_2|$.*
- *Complement: Let $\mathcal{A}$ be a 2-way alternating μ-automaton. Then we can construct a 2-way alternating μ-automaton $\mathcal{A}'$ of size at most $|\mathcal{A}|$ such that $L(\mathcal{A}')$ is the complement of $L(\mathcal{A})$.*

It is straightforward to construct a 2-way alternating μ-automaton that is equivalent to a given μ-automaton. Moreover, it is known that μ-automata, 2-way alternating μ-automata and the μ-calculus are equivalent over trees (this follows from [JW95]).

Theorem 1 ([JW95]) *Given $\phi \in \mathrm{L}_\mu[\Sigma]$, we can construct a μ-automaton $\mathcal{A}$ such that $L(\mathcal{A})$ is the set of Σ-trees such that $\mathcal{T} \models \phi$.*

Likewise, given a μ-automaton or 2-way alternating μ-automaton $\mathcal{A}$ over signature Σ, we can construct $\phi \in \mathrm{L}_\mu[\Sigma]$ such that $L(\mathcal{A})$ is the set of Σ-trees such that $\mathcal{T} \models \phi$.

A 1-way alternating automaton is an automaton that uses only directions $\mathbf{l}$ and $\mathbf{r}$. A (1-way) nondeterministic automaton is a 1-way alternating automaton such that every transition function formula is of the form $\bigvee_i (\mathbf{l}, q_{\mathbf{l}}^i) \wedge (\mathbf{r}, q_{\mathbf{r}}^i)$.

Closure properties We recall some closure properties of these automata First, the automata that we are using are closed under union and intersection (of their languages).

Proposition 2 *2-way alternating parity tree automata and 1-way nondeterministic parity tree automata are closed under union and intersection, with only a polynomial blow-up in the number of states and overall size.*

For example, this means that if we are given 2-way alternating parity tree automata $\mathcal{A}_1$ and $\mathcal{A}_2$, then we can construct in P a 2-way alternating parity tree automaton $\mathcal{A}$ such that $L(\mathcal{A}) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$. In automaton constructions, when we say, e.g., "take the intersection of $\mathcal{A}_1$ and $\mathcal{A}_2$", we mean take this automaton $\mathcal{A}$ such that $L(\mathcal{A}) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$.

Another important language operation is projection. Let L' be a language of trees over propositions $\Sigma \cup \{P\}$. The *projection* of L' with respect to P is the language of trees $\mathcal{T}$ over Σ such that there is some $\mathcal{T}' \in L'$ such that $\mathcal{T}$ and $\mathcal{T}'$ agree on all propositions in Σ. Projection is easy for nondeterministic automata since the valuation for the projected proposition can be guessed by Eve.

Proposition 3 *1-way nondeterministic parity tree automata are closed under projection, with no change in the number of states and overall size.*

Finally, complementation is easy for alternating automata by taking the *dual* automaton, obtained by switching conjunctions and disjunctions in the transition function, and incrementing all of the priorities by one.

Proposition 4 *2-way alternating parity tree automata are closed under complementation, with no change in the number of states and overall size.*

Connections between 2-way and 1-way automata It was shown by Vardi [Var98] that 2-way alternating parity tree automata can be converted to equivalent 1-way nondeterministic automata, with an exponential blow-up.

Theorem 1 ([Var98]) *Let $\mathcal{A}$ be a 2-way alternating parity tree automaton. We can construct a 1-way nondeterministic parity tree automaton $\mathcal{A}'$ such that $L(\mathcal{A}) = L(\mathcal{A}')$. The number of states of $\mathcal{A}'$ is exponential in the number of states of $\mathcal{A}$, but the number of priorities of $\mathcal{A}'$ is linear in the number of priorities of $\mathcal{A}$.*

1-way nondeterministic tree automata can be seen as a special case of 2-way alternating automata, so the previous theorem shows that 1-way nondeterministic and 2-way alternating parity automata are equivalent, in terms of their ability to recognize trees starting from the root.

We need another conversion from 1-way nondeterministic to 2-way alternating automata that we call *localization*. This is the process by which a 1-way nondeterministic automaton that is running on trees with extra information about some predicate annotated on the tree is converted to an equivalent 2-way alternating automaton that operates on trees without these annotations, but under the assumption that these predicates hold only locally at the position the 2-way automaton is launched from. A similar localization idea is present in prior work (see, e.g., [BCK$^+$14, BKR15]).

Theorem 2 *Let $\Sigma' := \Sigma \cup \{P_1, \ldots, P_j\}$. Let $\mathcal{A}'$ be a 1-way nondeterministic parity automaton on Σ'-trees. We can construct a 2-way alternating parity automaton $\mathcal{A}$ on Σ-trees such that for all Σ-trees $\mathcal{T}$ and $v \in \mathrm{dom}(\mathcal{T})$,*

$$\mathcal{A}' \text{ accepts } \mathcal{T}' \text{ from the root iff } \mathcal{A} \text{ accepts } \mathcal{T} \text{ from } v,$$

where $\mathcal{T}'$ is the Σ'-tree obtained from $\mathcal{T}$ by setting $P_1^{\mathcal{T}'} = \cdots = P_j^{\mathcal{T}'} = \{v\}$. The number of states of $\mathcal{A}$ is linear in the number of states of $\mathcal{A}'$, and the overall size is linear in the size of $\mathcal{A}'$.

Emptiness testing Finally, we make use of the well-known fact that language emptiness of tree automata is decidable.

Theorem 3 ([EJ88],[Var98]) *For 1-way nondeterministic parity tree automata, emptiness is decidable in time polynomial in the number of states and exponential in the number of priorities. For 2-way alternating parity automata, it is decidable in time exponential in the number of states and priorities.*

2.1.8 Translation from Logic to Automata

In this subsection, we construct automata for θ in GNFP-UP$[\sigma]$. Before we give some details of the construction, it is helpful to consider how automata can be used to analyze fixpoints.

Using localized automata for fixpoints Testing whether some tuple $\boldsymbol{t}$ is in the least fixpoint $[\mathbf{lfp}^{\vec{z}}_{Y,\boldsymbol{y}} \,.\, \phi]$ in some structure $\mathfrak{A}$ and for some fixed valuation of the parameters (and any other free variables) can be viewed as a game. Positions in this game consist of the current tuple $\boldsymbol{y}$ being tested in the fixpoint, with the initial position being $\boldsymbol{t}$. In general, in position $\boldsymbol{y}$, one round of the game consists of the following:

- Eve chooses some valuation for Y such that $\phi(\boldsymbol{y}, Y)$ holds (if it is not possible, she loses), then
- Adam chooses tuple $\boldsymbol{y}' \in Y$ (if it is not possible, he loses), and the game proceeds to the next round in position $\boldsymbol{y}'$.

Adam wins if the game continues forever.

The idea is that if $\boldsymbol{t}$ is really in the least fixpoint, then it must be added in some fixpoint approximant. This gives Eve a strategy for choosing Y at each stage in the game, in such a way that after finitely many challenges by Adam, she should be able to guess the empty valuation.

When the fixpoint can consist of only guarded tuples, there is a version of this game on a tree encoding $\mathcal{T}$ of a structure, that can be implemented using tree automata. We start with an automaton $\mathcal{A}_\phi$ for the body ϕ of the fixpoint. We start with *localized versions* of this automaton because we need to launch different versions based on Adam's challenges. A *local assignment* $\boldsymbol{b}/\boldsymbol{y}$ for $\boldsymbol{b} = b_1 \dots b_n \in U_k^n$ and $\boldsymbol{y} = y_1 \dots y_n$ is a mapping such that $y_i \mapsto b_i$. A node v in $\mathcal{T}$ with $\boldsymbol{b} \subseteq \overline{v}$ and a local assignment $\boldsymbol{b}/\boldsymbol{y}$, specifies a valuation for $\mathrm{encode}\boldsymbol{y}$. We say it is local since the free variable markers for $\boldsymbol{y}$ would all appear locally in v. If we have an automaton $\mathcal{A}$ running on trees with free variable markers for $\boldsymbol{y}$, we say that we *localize $\mathcal{A}$ to $\boldsymbol{b}/\boldsymbol{y}$* if we apply the localization theorem (Theorem 2) to the predicates V_{b_i/y_i}, and then eliminate the dependence on any other V_{c/y_i} for $c \neq b_i$ by always assuming these predicates do not hold. This results in an automaton that simulates $\mathcal{A}$ under the assumption that the free variables $\boldsymbol{y}$ correspond to the elements $[v, \boldsymbol{b}]$, but it no longer relies on free variable markers for $\boldsymbol{y}$. These localized automata are important because they can be launched to test that a tuple of elements that appear together in a node satisfy some property — without having the markers for this tuple explicitly written on the tree.

We now describe the version of the fixpoint game using localized automata. Initially, Eve navigates to a node in $\mathcal{T}$ carrying $\boldsymbol{t}$, and launches the appropriate localized $\mathcal{A}_\phi$ from there. In general, the game proceeds as follows:

- Eve and Adam simulate some localized version of $\mathcal{A}_\phi$. During the simulation, Eve can guess a valuation for Y (recall that $\mathcal{A}_\phi$ runs on trees with an annotation describing the valuation for the second-order variable Y, and that information is missing from $\mathcal{T}$). Because Y can only contain guarded tuples, this amounts to guessing an annotation of the tree with this valuation.
- When Eve guesses some $\boldsymbol{y}' \in Y$, Adam can either continue the simulation, or challenge her on this assertion. A challenge corresponds to launching a new localized copy of $\mathcal{A}_\phi$ from the node carrying $\boldsymbol{y}'$ (again, we know that $\boldsymbol{y}'$ must be present locally in a node, since any tuple in the fixpoint must be guarded).

After each challenge, the game continues as before (with the new copy of $\mathcal{A}_\phi$ being simulated, Eve guessing a new valuation for Y, etc.). Adam wins if he challenges infinitely often, or if the game stabilizes in some simulation of $\mathcal{A}_\phi$ where he wins.

Assuming we have localized automata for ϕ, we can implement this game using a 2-way alternating parity automaton. We assign a large odd priority (larger than the priorities in $\mathcal{A}_\phi$) to the states where Adam challenges, so that he wins if he is able to challenge infinitely many times; the other priorities are just inherited from $\mathcal{A}_\phi$. Simultaneous fixpoints can be handled similarly.

To analyze the fixpoints like this, our inductive automaton construction must produce 2-way localized automata at each stage — if we did not, then each time we reached a fixpoint and needed localized automata for the body of the fixpoint, we would get an exponential blow-up. For GFP and GNFP, we can define directly the localized versions of the automata using a state set of size at most singly exponential in the size of the input formula. However, by adding parameters in GNFP-UP, this direct definition of a localized version becomes more challenging. We are forced to construct non-localized automata at some points — namely, for subformulas that introduce new parameters — and then apply Theorems 1 and 2, resulting in an exponential blow-up. The parameter depth is a measure of how many of these blow-ups occur.

Construction We now describe more details of the construction of an automaton for normal form $\theta \in \text{GNFP-UP}[\sigma]$. First, it is straightforward to construct an automaton that checks consistency:

Proposition 5 *There is a 2-way alternating parity tree automaton $\mathcal{C}$ that checks whether or not a σ_k-tree (possibly extended with additional free variable markers for $\boldsymbol{z}$ and $\boldsymbol{Z}$) is consistent. The size of $\mathcal{C}$ is at most exponential in $(|\sigma| + |\boldsymbol{z}| + |\boldsymbol{Z}|) \cdot |U_k|^k$.*

Hence, we can concentrate on defining an automaton for θ that runs on consistent trees and accepts iff the decoding of the consistent input tree satisfies θ.

The main theorem states that the size of the automaton for θ is a tower of exponentials whose height depends on the pdepth. Given a function f, we write $\exp_f^n(m)$ for a tower of exponentials of height n based on f, i.e. $\exp_f^0(m) = m$ and $\exp_f^n(m) = 2^{f(\exp_f^{n-1}(m))}$.

Theorem 4 *For normal form $\theta \in (\text{GNFP-UP})^k[\sigma]$ with $\mathit{pdepth}(\theta) \geqslant 1$, we can construct a 2-way alternating parity tree automaton $\mathcal{A}_\theta$ such that for all consistent σ_k-trees $\mathcal{T}$, $\mathfrak{D}(\mathcal{T}) \models \theta$ iff $\mathcal{T} \in L(\mathcal{A}_\theta)$, and the size of $\mathcal{A}_\theta$ is at most $(\mathit{pdepth}(\theta)+1)$-exponential in $|\theta|$.*

More precisely, there is a polynomial function f independent of θ such that the size is at most $\exp_f^{\mathit{pdepth}(\theta)}(f(m) \cdot 2^{f(klr)})$ where $m = |\theta|$, $l = |\mathrm{const}(\sigma)|$, and $r = \mathrm{rank}_{\mathrm{CQ}}(\sigma)$ (see definitions below).

The main factor affecting the output size is the pdepth, since this determines the height of the tower of exponentials. However, for more precise bounds, the other factors affecting the size are the *size of the formula* θ, the *width* k, the number of *constants* in σ, and the *CQ-rank* (the maximum number of conjuncts ψ_i in any CQ-shaped subformula $\exists \boldsymbol{x}.\, \bigwedge_i \psi_i$ for non-empty $\boldsymbol{x}$).

The proof of Theorem 4 is by induction on $|\theta|$, and constructs localized 2-way automata for subformulas of θ. For GNFP subformulas, it is known from [BtCCV15] how to construct 2-way automata:[1]

Lemma 1 ([BtCCV15]) *Let $\psi(\boldsymbol{y}, \boldsymbol{Z}) \in \text{GNFP}^k[\sigma']$ in normal form. Then for every local assignment $\boldsymbol{b}/\boldsymbol{y}$, we can construct a 2-way alternating parity tree automaton $\mathcal{A}_\psi^{\boldsymbol{b}/\boldsymbol{y}}$ such that for all consistent $\tilde{\sigma}'_k$-trees $(\mathcal{T}, \boldsymbol{Z}^{\rightarrow})$ and for all nodes $w \in \text{dom}(\mathcal{T})$ with $\boldsymbol{b} \subseteq \overline{w}$,*

$$\mathfrak{D}(\mathcal{T}), [w, \boldsymbol{b}], \boldsymbol{Z} \models \psi \;\text{ iff }\; \mathcal{A}_\psi^{\boldsymbol{b}/\boldsymbol{y}} \text{ accepts } (\mathcal{T}, \boldsymbol{Z}^{\rightarrow}) \text{ from } w.$$

There is a polynomial function f independent of ψ such that the number of states for all such localized automata is at most $N := f(m) \cdot 2^{f(klr)}$ where $m = |\psi|$, $l = |\text{const}(\sigma')|$, and $r = \text{rank}_{\text{CQ}}(\psi)$. The number of priorities in each automaton is linear in $|\psi|$. The overall size is at most exponential in $|\sigma'| \cdot N$.

We use these automata for GNFP as building blocks for our GNFP-UP construction. Recall that pdepth 0 formulas can always be viewed as GNFP formulas. We can also transform parts of the formula into GNFP formulas over a slightly different signature.

For this purpose, given $\psi \in (\text{GNFP-UP})^k[\sigma]$ with $\text{params}(\psi) \subseteq \boldsymbol{z}$, define the *augmented signature* $\sigma_{\boldsymbol{z},\psi}$ to be the signature σ together with additional constants $z \in \boldsymbol{z}$ and subformula predicates F_η for subformulas η with $\text{params}(\eta) \subseteq \boldsymbol{z}$. For such η, the arity of F_η is usually $|\text{free}(\eta) \setminus \text{params}(\eta)|$; in the special case that η is a fixpoint formula, then the arity of F_η is the arity of this fixpoint predicate. Then we can transform the outer part of a GNFP-UP formula to a GNFP formula over this augmented signature. We can only perform this transformation on the outer part of the formula that uses the same set of parameters. Consider $\eta \in (\text{GNFP-UP})^k[\sigma]$ with $\text{free}(\eta) \subseteq \boldsymbol{y}\boldsymbol{z}$ and $\text{params}(\eta) \subseteq \boldsymbol{z}$. We define $\text{transform}_{\boldsymbol{z}}(\eta) \in \text{GNFP}^k[\sigma_{\boldsymbol{z},\eta}]$ inductively as follows:

$$\begin{aligned}
&\text{transform}_{\boldsymbol{z}}(R\,\boldsymbol{t}) := R\,\boldsymbol{t} \qquad \text{transform}_{\boldsymbol{z}}(Y\,\boldsymbol{t}) := Y\,\boldsymbol{t}\\
&\text{transform}_{\boldsymbol{z}}(\alpha \wedge \neg\phi) := \alpha \wedge \neg \text{transform}_{\boldsymbol{z} \cap \text{free}(\phi)}(\phi)\\
&\text{transform}_{\boldsymbol{z}}([\mathbf{lfp}^{\boldsymbol{z}'}_{X,\boldsymbol{x}}.S](\boldsymbol{t})) :=\\
&\quad \begin{cases} F_{[\mathbf{lfp}^{\boldsymbol{z}'}_{X,\boldsymbol{x}}.S](\boldsymbol{t})}\,\boldsymbol{t} \text{ if there is } \phi_j \in S \text{ with } \text{params}(\phi_j) \nsubseteq \boldsymbol{z} \\ [\mathbf{lfp}_{X,\boldsymbol{x}}.S'](\boldsymbol{t}) \text{ o.w.} \end{cases}\\
&\quad \text{where } S' \text{ is the result of applying } \text{transform}_{\boldsymbol{z}} \text{ to each } \phi_j \in S\\
&\text{transform}_{\boldsymbol{z}}(\textstyle\bigvee_i \exists \boldsymbol{x}_i.\, \bigwedge_j \psi_{ij}) :=\\
&\quad \begin{cases} F_{\bigvee_i \exists \boldsymbol{x}_i.\, \bigwedge_j \psi_{ij}}\,\boldsymbol{y} \text{ if there is } i,j \text{ such that } \text{params}(\psi_{ij}) \nsubseteq \boldsymbol{z} \\ \bigvee_i \exists \boldsymbol{x}_i.\, \bigwedge_j \text{transform}_{\boldsymbol{z} \cap \text{free}(\psi_{ij})}(\psi_{ij}) \text{ o.w.} \end{cases}
\end{aligned}$$

The GNFP formula obtained using this transformation is "equivalent" to the GNFP-UP formula, under the assumption that the additional predicates in the augmented signature are interpreted in the expected way. It does not increase the width, CQ-rank, or the size of the formula.

[1] [BtCCV15] used a different encoding of the tree-like models, but the adaptation to the encoding here requires only minor technical changes.

If the transformation applied to η only introduces $F_{\eta'}$ for strict subformulas η' of η, then we say the transformation is *helpful*. In a helpful transformation, all occurrences of these new predicates $F_{\eta'}$ appear under a guard of $\mathrm{free}(\eta') \setminus \mathrm{params}(\eta')$. Another way to understand the parameter depth is to say that the parameter depth measures the number of unhelpful breakpoints we reach as we try to transform the entire formula using this operation.

The main idea of the construction, is to transform the outer part of the formula into a GNFP formula. If the transformation is helpful, we can then use the GNFP automaton for the outer part of the formula, and plug-in inductively defined automata checking the subformulas. When this is not possible, we must use different techniques which result in an exponential blow-up at these stages.

This leads to our main result for this work:

Theorem 5 *Satisfiability for $\theta \in$ GNFP-UP is decidable in $(pdepth(\theta) + 2)$-ExpTime.*

2.1.9 Expressivity examples

In this subsection, we give some examples showing that GNFP-UP subsumes and extends a wide range of logics. This provides evidence of its power, and explains some of the good properties of these previously-studied logics.

To help study the power of GNFP-UP, we first define a way to measure how the parameters are used. Roughly speaking, the *parameter depth* is the maximum number of nested parameter changes. We define $\mathrm{pdepth}_{\boldsymbol{z}}(\eta)$ inductively as follows:

$$\begin{aligned}
&\mathrm{pdepth}_{\boldsymbol{z}}(R\,\boldsymbol{t}) = \mathrm{pdepth}_{\boldsymbol{z}}(Y\,\boldsymbol{t}) := 0\\
&\mathrm{pdepth}_{\boldsymbol{z}}(\alpha \wedge \neg\phi) := \mathrm{pdepth}_{\boldsymbol{z}\cap\mathrm{free}(\phi)}(\phi)\\
&\mathrm{pdepth}_{\boldsymbol{z}}(\textstyle\bigvee_i \exists \boldsymbol{x}_i.\, \bigwedge_j \psi_{ij}) := \max_i p_i \text{ s.t.}\\
&\qquad p_i := \begin{cases} 1 + \max_j \mathrm{pdepth}_{\mathrm{params}(\psi_{ij})}(\psi_{ij}) & \text{if } \exists j.\, \mathrm{params}(\psi_{ij}) \not\subseteq \boldsymbol{z} \\ \max_j \mathrm{pdepth}_{\boldsymbol{z}}(\psi_{ij}) & \text{otherwise} \end{cases}\\
&\mathrm{pdepth}_{\boldsymbol{z}}([\mathbf{lfp}^{\boldsymbol{z}'}_{X_m,\boldsymbol{x}_m}.S](\boldsymbol{t})) :=\\
&\qquad \begin{cases} 1 + \max_{\phi_j\in S} \mathrm{pdepth}_{\mathrm{params}(\phi_j)}(\phi_j) & \text{if } \exists j.\, \mathrm{params}(\phi_j) \not\subseteq \boldsymbol{z} \\ \max_{\phi_j\in S} \mathrm{pdepth}_{\boldsymbol{z}}(\phi_j) & \text{otherwise} \end{cases}.
\end{aligned}$$

The parameter depth $\mathrm{pdepth}(\phi)$ for normal form $\phi \in$ GNFP-UP is just $\mathrm{pdepth}_{\mathrm{free}(\phi)}(\phi)$. For ϕ not necessarily in normal form, we define it to be the pdepth after converting to normal form.

Observe that a formula that does not use any parameters has pdepth 0. Even a formula that does use parameters can have pdepth 0 if all of its parameters actually come from free variables of the formula. This is because parameters like this can be viewed as constants since they have a fixed interpretation in any structure. Because of this, if $\phi \in$ GNFP-UP$[\sigma]$ with $\mathrm{pdepth}(\phi) = 0$ and $\mathrm{params}(\phi) = \boldsymbol{z}$, then we can view ϕ as a GNFP formula without parameters, over the signature σ extended with extra constants $\boldsymbol{z}$.

In general, the pdepth increases when we pass through a subformula that introduces more parameters. This can happen when passing through existential quantification that introduces a variable that is later used as a parameter (see the third case in the pdepth definition), or it can happen when passing through a fixpoint definition that introduces a fixpoint variable that is later used as a parameter (see the fourth case).

Later, we will see that the parameter depth is the major factor impacting the complexity of satisfiability testing.

Traditional guarded logics GNFP-UP subsumes all of the previously mentioned guarded logics (and their fixpoint extensions), including GFP sentences, UNFP formulas, and GNFP formulas. Unsurprisingly, these traditional guarded logics without parameters can be expressed as GNFP-UP formulas of pdepth 0.

Navigational queries There are several languages for navigational queries in graph databases, where the signature σ consists only of binary and unary relations. For these languages, a regular expression E over symbols R, R^- coming from binary relations $R \in \sigma$ can be seen as defining a *navigation relation* that holds for (x, y) exactly when there is some path between x and y matching E. A *conjunctive 2-way regular path query* (C2RPQ) [CDLV00] is just a CQ over such expressions. This notion of C2RPQ has not only been used for expressing queries but also inside rule languages and logics. We studied in [ABBV18] a problem of certain query answering for a notion of transitivity in the rules. This fragment can be captured by GNFP-UP and the techniques developed here can be directly for this work. Finally, C2RPQ have been integrated to a logic r-UNF [JLMS18] to subsume different works on reasoning with constraints integrating C2RPQ inside the rules and C2RPQ. They developed similar techniques as the one presented in this work. Interestingly, we could reproof their upper bound by noticing that the regular path queries can be captured by efficient tree automata.

Example 2 *Consider some* C2RPQ *over signature σ. Let $\Sigma := \{R, R^- : R \text{ is a binary relation in } \sigma\}$.*

Given a regular expression E over Σ, we can capture the navigation relation defined by it using a GNFP-UP *formula E'. We start with a finite state automaton $\mathcal{A} = \langle \Sigma, Q, q_0, \Delta, F \rangle$ for E and write a* GNFP-UP$[\sigma]$ *formula with simultaneous fixpoints $E'(x, y) := [\mathbf{lfp}^{y}_{X_0,x} \ . \ S](x)$ which has a second-order variable X_i for each state $q_i \in Q$, and the equation for the i-th component X_i, x_i in S captures the possible transitions from state q_i:*

$$\bigvee_{(q_i,T,q_j)\in\Delta} \exists z.(\chi_T(x_i, z) \wedge X_j\, z) \ \vee \ \begin{cases} x_i = y & \text{if } i \in F \\ \bot & \text{if } i \notin F \end{cases}$$

where $\chi_T(x_i, z)$ is Rx_iz if $T = R$ and Rzx_i if $T = R^-$.

Once we have E' in GNFP-UP *for each regular expression E appearing in the* C2RPQ*, it is easy to translate into* GNFP-UP *by replacing each $E(x, y)$ in the* C2RPQ *by $E'(x, y)$. These* GNFP-UP *formulas have parameter depth 1: the* GNFP-UP *formula $E'(x, y)$ for each regular expression predicate $E(x, y)$ has pdepth 0; when these are substituted in the CQ, the resulting formula has pdepth at most 1 since the existential quantification may introduce variables that are used as parameters in the inner formulas.* GNFP-UP *can also express unions of* C2RPQs*.*

We can in fact replace regular expressions in C2RPQs by a variant of *propositional dynamic logic* (PDL). PDL consists of programs (defining binary relations within a labelled graph) and tests (defining unary relations within a graph) defined by mutual recursion. Programs contain all binary relation symbols and are closed under concatenation, union, and Kleene star. Tests contain all unary relation symbols and are closed under boolean operations. Given a test t, we can define a program $t?$ that returns pairs (x, x) such that x is in the unary relation defined by t, and given a program P we can form a test $\langle P \rangle$, defining a relation consisting of pairs (u, u) such that there exists v with (u, v) in the language described by P. We let CQPDL denote the language of *conjunctive queries where binary relations can be* PDL *programs*. Clearly, this subsumes C2RPQs, and it also subsumes extensions defined in the description logic literature [BCOS14]. If P is restricted to be a traditional regular expression, then the corresponding GNFP-UP formula for $\langle P \rangle$ has pdepth at most 1. By writing expressions with more complicated nesting of these tests, however, these formulas can reach higher parameter depth levels.

Fragments of Datalog Datalog is a syntax for expressing the negation-free fragment of least fixpoint logic. It is heavily used to express database queries that involve some form of recursion. We argue that all the previously defined fragments of Datalog that have decidable static analysis problems are contained in GNFP-UP.

Formally, a *Datalog query* is specified by

$$\Pi = \langle \mathrm{EDB}^{\Pi}, \mathrm{IDB}^{\Pi}, \mathrm{Rules}^{\Pi}, \mathrm{goal} \rangle$$

where the extensional predicates EDB^{Π} and intensional predicates IDB^{Π} are disjoint sets, Rules^{Π} consists of formulas of the form $R(x_1 \dots x_n) \leftarrow \psi(\boldsymbol{xy})$ where R is an IDB predicate and ψ is a conjunction of atoms, and goal is a distinguished member of IDB^{Π}. Given some structure $\mathfrak{A}$ we can evaluate goal in the structure obtained from $\mathfrak{A}$ by firing the rules of Π until a fixpoint has been reached. For a structure $\mathfrak{A}$ and query Π we let $\Pi(\mathfrak{A})$ be the value of the predicate goal so obtained. A *boolean* Datalog query is one where the goal predicate is 0-ary, and hence the query defines a boolean function on input structures.

Monadic Datalog restricts the IDBs to have arity 1. In this case, it is possible to express the query using a UNFP formula with simultaneous fixpoints without parameters. *Frontier-guarded Datalog* allows the use of intensional predicates with unrestricted arities, but for each rule $R(x_1 \dots x_n) \leftarrow \psi(\boldsymbol{xy})$, the variables $x_1 \dots x_n$ in the head of the rule, must appear in a single EDB atom appearing in the body ψ. This subsumes monadic Datalog since the single head variable in the monadic Datalog rules can be trivially guarded. Frontier-guarded Datalog can be expressed in GNFP. No parameters are necessary, so the parameter depth is 0.

The *flag and check queries* introduced in [RK13, BKR15] are based on fragments of Datalog queries that have been shown to have decidable analysis problems.

One family consists of the *monadically defined queries* (MQs). These are of the form $\exists \boldsymbol{y}.\Pi$ where Π is a Monadic Datalog query, the goal predicate is nullary, and the rules use special symbols $\boldsymbol{z}$. The answers to the query are (projections of) assignments to the special symbols for which the corresponding Monadic Datalog query evaluates to true. The idea is that the special symbols serve as flags for potential answers to the query, and the Datalog query checks if the flags mark actual answers.

Example 3 (based on [BKR15]) *The transitive closure of a binary relation R can be expressed by the* MQ Π *with special symbols z_1, z_2 where Π is*

$$U(y) \leftarrow R\,z_1 y \qquad\qquad U(y) \leftarrow Ux \wedge R\,xy \qquad\qquad \mathit{hit}() \leftarrow U z_2\ .$$

The answer to the query would consist of all pairs (a_1, a_2) for which the rules imply hit under the standard Datalog semantics, when interpreting z_1 as a_1 and z_2 as a_2.

In UNFP-UP*, this is $\psi(z_1, z_2) := [\mathbf{lfp}^{z_1,z_2}_{hit,\varnothing}.S]()$ where*

$$S := \begin{cases} U, y & := \ R\,z_1 y \vee \exists x.(Ux \wedge R\,xy) \\ hit, \varnothing & := \ U z_2 \end{cases}$$

Notice that the special symbols become parameters. Because it is a nullary predicate, the fixpoint is nullary too.

We can translate an arbitrary MQ $\exists \boldsymbol{y}.\Pi$ with special symbols $\boldsymbol{z}$ using a similar method: the monadic Datalog query becomes a simultaneous fixpoint ψ' in GNFP-UP, with the special symbols $\boldsymbol{z}$ as parameters, and the special nullary hit predicate as the goal predicate. The MQ itself can then be written in GNFP-UP

as $\exists \boldsymbol{y}.\psi'(\boldsymbol{z})$. The resulting formula has pdepth at most 1, since $\psi'(\boldsymbol{z})$ has pdepth 0, and $\exists \boldsymbol{y}.\psi'(\boldsymbol{z})$ may project some of the parameters (the previous example only has pdepth 0 because there is no such projection).

In [RK13], they also consider a nested version of these flag and check queries. An m-*nested* MQ is one where the monadic Datalog query is allowed to use predicates defined by $(m-1)$-nested MQs in the rule bodies (but these predicates cannot be used as guards); a 1-nested MQ is just the MQ defined above. In general, we can translate an m-nested MQ query into a GNFP-UP formula of pdepth at most m. [BKR15] defines other variants of flag and check queries; all of them can be similarly captured in GNFP-UP.

A Datalog query Π_1 is *contained in* a Datalog query Π_2 if for all input structures $\mathfrak{A}$, $\Pi_1(\mathfrak{A}) \subseteq \Pi_2(\mathfrak{A})$. Similarly given a sentence ϕ in some logic, we say Datalog query Π_1 is *contained in* Π_2 *relative to* ϕ if $\Pi_1(\mathfrak{A}) \subseteq \Pi_2(\mathfrak{A})$ for all $\mathfrak{A}$ satisfying ϕ. GNFP-UP can express the Datalog queries in the fragments above. Moreover, since it is closed under boolean combinations for sentences, it can also express containment of two boolean queries within each fragment, and containment relative to sentences ϕ that are expressible in GNFP-UP.

2.2 Lower bounds

2.2.1 Introduction

In this section, we study two problems of interest, with strong connections: containment of monadic datalog, validity problems on trees, In [BBGS20], we established a technique based on the reduction of tree validity to MDL containment that allows us to push the 2-ExpTime lower bound to the MDL containment problem. We recall here, the main results. The encoding to reduce the validity problem into the containment problem can be found in [BBGS20]. We first give the formal definitions of Datalog and containment.

2.2.2 Definitions

Datalog A *datalog program* [AHV95] over σ consists of:

1. A set of rules of the form $A \leftarrow \phi$, where ϕ is a conjunction of atoms over σ, and A is an atom over σ. We say A is the *head* and ϕ the *body* of the rule. We require that every variable occurring in the head of a rule r also occurs in its body.
2. A distinguished predicate goal of σ which occurs in the head of a rule, referred to as the *goal predicate*.

The relational symbols that do not occur in the head of any rule are the *input* or *extensional predicates*, while the others are *intensional predicates*. Similarly, the *extensional* (resp., *intensional*) signature of a program is the set of *extensional* (resp., *intensional*) predicates used by the program. Monadic datalog (MDL) denotes the sublanguage where all intensional predicates are monadic (unary), except for the goal predicate which can be either unary or nullary (in the latter case, we say that the program is *Boolean*).

For a datalog program P, an intensional predicate R of P, and an instance I interpreting the input predicates, we define the evaluation of R on I, denoted $R(I)$, as the union of the relations $P_k(R, I)$ defined via the following process, starting with $P_0(R, I) = \varnothing$:

1. Let I^k be the expansion of I with $P_k(R, I)$ for all intensional R.

2. If r is a rule with $R(x_1 \dots x_l)$ in the head, $\boldsymbol{w}$ the variables of r not present in the head, and $\phi(\boldsymbol{x}, \boldsymbol{w})$ the body of r, let $P_{k+1}(r, I)$ be defined by: $\{\boldsymbol{c} \in \mathrm{dom}(()I)^l \mid I^k \models \exists \boldsymbol{w}\, \phi(\boldsymbol{c}, \boldsymbol{w})\}$ where $\mathrm{dom}(()I)$ is the active domain of I.
3. Let then $P_{k+1}(R, I)$ denote the union of $P_{k+1}(r, I)$ over all r with R in the head.

Finally, the *query result* of P on I, denoted $P(I)$, is the evaluation of the goal predicate of P on I. We often assume P is Boolean, in which case the result of the program on I is the Boolean "true" iff goal() holds in I, and we simply say that I is a *model of* P or that I *satisfies* P. We alternatively refer to a *datalog query*, rather than to a datalog program, to emphasize that we are only interested in the evaluation on the goal predicate.

Under these semantics, it is easy to check that any UCQ can be transformed in linear time into an equivalent MDL query, that does not involve any intensional predicate apart from goal.

Containment The main problem we deal with in this work is the classical notion of *query containment* [AHV95].

Definition 1 *Let Q and Q' be two queries over a signature σ. We say Q is* contained *in Q', denoted $Q \sqsubseteq Q'$, if, for any instance I over σ, $Q(I) \subseteq Q'(I)$.*

Above we have defined containment in terms of the evaluation of Q over *all instances, finite and infinite*. However a simple (and well-known) argument shows that this coincides with containment when only finite instances are considered. If there is an instance I and tuple t such that $t \in Q(I)$, $t \notin Q'(I)$, then the fact that $t \in Q(I)$ is guaranteed by a finite collection of facts I_0 in I. Thus I_0 witnesses that Q is not contained in Q' over finite instances. Given this equivalence, *throughout this work we will assume that instances are finite*. For finite instances I, there will be a finite k such that the evaluation of datalog Q will be $P_{k+1}(\text{goal}, I)$ for goal the goal predicate.

Example 4 *Consider the following MDL program P that determines whether there is a path in a graph G from a node marked with the unary predicate S to one marked with the unary predicate T:*

$$\begin{aligned} \mathit{goal}() &\leftarrow T(x) \wedge \mathit{Reachable}(x) \\ \mathit{Reachable}(y) &\leftarrow G(x, y) \wedge \mathit{Reachable}(x) \\ \mathit{Reachable}(x) &\leftarrow R(x) \end{aligned}$$

Now consider the UCQs:

$$\begin{aligned} Q_1 &: \exists x \exists y\, R(x) \wedge G(x, y) \wedge T(y) \\ Q_2 &: (\exists x\, R(x) \wedge T(x)) \vee (\exists x' \exists y'\, G(x', y')) \end{aligned}$$

We have that $P \not\sqsubseteq Q_1$ but $P \sqsubseteq Q_2$: indeed, if I is the instance made of the facts $R(a)$ and $T(a)$, I is a model of P, but not a model of Q_1. And in any model of P, either the first rule defining $\mathit{Reachable}$ is used, and then the second disjunct of Q_2 holds, or only the second rule defining $\mathit{Reachable}$ is used, and then the first disjunct of Q_2 holds.

2.2.2.1 Trees and Tree Validity

Our results for the lower bound of datalog containment have a tight connection with "universality" or "validity" problems for queries over trees: given a schema describing a set of trees and a Boolean query over trees, does every tree satisfy the query? There are many variants of the problem, depending on the exact signature of trees used. We will thus define several signatures below.

Let Λ be a finite non-empty set of labels. We will consider the settings of both binary and unranked trees. Many of our lower bounds will work in the restricted setting of binary trees. For binary trees with labels from Λ, the following signature is natural.

The *relational signature of ordered, labeled, binary trees*, denoted $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2}}$, contains the binary predicates FirstChild, SecondChild, unary Root, Leaf predicates, and Label_α predicates for all $\alpha \in \Lambda$.

A tree T over $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2}}$ is a relational instance such that:

1. the non-empty Label^T_α's for $\alpha \in \Lambda$ form a partition of $\mathrm{dom}(()T)$ (one can thus talk about the *label* of a node n, which is the $\alpha \in \Lambda$ such that $n \in \mathsf{Label}^T_\alpha$);
2. $\mathsf{FirstChild}^T$ and $\mathsf{SecondChild}^T$ are one-to-one partial mappings with the same domain (the set of *internal nodes*), whose complement is Leaf^T (the set of *leaves*), and with disjoint ranges;
3. the inverses of $\mathsf{FirstChild}^T$ and $\mathsf{SecondChild}^T$ are one-to-one partial mappings;
4. $\exists x\, \mathsf{FirstChild}(x,x) \vee \mathsf{SecondChild}(x,x)$ does not hold;
5. Root^T contains exactly one element (the *root* r of T), and the following formula does not hold for r: $\exists x\, \mathsf{FirstChild}(x,r) \vee \mathsf{SecondChild}(x,r)$.

We denote as $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child}}$ (resp., $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child},\mathsf{Child}^?}$) the relational signature containing all the relations of $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2}}$ together with a binary Child relation (resp., binary Child and $\mathsf{Child}^?$ relations). A *tree* T over $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child}}$ is a relational instance that verifies the same axioms as a tree over $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2}}$, where Child^T is the disjoint union of $\mathsf{FirstChild}^T$ and $\mathsf{SecondChild}^T$. A tree over $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child},\mathsf{Child}^?}$ has the additional requirement that $\forall x \forall y\ \mathsf{Child}^?(x,y) \leftrightarrow (\mathsf{Child}(x,y) \vee x = y)$ holds.

Note that we omit the label alphabet Λ from notation such as $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2}}$ for readability. Our upper bound results concerning $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2}}$ and $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child}}$ will hold for any label set Λ, while in our lower bounds we will usually show hardness for any label set of size at least 2.

The *relational signature of unordered, labeled, unranked trees*, denoted $\mathcal{S}^{\text{unranked}}_{\mathsf{Child}}$, is made out of the binary predicate Child together with the unary Root, Leaf, and Label_α. A tree over $\mathcal{S}^{\text{unranked}}_{\mathsf{Child}}$ is a relational instance such that:

1. the non-empty Label^T_α's for $\alpha \in \Lambda$ form a partition of $\mathrm{dom}(()T)$;
2. Child^T is a tree in the usual sense, whose root is the only element of Root^T and whose leaves are exactly the elements of Leaf^T.

We sometimes consider as special cases trees formed of a single node (i.e., trees such that $|\mathrm{dom}(()T)| = 1$); we call them *root-only* trees.

Example 5 *Consider the simple abstract tree with a root labeled α and two children labeled β and γ respectively represented here with the root at the top:*

In the four signatures introduced, this tree can be represented as the following collection of facts:

all four signatures	$\mathsf{Root}(r), \mathsf{Label}_\alpha(r), \mathsf{Label}_\beta(f), \mathsf{Leaf}(f), \mathsf{Label}_\gamma(s), \mathsf{Leaf}(s)$
$\mathcal{S}^{\mathrm{bin}}_{\mathsf{Ch1},\mathsf{//}}$	$\mathsf{FirstChild}(r,f), \mathsf{SecondChild}(r,s)$
$\mathcal{S}^{\mathrm{bin}}_{\mathsf{Ch1},\mathsf{//},\mathsf{Child}}$	$\mathsf{FirstChild}(r,f), \mathsf{SecondChild}(r,s), \mathsf{Child}(r,f), \mathsf{Child}(r,s)$
$\mathcal{S}^{\mathrm{bin}}_{\mathsf{Ch1},\mathsf{//},\mathsf{Child},\mathsf{Child}^?}$	$\mathsf{FirstChild}(r,f), \mathsf{SecondChild}(r,s), \mathsf{Child}(r,f), \mathsf{Child}(r,s), \mathsf{Child}^?(r,r),$ $\mathsf{Child}^?(f,f), \mathsf{Child}^?(s,s), \mathsf{Child}^?(r,f), \mathsf{Child}^?(r,s)$
$\mathcal{S}^{\mathrm{unranked}}_{\mathsf{Child}}$	$\mathsf{Child}(r,f), \mathsf{Child}(r,s)$

We will consider several methods for defining families of trees, in particular tree automata and document type definitions (DTDs). We define them formally in the binary case.

Definition 2 *A* nondeterministic tree automaton on binary trees *(or $BNTA$) over finite alphabet Λ is of the form $(\Omega, \Delta_0, \Delta, F)$, where Ω (the control states) is a finite set, $\Delta_0 \subset \Lambda \times \Omega$, $\Delta \subset \Omega^2 \times \Lambda \times \Omega$, and $F \subset \Omega$. A run ρ of a $BNTA$ over a Λ-labeled binary tree is an assignment of states to nodes. A run is accepting if for all leaves l labeled with $\alpha \in \Lambda$, $(\alpha, \rho(l)) \in \Delta_0$; the root is assigned a state in F; and if n has left and right children n_1 and n_2 respectively and label α, then $(\rho(n_1), \rho(n_2), \alpha, \rho(n)) \in \Delta$.*

A deterministic tree automaton on binary trees *($BDTA$) over Λ is a $BNTA$ in which for every $(q_1, q_2, a) \in \Omega^2 \times \Lambda$, there is at most one q such that $(q_1, q_2, a, q) \in \Delta$.*

The set of all binary trees having an accepting run of $BNTA$ A is the *language* of A, noted $L(A)$. Such a language is then said to be *regular.*

A nondeterministic tree automaton over ranked trees (NTA_{Rk}) is defined similarly, but with $\Delta \subset \bigcup_{i \leqslant r} \Omega^i \times \Lambda \times \Omega$ for some r. Such an automaton expects trees in which the outdegree of each vertex is at most r. The notion of deterministic tree automaton over ranked trees (DTA_{Rk}), the language of such an automaton, and regularity of a language of ranked trees is defined analogously to above.

We will also make use of the corresponding notion of nondeterministic tree automaton over unranked trees, NTA_{Unr} and of a regular language for unranked trees, see [GS97]. We will not need to know the definition of a NTA_{Unr}, since most of the results involving NTA_{Unr} will come from prior work. We will use the following simple facts relating NTA_{Unr} to their ranked counterparts:

- A $BNTA$, and more generally an NTA_{Rk}, is a special case of a NTA_{Unr}, since we can enforce a restriction on the rank with an automaton.
- A witness for the non-emptiness of an NTA_{Unr} A can always be taken to have rank polynomial in the size of A. This can be shown by just "trimming" a witness.

A *DTD for binary trees over* Λ ($BDTD$) is a pair (d, l_0) where d is a function from Λ to $2^{(\Lambda \times \Lambda) \cup \{\epsilon\}}$ giving the constraints over the labels of the children of a node; l_0, an element of Λ, is the root label. A binary tree t is *accepted by a $BDTD$* (d, l_0) if (i) for any node n labeled a, if n is a leaf then $\epsilon \in d(a)$ and, otherwise, if b and c are the labels of the first and second children of n then $(b, c) \in d(a)$; (ii) the root of t is labeled by l_0. The set of all trees accepted by a $BDTD$ D is the *language* of D, noted $L(D)$. The standard notion of a DTD [Nev02] is for unranked trees. For clarity and to keep a uniform notation we refer to these as $UDTD$s. For these, d is a function from Λ to regular expressions over Λ. The notion of acceptance of an unranked tree by a $UDTD$ is standard, and we will not have need of it here. We will need the well-known and simple fact that $BDTD$s can be turned into $BNTA$s accepting the same language in linear time, and similarly for the unranked case.

Definition 3 *A query on one of the signatures is* valid *over an automaton or DTD appropriate for that signature (e.g., $BNTA$ or $BDTD$ for a signature for binary trees) if for all trees that satisfy the schema, the query returns true. A query is* valid *with respect to a set of node labels if the query returns true on all trees over that set of node labels.*

2.2.3 Results on Tree Validity Problems

First, we overview the results on tree validity that are either explicitly in prior work, can be derived with little effort from prior work, or are easy to derive directly.

We first note that validity over all trees is tractable for CQs over $\mathcal{S}^{\text{unranked}}_{\mathsf{Child}}$, $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2}}$, $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child}}$, and $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child},\mathsf{Child}^?}$. Apart from this very special case, the best upper bound known for the tree validity problems we consider is 2-ExpTime. Indeed, in [BMS08, Theorem 11], validity of a query over $\mathcal{S}^{\text{unranked}}_{\mathsf{Child}}$ with respect to an NTA_{Unr} was shown to be in 2-ExpTime for CQs. This 2-ExpTime upper bound actually holds for all considered problems on tree validity. For most of our signatures, such a bound can be obtained as follows. Convert the UCQ Q to an exponential-sized tree automaton (e.g., $BNTA$ for signatures appropriate to binary trees) A_Q in exponential time. See, for example, [ABS15b, Proposition B.1] for this conversion. Then using standard automata techniques [CDG$^+$02] we can determinize A_Q in exponential time, complement it, and intersect it with the automaton representing the schema in polynomial time. Finally, we can test the resulting automaton for emptiness in polynomial time. In the case of $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child},\mathsf{Child}^?}$, we first convert a UCQ Q over $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child},\mathsf{Child}^?}$ to a positive existential query Q' over $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child}}$. The query Q' can be converted in exponential time to an alternating automaton over trees. The construction is a standard induction: the atoms are converted to automata that work over trees with the free variables annotated on the tree. Conjunction and disjunction are done using the closure properties of alternating automata, which allow positive Boolean combinations in the transition function. Existential quantification can be assumed to be outermost, and requires projecting out the annotations. This can be done by converting the alternating automata to a non-deterministic automata in exponential time; for non-deterministic automata the projection step is straightforward. Emptiness of alternating automata can be checked in exponential time [CDG$^+$02], which gives the 2-ExpTime bound.

Let us now discuss existing lower bounds. The validity problem with respect to DTDs over $\mathcal{S}^{\text{unranked}}_{\mathsf{Child}}$ has been studied in [BMS13]. [BMS13, Theorem 12] shows that the validity problem for $\mathcal{S}^{\text{unranked}}_{\mathsf{Child}}$ is ExpTime-hard for *child-only tree-pattern queries*. Given that one can convert these straightforwardly to CQs, we obtain ExpTime-hardness of the validity problem for CQs (and thus UCQs) with respect to $UDTD$ and NTA_{Unr}.

Inspection of prior work easily shows the lower bound carries over to the $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child}}$ signature and, consequently, to the $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child},\mathsf{Child}^?}$ signature, as we now explain. Theorem 12 of [BMS13] relies on Theorem 11 in the same paper, whose proof involves a reduction from finding a winning strategy in a game on tiling systems [Chl86, Rectangle Tiling Game]. Critically, the number of possible moves in this strategy is bounded, by the number of different tiles, which is fixed. Thus the trees involved in the hardness proof are actually ranked. Now, we use a standard encoding of b-ranked trees as binary trees where every node n with at most b children is replaced with a binary subtree of height exactly $\lceil \log_2(b) \rceil$ whose leaves are the children of n. This means that, in the CQ, we replace every Child atom with a chain of $\lceil \log_2(b) \rceil$ child atoms. In the DTD, we enumerate the bounded number of possible words for the labels of children of every node label, and choose fresh node labels for every such possible word and every position in the binary tree encoding the unranked Child relation. It then becomes easy to transform the $UDTD$ on unranked trees into a $BDTD$ on the encoded binary trees.

Table 2.1: Summary of results on the complexity of tree validity of CQs and UCQs, over various tree signatures and with respect to DTDs, tree automata, or all trees

;
lower bounds w.r.t. tree automata are transferred to lower bounds w.r.t. DTDs

Signature	CQ (DTD or tree automaton)	UCQ (DTD or tree automaton)	CQ (all trees)	UCQ (all trees)
$\mathcal{S}^{\text{unranked}}_{\mathsf{Child}}$	ExpTime-complete	ExpTime-complete	P	ExpTime-complete
$\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2}}$	in ExpTime	ExpTime-complete	P	ExpTime-complete
$\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child}}$	ExpTime-complete	ExpTime-complete	P	ExpTime-complete
$\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child},\mathsf{Child}^?}$	2-ExpTime-complete	2-ExpTime-complete	P	2-ExpTime-complete

Table 2.2: Summary of results on the complexity of query containment.

Query containment setting	Containment	Upper bound	Lower bound
MDL in MDL	2-ExpTime-complete	[CGKV88, Thm 7.2]	[BBGS20]
MDL in UCQ	2-ExpTime-complete	[CGKV88, Thm 7.2]	[BBGS20]
MDL in CQ	2-ExpTime-complete	[CGKV88, Thm 7.2]	[BBGS20]
Datalog in UCQ	2-ExpTime-complete	[CV97, Thm 5.12]	[CV97, Thm 5.15]

In [BBGS20], we establish tight complexity bounds for the validity of CQs and UCQs over all four tree signatures introduced ($\mathcal{S}^{\text{unranked}}_{\mathsf{Child}}$, $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2}}$, $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child}}$, and $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child},\mathsf{Child}^?}$) with respect to DTDs, tree automata, and over all trees. The results are summarized in Table 2.1. The results on validity over all trees refer to the combined complexity of the problem that takes as input both the query and label set, determining if the query is valid for that label set. There is one exception where a tight bound is still open: the case of CQs over $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2}}$ with respect to $BDTD$s or $BNTA$s. In all other cases (beyond the trivial P case of CQs over all trees), we establish 2-ExpTime-completeness (for $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child},\mathsf{Child}^?}$) and ExpTime-completeness (for the other three signatures).

2.2.4 Results on Containment for MDL and Access Constraints

The upper bound of containment of MDL queries in UCQs is in 2-ExpTime by [CGKV88] (indeed, this holds also for containment of two MDL queries [CGKV88] or for Datalog in UCQs [CV92, CV97]). In [BBGS20], we show that this problem is 2-ExpTime-hard, thus obtaining a tight characterization of its complexity.

Table 2.2 summarizes these results and provides references to the corresponding theorems.

The reduction of validity of trees to obtain a lower bound has also been adapted and used for proving other lower bounds for logic satisfiability [BBV16], containment of other Datalog sublanguages [BKR15] and certain query answering [BL16, BMMP16].

Chapter 3

From Trees to Logic

Evaluation of MSO over trees has brought a lot of attention over the past forty years due to the interesting properties which this language has. It has an efficient evaluation which linear in the size of the tree and this property extends to more complex results such as counting, probabilistic evaluation and enumeration. All these problems have been well studied. In this chapter, I propose to approach these problems through the lens of circuits. I focus on the problems of enumeration. During our different studies, we introduced different kinds of circuits to represent the answers of a MSO query to obtain the desired results. In the rest of the chapter, I will focus on a particular representation called multivalued circuits, this kind of circuit is particularly useful for obtaining tight results for the different enumeration problems for MSO formulas with first-order free variables. Finally, I focus on the equivalent definition of MSO formula based on Tree automata.

After presenting the preliminaries, I present the construction of the multivalued circuit for tree automata which is the base of different results. I then state the main results for enumeration and rank enumeration and the main bricks of the algorithm.

3.1 Preliminaries

In order to define the notion of answer of queries, we need to define the notion of assignment.

Assignments. For two finite sets D of *values* and X of *variables*, an *assignment on domain D and variables X* is a mapping from X to D. We write D^X the set of such assignments. We can see assignments as sets of *singleton assignments*, where a *singleton assignment* is an expression of the form $[x \to d]$ with $x \in X$ and $d \in D$.

Two assignments $\tau \in D^Y$ and $\sigma \in D^Z$ are *compatible*, written $\tau \simeq \sigma$, if we have $\tau(x) = \sigma(x)$ for every $x \in Y \cap Z$. In this case, we denote by $\tau \bowtie \sigma$ the assignment of $D^{Y \cup Z}$ defined following the natural join, i.e., for $y \in Y \backslash Z$ we set $(\tau \bowtie \sigma)(y) := \tau(y)$, for $z \in Z \backslash Y$ we set $(\tau \bowtie \sigma)(z) := \sigma(z)$, and for $x \in Z \cap Y$, we set $(\tau \bowtie \sigma)(x)$ to the common value $\tau(x) = \sigma(x)$. Two assignments $\tau \in D^Y$ and $\sigma \in D^Z$ are *disjoint* if $Y \cap Z = \emptyset$: then they are always compatible and $\tau \bowtie \sigma$ corresponds to the relational product, which we write $\tau \times \sigma$.

Given $R \subseteq D^Y$ and $S \subseteq D^Z$, we define $R \wedge S = \{\tau \bowtie \sigma \mid \tau \in R, \sigma \in S, \tau \simeq \sigma\}$: this is a subset of $D^{Y \cup Z}$. Note how, if the domain is $D = \{0, 1\}$, then this corresponds to the usual conjunction for Boolean functions, and in general we can see it as a relational join, or a relational product whenever $Y \cap Z = \emptyset$. Further, we define $R \vee S = \{\tau \in D^{Y \cup Z} \mid \tau|_Y \in R \text{ or } \tau|_Z \in S\}$, which is again a subset

of $D^{Y \cup Z}$. Again observe how, when $D = \{0, 1\}$, this corresponds to disjunction; and in general, we can see this as a relational union except that assignments over Y and Z are each implicitly completed in all possible ways to assignments over $Y \cup Z$.

3.1.1 Trees and Automata

We define words, trees, and valuations, present our automata and a homogenization lemma, and state our problem.

Trees. We work with trees that are all *rooted* and *ordered*, i.e., there is an order on the children of each node. Given a set Λ of tree labels, a Λ-*tree* T (or *tree* when Λ is clear from context) is a pair of a rooted tree (also written T) and of a *labeling function* λ that maps each node n of T to a *label* $\lambda(n) \in \Lambda$. We write $\mathrm{Leaf}(T)$ for the set of leaves of T. We abuse notation and identify T with its set of nodes, i.e., we can write that $\mathrm{Leaf}(T) \subseteq T$. An *internal node* is a node of $T \backslash \mathrm{Leaf}(T)$. All trees in this part will be *binary*, i.e., every internal node has exactly two children, which we refer to as *left* and *right* child.

When evaluating a query with *variables* $\mathcal{X}$ on a Λ-tree T, we will see its possible results as *valuations*: an $\mathcal{X}$-*valuation* of T is a function $\nu : \mathrm{Leaf}(T) \to 2^{\mathcal{X}}$ that assigns to every leaf n of T a set of variables $\nu(n) \subseteq \mathcal{X}$ called the *annotation* of n. Note that our variables are second-order, i.e., each variable can be interpreted as a set of nodes of T. We represent v concisely as *assignments*: an $\mathcal{X}$-*assignment* is a set S of *singletons* which are pairs of the form $\langle Z : n \rangle$, where $Z \in \mathcal{X}$ and $n \in \mathrm{Leaf}(T)$.

Tree Variable Automata We will write our query on trees using *tree automata* that can express, e.g., queries in *monadic second-order logic* (MSO): see [TW68] and [ABJM19, Appendix E.1].

Formally, a *tree variable automaton* TVA on binary Λ-trees for variable set $\mathcal{X}$ (or $\Lambda, \mathcal{X}$-TVA) is a tuple $A = (Q, \iota, \delta, F)$, where Q is a finite set of *states*, $\iota \subseteq \Lambda \times 2^{\mathcal{X}} \times Q$ is the *initial relation*, $\delta \subseteq \Lambda \times Q \times Q \times Q$ is the *transition relation*, and $F \subseteq Q$ is the set of *final states*. The *size* $|A|$ of A is $|Q| + |\iota| + |\delta|$. This definition only applies to *binary* Λ-trees.

To simplify notation we often see δ as a tuple of functions, i.e., for each $l \in \Lambda$ we have a function $\delta_l : Q \times Q \to 2^Q$ defined by $\delta_l(q_1, q_2) = \{q \in Q \mid (l, q_1, q_2, q) \in \delta\}$: this intuitively tells us to which states the automaton can transition on an internal node with label l when the states of the two children are respectively q_1 and q_2. Note that, following our definition of a valuation and of ι, the automaton is only reading annotations on leaf nodes.

Having fixed Λ and $\mathcal{X}$, given a Λ-tree T and an $\mathcal{X}$-valuation ν of T, given a $\Lambda, \mathcal{X}$-TVA $A = (Q, \iota, \delta, F)$, a *run* of A on T under ν is a function $\rho : T \to Q$ satisfying the following:

- For every $n \in \mathrm{Leaf}(T)$, we have $(\lambda(n), \nu(n), \rho(n)) \in \iota$;
- For every internal node n with label l and children n_1, n_2, we have $\rho(n) \in \delta_l(\rho(n_1), \rho(n_2))$.

The run is *accepting* if it maps the root of T to a state in F, and we say that A *accepts* T under ν if there is an accepting run of A on T under ν. The *satisfying valuations* of A on T is the set of the $\mathcal{X}$-valuations ν of T such that A accepts T under ν, and the *satisfying assignments* are the corresponding assignments $\alpha(\nu)$. Thus, the automaton A defines a query on Λ-trees with second-order variables $\mathcal{X}$, and its results on a Λ-tree T are the satisfying assignments of A on T.

Homogenization. It will be useful to assume a *homogenization* property on automata. Given a $\Lambda, \mathcal{X}$-TVA $A = (Q, \iota, \delta, F)$, we call $q \in Q$ a 0-*state* if there is some Λ-tree T and run ρ of A on T that maps the root of T to q under the *empty* $\mathcal{X}$-*valuation* $\nu_\varnothing$ of T defined as $\nu_\varnothing(n) := \varnothing$ for each

$n \in \mathrm{Leaf}(T)$. We call q a 1-*state* if there is some Λ-tree T and run ρ of A on T mapping the root of T to q under some *non-empty* $\mathcal{X}$-valuation, i.e., a valuation ν different from the empty valuation. Intuitively, a 0-state is a state that A can reach by reading a tree annotated by the empty valuation, and a 1-state can be reached by reading a tree with at least one non-empty annotation. In general, a state can be both a 0-state and a 1-state, or it can be neither if there is no way to reach it. We say that A is *homogenized* if every state is either a 0-state or a 1-state and no state is both a 0-state and a 1-state. We can easily make automata homogenized, by duplicating the states to remember if we have already seen a non-empty annotation:

Lemma 2 *Given a $\Lambda, \mathcal{X}$-TVA A, we can compute in linear time a $\Lambda, \mathcal{X}$-TVA A' which is homogenized and equivalent to A.*

3.1.2 Enumeration problem

Our goal is to efficiently enumerate the results of queries on trees. The inputs to the problem are the Λ-tree T and the query given as a $\Lambda, \mathcal{X}$-TVA A, with Λ the tree alphabet and $\mathcal{X}$ the variables. The output is the set of the satisfying assignments of A and T. We present an *enumeration algorithm* to produce them, which first runs a *preprocessing phase* on A and T: we compute a concise representation of the output as an *multivalued circuit*. Second, the *enumeration phase* produces each result to the query, with no duplicates, while bounding the maximal *delay* between two successive answers. Third, our algorithm must handle *updates* to T, i.e., given an edit operation on T, efficiently update the assignment circuit and index and restart the enumeration on the updated tree.

Our main result shows how to solve this problem with preprocessing linear in T and polynomial in A; with delay independent from T, polynomial in A, and linear in each produced assignment; and with update time logarithmic in T and polynomial in A.

3.1.3 Rank Enumeration

Ranking functions. Our notion of ranking functions will give a score to each assignment, but to state their properties we define them on partial assignments. Formally, a *partial assignment* is a mapping $\nu : X \to D \cup \{\bot\}$, where $\bot$ is a fresh symbol representing *undefined*. We denote by $\overline{D^X}$ the set of partial assignments on a domain D and variables X. The *support* $\mathsf{supp}(\nu)$ of ν is the subset of X on which ν is defined. We extend the definitions of compatibility, of $\bowtie$, and of disjointness, to partial assignments in the expected way.

We then consider ranking functions defined by partial assignments $\overline{D^X}$, on which we will impose *subset-monotonicity*. Formally, a (D, X)-*ranking function* w is a function[1] $\overline{D^X} \to \mathbb{R}$ that gives a score to every partial assignment. Such a ranking function induces a *weak ordering*[2] $\preceq$ on $\overline{D^X}$, with $\mu \preceq \mu'$ defined as $w(\mu) \leqslant w(\mu')$. We always assume that ranking functions can be computed efficiently, i.e., with running time that only depends on X, not D.

By a slight notational abuse, we define the score $w(\tau)$ of partial assignment $\tau \in \overline{D^Y}$ with $Y \subseteq X$ by seeing τ as a partial assignment on X which is implicitly extended by assigning $\bot$ to every $z \in X \backslash Y$. Following earlier work [DK19, TGR22, DHK22], we then restrict our study to ranking functions that are *subset-monotone* [TGR22]:

[1] As usual, when we write $\mathbb{R}$, we assume a suitable representation, e.g., as floating-point numbers.

[2] Recall that a weak ordering $\preceq$ on A is a total preorder on A, i.e., $\preceq$ is transitive and we have either $x \preceq y$ or $y \preceq x$ for every $x, y \in A$. In particular, it can be the case that two distinct elements x and y are tied, i.e., $x \preceq y$ and $y \preceq x$.

Definition 4 *A (D, X)-ranking function $w : \overline{D^X} \to \mathbb{R}$ is* subset-monotone *if for every $Y \subseteq X$ and partial assignments $\tau_1, \tau_2 \in \overline{D^Y}$ such that $w(\tau_1) \leqslant w(\tau_2)$, for every partial assignment $\sigma \in \overline{D^{X \setminus Y}}$ (so disjoint with τ_1 and τ_2), we have $w(\sigma \times \tau_1) \leqslant w(\sigma \times \tau_2)$.*

We use in particular the following consequence of subset-monotonicity (, where we call $\tau \in \overline{D^X}$ *maximal* (or maximum) for $w : \overline{D^X} \to \mathbb{R}$ when for every $\tau' \in \overline{D^X}$ we have $w(\tau') \leqslant \tau(\tau)$:

Lemma 1 *Let $R \subseteq \overline{D^Y}$ and $S \subseteq \overline{D^Z}$ with $Y \cap Z = \varnothing$, and let $w : \overline{D^{Y \cup Z}} \to \mathbb{R}$ be subset-monotone. If τ is a maximal element of R and σ is a maximal element of S with respect to w, then $\tau \times \sigma$ is a maximal element of $R \wedge S$ with respect to w.*

We give a few examples of subset-monotone ranking functions. Let $W : X \times D \to \mathbb{R}$ be a function assigning scores to singleton assignments, and define the (D, X)-ranking function $\mathsf{sum}_W : \overline{D^X} \to \mathbb{R}$ by $\mathsf{sum}_W(\tau) = \sum_{x \in X, \tau(x) \neq \bot} W(x, \tau(x))$. Then sum_W is subset-monotone. Similarly define $\mathsf{max}_W : \overline{D^X} \to \mathbb{R}$ by $\mathsf{max}_W(\tau) = \max_{x \in X, \tau(x) \neq \bot} W(x, \tau(x))$, or prod_W in a similar manner (with non-negative scores for singletons); then these are again subset-monotone. In particular, we can use sum_W to encode lexicographic orderings on $\overline{D^X}$.

Enumeration and problem statement. Our goal in this article is to efficiently enumerate the satisfying assignments of circuits in nonincreasing order according to a ranking function. We will in particular apply this for the ranked enumeration of the answers to MSO queries on trees. We call this problem RankEnum. Formally, the input to RankEnum consists of a multivalued circuit C on domain D and variables X, and a (D, X)-ranking function w that is subset-monotone. The output to enumerate consists of all of $\mathrm{rel}(C)$, without duplicates, in nonincreasing order of scores (with ties broken arbitrarily).

Formally, we work in the RAM model on words of logarithmic size [AHU74], where memory cells can represent integers of value polynomial in the input length, and on which arithmetic operations take constant time. We will in particular allocate arrays of polynomial size in constant time, using lazy initialization [GJ22]. We measure the performance of our algorithms in the framework of *enumeration algorithms*, where we distinguish two phases. First, in the *preprocessing phase*, the algorithm reads the input and builds internal data structures. We measure the running time of this phase as a function of the input; in general, the best possible bound is *linear preprocessing*, e.g., preprocessing in $O(|C|)$. Second, in the *enumeration phase*, the algorithm produces the assignments, one after the other, without duplicates, and in nonincreasing order of scores; the order of assignments that are tied according to the ranking function is not specified. The *delay* is the maximal time that the enumeration phase can take to produce the next assignment, or to conclude that none are left. We measure the delay as a function of the input, as a function of the produced assignments (which each have size $|X|$), and also as a function of the number of results that have been produced so far.

3.2 Construction of the circuit for deterministic tree automata over trees

We present in this section how to represent efficiently the answers of evaluation of tree automata over trees via circuit.

Multivalued circuits. A *multivalued circuit C on domain D and variables X* is a DAG with labelled vertices which are called *gates*. The circuit also has a distinguished gate r called the *output gate of C*. Gates having no incoming edges are called *inputs* of C. Moreover, we have:

- Every input of D is labeled with a pair of the form $\langle x : d \rangle$ with $x \in X$ and $d \in D$;
- there exists two constant gates labeled by $\top$ or $\top$ gates.
- Every other gate of D is labeled with either $\vee$ (a *$\vee$-gate*) or $\wedge$ (a *$\wedge$-gate*).

We denote by $|C|$ the number of edges in C.

Given a gate v of C, the *inputs of v* are the gates w of C such that there is a directed edge from w to v. The *set of variables below v*, denoted by $\mathrm{var}\ 1$, is then the set of variables $x \in X$ such that there is an input w which is labeled by $\langle x : d \rangle$ for some $d \in D$ and which has a directed path to v. Equivalently, if v is an input labeled by $\langle x : d \rangle$ then $\mathrm{var}\ 1 := \{x\}$, otherwise $\mathrm{var}\ 1 := \bigcup_{i=1}^{k} \mathrm{var}\ 1$ where $v_1, \ldots, v_k$ are the inputs of v. We assume that the set X of variables of the circuit is equal to $\mathrm{var}\ 1$ for r the output gate of C: this can be enforced without loss of generality up to removing useless variables from X.

For each gate v of C, the *set of assignments* $\mathrm{rel}(v) \subseteq D^{\mathrm{var}\ 1}$ *of v* is defined inductively as follows. If v is an input labelled by $\langle x : d \rangle$, then $\mathrm{rel}(v)$ contains only the assignment $[x \mapsto d]$. Otherwise, if v is an internal gate with inputs $v_1, \ldots, v_k$ then $\mathrm{rel}(v) := \mathrm{rel}(v_1)\ \mathsf{op} \cdots \mathsf{op}\ \mathrm{rel}(v_k)$ where $\mathsf{op} \in \{\vee, \wedge\}$ is the label of v. The *set of assignments* $\mathrm{rel}(C)$ of C is that of its output gate. Note that, if $D = \{0, 1\}$, then the set of assignments of C precisely corresponds to its satisfying valuations when we see C as a Boolean circuit in the usual sense.

We say that a $\wedge$-gate v is *decomposable* if all its inputs are on disjoint sets of variables; formally, for every pair of inputs $v_1 \neq v_2$ of v, we have $\mathrm{var}\ 1 \cap \mathrm{var}\ 1 = \varnothing$. A $\vee$-gate v is *smooth* if all its inputs have the same set of variables (so that implicit completion does not occur); formally, for every pair of inputs v_1, v_2 of v, we have $\mathrm{var}\ 1 = \mathrm{var}\ 1$. A $\vee$-gate v is *deterministic* if every assignment of v is computed by only one of its inputs; formally, for every pair of inputs $v_1 \neq v_2$ of v, if $\tau \in \mathrm{rel}(v)$ then either $\tau|_{\mathrm{var}\ 1} \notin \mathrm{rel}(v_1)$ or $\tau|_{\mathrm{var}\ 1} \notin \mathrm{rel}(v_2)$.

Let v be an internal gate with inputs $v_1, \ldots, v_k$. Observe that if v is decomposable, then $\mathrm{rel}(v) = \bigtimes_{i=1}^{k} \mathrm{rel}(v_i)$. If v is smooth then $\mathrm{rel}(v) = \bigcup_{i=1}^{k} \mathrm{rel}(v_i)$. If moreover v is deterministic, then $\mathrm{rel}(v) = \biguplus_{i=1}^{k} \mathrm{rel}(v_i)$, where $\uplus$ denotes disjoint union. Accordingly, we denote decomposable $\wedge$-nodes as *$\times$-nodes*, denote smooth $\vee$-nodes as *$\cup$-nodes*, and denote smooth deterministic $\vee$-nodes as *$\uplus$-nodes*.

A multivalued circuit is *decomposable* (resp., *smooth*, *deterministic*) if every $\wedge$-gate is decomposable (resp., every $\vee$-gate is smooth, every $\vee$-gate is deterministic). A *multivalued DNNF on domain D and variables X* is then a decomposable multivalued circuit on D and X. A *multivalued d-DNNF on domain D and variables X* is a deterministic multivalued DNNF on D and X. In this chapter, we only work with circuits that are both decomposable and smooth, i.e., smooth multivalued DNNFs. Note that smoothness can be ensured on Boolean circuits in quadratic time [SdBBA19], and the same can be done on multivalued circuits.

Building multivalued Circuits We have defined the multivalued circuits that we want to compute, defined the notion of a DNNF and a width parameter for them. We can now state our main result for this section, namely, that we can efficiently construct multivalued circuits. Observe that, while the depth of the circuit depends on the input tree, the width only depends on $|Q|$, which will be crucial for our delay bounds.

Lemma 3 *Given any binary Λ-tree T and homogenized $\Lambda, \mathcal{X}$-TVA $A = (Q, \iota, \delta, F)$, we can construct in time $O(|T| \times |A|)$ a DNNF C which is a multivalued circuit of A and T, a v-tree $\mathcal{T}$, and a structuring function from C to $\mathcal{T}$, such that C has width $|Q|$ and depth $O(\mathrm{height}(T))$.*

Proof 1 *We construct $\mathcal{T}$ by taking T, removing all node labels, and labeling each leaf node n by the set of singletons $\langle \mathcal{X} : n \rangle$: note that $\mathcal{T}$ is indeed a ν-tree for the set of variables $C_{\mathrm{var}}\, 1\{\langle Z : n\rangle \mid Z \in \mathcal{X}, n \in T\}$ of C given by the definition of multivalued circuits.*

We now present the construction of C bottom up. We first describe the case of a leaf node n of T with label $l \in \Lambda$. In this case, we construct the box B_n for n as follows:

- *For every 0-state q of A, we set $\gamma(n, q)$ to be a $\top$-gate if $(l, \varnothing, q) \in \iota$, and a $\bot$-gate otherwise.*
- *For every 1-state q of A with no tuples of the form $(l, \mathcal{Y}, q)$ in ι, we set $\gamma(n, q)$ to be a $\bot$-gate.*
- *For every 1-state q of A with at least one tuple of the form $(l, \mathcal{Y}, q)$, we set $\gamma(n, q)$ to be a $\vee$-gate having as inputs one variable gate labelled by $\langle \mathcal{Y} : n\rangle$ for each $\mathcal{Y} \subseteq \mathcal{X}$ such that $(l, \mathcal{Y}, q) \in \iota$. Note that $\mathcal{Y}$ is then nonempty because q is a 1-state.*

It is clear that B_n has at most $|Q|$ $\vee$-gates and that all restrictions for DNNFs are met.

For an inner node n of T with label l and child nodes n_1 and n_2, we construct the box B_n as follows. First, for every 0-state of A, we set $\gamma(n, q)$ to be a $\top$-gate if and only if there are states q_1 and q_2 in A such that $(q_1, q_2, q) \in \delta_l$ and $\gamma(n_1, q_1)$ and $\gamma(n_2, q_2)$ are both $\top$-gates. Otherwise, we set $\gamma(n, q)$ to be a $\bot$-gate.

Second, for every 1-state q of A and every triple $(q_1, q_2, q) \in \delta_l$, let $g_1 := \gamma(n_1, q_1)$ and $g_2 := \gamma(n_2, q_2)$. We define a gate g^{q_1,q_2} such that we have the equality:

$$\mathrm{S}(g^{q_1,q_2}) = \mathrm{S}(g_1) \times \mathrm{S}(g_2) \tag{*}$$

but while respecting the rule that $\top$ and $\bot$-gates can never be used as input to another gate. Specifically:

- *If one of g_1, g_2 is a $\bot$-gate, we set g^{q_1,q_2} to be a $\bot$-gate, which clearly satisfies (*);*
- *If one of g_1, g_2 is a $\top$-gate, we set g^{q_1,q_2} to be the other gate; this also satisfies (*);*
- *Otherwise we set g^{q_1,q_2} to be a $\times$ gate with inputs g_1 and g_2.*

Having created the necessary gates g^{q_1,q_2} for the triples of δ_l, we now create $\gamma(n, q)$ for every 1-state q as a gate that satisfies:

$$\mathrm{S}(\gamma(n, q)) = \bigcup_{(q_1,q_2,q)\in\delta_l} \mathrm{S}(g^{q_1,q_2}) \tag{**}$$

Specifically:

- *If all g^{q_1,q_2} in the union are $\bot$-gates (in particular if the union is empty), we set $\gamma(n, q)$ to also be a $\bot$-gate, respecting (**);*
- *Otherwise we exclude all $\bot$-gates from the union and set $\gamma(n, q)$ to be a $\cup$-gate, which has all remaining gates g^{q_1,q_2} as input, satisfying (**).*

We can easily check that all rules of multivalued circuits are respected. In particular, all $\cup$-gates and $\times$-gates have the right fan-in. To check that we never use $\top$ and $\bot$ as input to another gate, the only subtlety is that, when defining the $\cup$-gate $\gamma(n, q)$ for a 1-state q, we must check that g^{q_1,q_2} can never be a $\top$-gate, but this is because one of q_1 and q_2 must be a 1-state, hence it cannot be a 0-state because A is homogenized; now it can be seen by induction that whenever $\gamma(n', q')$ is a $\top$-gate then q' is a 0-state. It is also clear that the definition of a DNNF is respected, in particular, the inputs to $\times$-gates are $\cup$-gates in the two child boxes.

In terms of accounting, it is clear that there are at most $|Q|$ $\cup$-gates in each B_n, that the depth of the circuit is as stated, and the construction of the whole circuit is in time $O(|A| \times |T|)$ as promised. Last, a

straightforward bottom-up induction on T shows that the gates $\gamma(n, q)$ capture the correct set for any n, i.e., that for any leaf node n and any $q \in Q$ we have:

$$\mathrm{S}(\gamma(n,q)) \quad = \quad \{\langle \mathcal{Y} : n \rangle \mid (\lambda(n), \mathcal{Y}, q) \in \iota\}$$

and for any internal node n with label l and children n_1 and n_2 and any $q \in Q$ we clearly have the following, by (*) and (**) (and their analogues in the case of 0-states):

$$\mathrm{S}(\gamma(n,q)) \quad = \bigcup_{(q_1,q_2,q)\in\delta_l} \mathrm{S}(\gamma(n_1,q_1)) \times \mathrm{S}(\gamma(n_2,q_2))$$

Hence, the construction is correct, which concludes the proof.

In our original paper [ABJM17], we studied the enumeration of the MSO formula for a representation of the answer using Boolean circuits and in particular d-DNNF circuits. This representation brings several difficulties. First, the representation of an assignment of the variables is defined through a set of variables of the form $< x : a >$ where x is a first-order variable and a is a value or of the form $< X : a >$ where X is a second order variable. For the assignments of first-order variables, we impose that our circuits accept only valuations for which for each free variable x, there exists only one variable of the form $< x : a >$ assigned to the value $\top$. The classical results for enumerating valuations for d-DNNF give a linear delay in the size of the valuations. Unfortunately, this result is not sufficient as the number of variables in the translation in the representation of the set of answers implies a valuation of size linear in the size of the tree. Therefore, with the classical approaches, we could not get the known results: linear time preprocessing and constant delay. For solving, this problem we focus on enumerating assignments i.e. only the variables assigned to $\top$. For representing these assignments, we used different equivalent representations: zero suppressed semantics [ABJM17], set circuits [ABMN19b] and finally multivalue circuits [ABCM24].

In this section, we restate our result for multi-value circuits and we represent our algorithm through a more flexible pattern.

Theorem 6 *For any constant $n \in \mathbb{N}$, we can solve the enumeration problem on an input smooth multivalued d-DNNF circuit C with n variables , with preprocessing $O(|C|)$ and delay $O(1)$*

The main idea of the enumeration is not to enumerate exactly an assignment but to enumerate the $\wedge$-gates and the variables gates of the circuits defining an assignment. We can notice that because our multivalue circuits have a constant number of variables, the subtree composed of only $\wedge$-gates defining the assignment has a constant size.

The main difficulty when enumerating such subtree is to go through the $\vee$ gates to find the next possible $\wedge$-gate. Due to the fact, that the number of $\vee$ gates to go through to go from one $\wedge$ gate to the another can be linear in the size of the tree.

To that, we want to create for each $\vee$ gate which stores the descendant $\wedge$ gates reachable using $\vee$ gates.

We can notice that we can build such sets through a bottom-up evaluation, however, we need some particular properties to get the best-known result:

- First, the structure should be functional, i.e. update operations do not change the initial structure. Because the circuit is a †the sets can be shared by different gates.
- Secondly, this structure should have the following operations :

- *Initialize*, in time $O(1)$, which produces an empty queue;
- *Push*, in time $O(1)$, which adds an element to the set
- *Get*, in time $O(1)$, which either indicates that Q is empty or otherwise returns an element of the set which is considered the first element of the queue
- *Pop*, in time $O(1)$ which returns the storing all the pairs of Q except p;
- *Union*, in time $O(1)$, which takes as input a second queue Q' and returns a queue over the elements of Q and Q'.

In [ABJM19], we present a structure to get the desired bounds. We can notice that once we get the structure, we can enumerate the subtrees using these sets in the straightforward manner. For enumerating, the assignemnents at a $\wedge$ with two subcircuits C_1 and C_2. It is sufficient to compose each assignment of C_1 each assignement of C_2 by enumerating the assignement of C_1 and C_2 in two nested loops.

3.2.1 Updates

As the enumeration of answers of MSO over trees was solved, there was a lot of attention to how to update the compact representation of the answers when the tree is updated. [LM14] is the first paper presenting results showing that enumeration can done with a linear preprocessing, a logarithmic delay and the structure can be updated in logarithmic square. One of the complexity comes from the shape of a tree. Indeed a tree can have very diverse shapes for the same size. This paper uses a previous result showing that any tree can be balanced in linear time such that its depth is logarithmic square. Their enumeration result was with a linear preprocessing and a logarithm delay.

Some other papers are focused on enumeration for words [NS18] obtaining the desired complexity which is linear time preprocessing, a constant delay and a logarithmic time for updating the structure.

In [ABMN19b], we present a generic approach for dealing with updates for any techniques using a bottom-up evaluation over d-DNNF. We restate this approach with the following lemma.

Lemma 4 *Let ϕ be a MSO formula with first-order free variables. Let T be a tree. Let $C_{\phi,T}$ be the multivalue circuit representing the solution of $\phi(T)$. Let U be a sequence of updates over the tree. Let n be the number of nodes and its ancestors are impacted by U. There exists a transformation $C_{\phi,T}$ in $C_{U(\phi T)}$ of complexity linear in n and the set of impacted gates closed under ancestor is linear in $|U|$.*

Thus, we can notice that the complexity of maintaining the circuit depends principally of the depth of trees. Fortunately, there exist two main results showing that a tree can be balanced to another tree such that any MSO formula over the first tree can be translated into another MSO formula on the second tree. [BPV04] presents an approach to balance a tree with a depth of $O(\log^2(|T|)$ such that for each update on T implies that $\log^2(|T|)$ nodes (closed under ancestors) on T' are affected. [KMN22] presents an approach to balance a tree with a depth of $O(\log(|T|)$ such that for each update on T implies that $\log(|T|)$ nodes (closed under ancestors) on T' are affected. For information [KMN22] is a revision of [Nie18].

In the context of enumeration, we can notice that the initial structures needed for the enumeration are built in a bottom-up manner and therefore maintaining the number of initial structures is linear in the number of gates of the circuits impacted by the update. By combining these remarks, we obtain the following result proved in [ABMN19b].

Corrollary 1 *Let ϕ be a MSO formula with first-order free variables. Let T be a tree. The enumeration of $\phi(T)$ can be performed with a linear time preprocessing a constant delay and maintenance of updates in $\log(|T|)$.*

3.3 Enumerations for multi-valued d-DNNF circuits

We present the main part of an algorithm for smooth multivalued DNNF circuits that are further assumed to be *deterministic*, but which achieves linear-time preprocessing and delay $O(\log(K+1))$, where K denotes the number of satisfying assignments produced so far. This proves the following result.

Theorem 7 *For any constant $n \in \mathbb{N}$, we can solve the* RankEnum *problem on an input smooth multivalued d-DNNF circuit C with n variables and a subset-monotone ranking function, with preprocessing $O(|C|)$ and delay $O(\log(K+1))$, where K is the number of assignments produced so far.*

The idea is similar as the enumeration, we try to enumerate the $\wedge$-gates and variable gates defining the assignements. However, we need to keep an order over the partial assignements which can be done using queues. In order to quickly "jump" over $\uplus$-gates, we can use a particular queue, Brodal queue, presented below. Secondly, we describe briefely the algorithm to handle $\times$ gates during the enumeration. Details can be find in [ABCM24].

Brodal queues. Similar to [BGJR21], our algorithms will use priority queues, in a specific implementation called a *(functional) Brodal queue* [BO96]. Intuitively, Brodal queues are priority queues which support union operations in $O(1)$, and which are *purely functional* in the sense that operations return a queue without destroying the input queue(s). More precisely, a *Brodal queue* is a data structure which stores a set of priority-data pairs of the form $(\mathbf{p} : \mathrm{foo}, \mathbf{d} : \mathrm{bar})$ where foo is a real number and bar an arbitrary piece of data, supporting operations defined below. Brodal queues are *purely functional and persistent*, i.e., for any operation applied to some input Brodal queues, we obtain as output a new Brodal queue Q', such that the input queues can still be used. Note that the structures of Q' and the input Brodal queues may be sharing locations in memory; this is in fact necessary, e.g., to guarantee constant-time bounds. However, this is done transparently, and both Q' and the input Brodal queues can be used afterwards[3]. Brodal queues support the following:

- *Initialize*, in time $O(1)$, which produces an empty queue;
- *Push*, in time $O(1)$, which adds to Q a priority-data pair;
- *Find-Max*, in time $O(1)$, which either indicates that Q is empty or otherwise returns some pair $(\mathbf{p} : \mathrm{foo}, \mathbf{d} : \mathrm{bar})$ with foo being maximal among the priority-data pairs stored in Q (ties are broken arbitrarily);
- *Pop-Max*, in time $O(\log(|Q|))$, which either indicates that Q is empty or returns two values: first, the pair p returned by Find-Max, second a queue storing all the pairs of Q except p;
- *Union*, in time $O(1)$, which takes as input a second Brodal queue Q' and returns a queue over the elements of Q and Q'.

The preprocessing initialize the structures in order to ranked enumerate the partial assignements. In particular, the $\times$-gate reachables from a path of $\vee$-gates are collected and ranked in linear time thanks to Brodal queues.

The main complexity comes from the enumeration of the $\times$ gate. Intuivily, a $\times$ gate does need to ranked enumerate a product of two sets.

[3]This is similar to how persistent linked lists can be modified by removing the head element or concatenating with a new head element. Such operations can run in constant time and return the modified version of the list without invalidating the original list; with both lists transparently sharing some memory locations.

We explainn how given as input two tables (indexed starting from 1) A, B of reals of size n_1, n_2 sorted in nonincreasing order, we can enumerate the set of integer pairs $\{(i,j) \mid (i,j) \in \{1,\dots,n_1\} \times \{1,\dots,n_2\}\}$, in nonincreasing order of the score $A[i] \odot B[j]$, with $O(1)$ preprocessing and a delay $O(\log K)$ where K is the number of pairs outputted so far.

Intuitively, this will be applied at every $\times$-gate g, with $[n_1]$ (resp., $[n_2]$) representing the satisfying valuations of the first (resp., second) input of g sorted in nonincreasing order, as in the table A (resp., B). . We initialize a two-dimensional bit table R of size $n_1 \times n_2$ to contain only zeroes (again using lazy initialization [GJ22, Section 2.5]), whose role will be to remember which pairs have been seen so far, and a priority queue Q containing only the pair $(\mathbf{p} : A[1] \odot B[1],\ \mathbf{d} : (1,1))$; we set $R[1,1]$ to true because the pair $(1,1)$ has been seen. Then, while the queue is not empty, we do the following. We pop (call *Pop-Max*) from Q, obtaining a priority-data pair of the form $(\mathbf{p} : A[i] \odot B[j],\ \mathbf{d} : (i,j))$. We output the pair (i,j). Then, for each $(p,q) \in \{(i+1,j),(i,j+1)\}$ that is in the $[n_1] \times [n_2]$ grid, if the pair (p,q) has not been seen before, then we push into Q the pair $(\mathbf{p} : A[p] \odot B[q],\ \mathbf{d} : (p,q))$ and mark (p,q) as seen in R.

We last move on to the enumeration phase. We first give a high-level description of how the enumeration phase works, before presenting the details.

The operation $\mathsf{Get}(g,j)$**.** We will define a recursive operation Get, running in complexity $O(\log(K+1))$, that applies to a gate g and integer $1 \leqslant j \leqslant i_g + 1$ and does the following. If $j \leqslant i_g$ then $\mathsf{Get}(g,j)$ simply returns the satisfying assignment of g that is stored in $T_g[j]$ (i.e., this assignment has already been computed). Otherwise, if $j = i_g + 1$, then $\mathsf{Get}(g,j)$ finds the next assignment to be enumerated, inserts it into T_g, and returns that assignment. Note that, in this case, calling $\mathsf{Get}(g,j)$ modifies the memory for g and some other gates g'. Specifically, it modifies the tables $T_{g'}$ and $R_{g'}$, the queues $Q_{g'}$, and the integers $i_{g'}$ for various gates g' having a directed path to g (including $g' = g$).

When we are not executing an operation Get, the memory will satisfy the following invariants, for every g of C:

- The table T_g contains assignments $\tau \in \mathrm{rel}(g)$, ordered by nonincreasing score and with no duplicates; and i_g is the current size of T_g;
- For any assignment $\tau \in \mathrm{rel}(g)$ that does not occur in T_g, it is no larger than the last assignment in T_g, i.e., we have $w(\tau) \leqslant w(T_g[i_g])$.
- The queues Q_g will also satisfy some invariants, which will be presented later.
- The tables R_g for the $\times$-gates record whether we have already seen pairs of satisfying assignments of the two children, similarly to how this is done in the $A \odot B$ algorithm.

The tables T_g store the assignments in the order in which we find them, which is compatible with the ranking function. This allows us, in particular, to obtain in constant time the j-th satisfying assignment of $\mathrm{rel}(g)$ if it has already been computed, i.e., if $j \leqslant i_g$. The reason why we keep the assignments in the tables T_g is because we may reach the gate g via many different paths throughout the enumeration, and these paths may be at many different stages of the enumeration on g.

At the top level, if we can implement Get while satisfying the invariants above, then the enumeration phase of the algorithm is simple to describe: for j ranging from 1 to $\#r$, we output $\mathsf{Get}(r,j)$, where r is the output gate of C.

Implementing Get**.** We first explain the intended semantics of data values in the queues Q_b:

- If g is a $\uplus$-gate then Q_b will always contain pairs of the form $(\mathbf{p} : w(\tau),\ \mathbf{d} : (g', j, \tau))$ where $g' \in \mathsf{exit}(g)$ and $j \in \{1, \ldots, i_{g'} + 1\}$ and $\tau \in \mathrm{rel}(g')$, and the idea is that at the end of the enumeration τ will be stored at position j in $T_{g'}$.
- If g is a $\times$-gate, letting g'_1 and g'_2 be the input gates, then Q_b will always contain pairs of the form $(\mathbf{p} : w(\tau_1 \times \tau_2),\ \mathbf{d} : (j_1, j_2, \tau_1, \tau_2))$ with $\tau_i \in \mathrm{rel}(g_i)$ and at the end of the enumeration τ_i will be at position j_i in T_{g_i} with $j_i \in \{1, \ldots, i_{g'_i} + 1\}$ for all $i \in \{1, 2\}$.
- If g is an input gate, then Q_b initially contains the only assignment captured by g, becomes empty the first time we call $\mathsf{Get}(g, 1)$, and remains empty thereafter.

Chapter 4

Dialog between Logic and Circuit

The notion of provenance is a key notion to describe the contribution of the tuples to satisfying a query. The classical notion is called Boolean Provenance. Its definition states that it represents all the subsets of the database satisfying the query. Different extensions have been proposed and most of these notions have been unified through a common framework based on semi-ring in [GKT07]. Different provenance definitions are equivalent to the computation of the value based on a particular semi-ring. Each fact in the database is assigned to a value of the semi-ring. [GKT07] explains how to compute for each positive operator of relational algebra $(\sigma, \times, \bowtie, \pi)$ and each tuple resulting from applying this operator. It generalized for Datalog programs through their notions of proof trees by summing the provenance of each proof tree which is equal to the product of the value of the extensional facts in it. This notion has been adopted in the database community; however it has some issues. First of all, the number of proof trees of the same answer can be infinite. Following the semi-ring, it is possible that the sum of infinite values is not defined. Secondly, the size of the provenance can be enormous and despite that the complexity is polynomial in the time of the database, in practice it is not reasonable to compute the full provenance by using classical formula values.

The problem of the infinite number of proof trees has been studied differently, the work [KNP$^+$24] studied semi-ring where it is possible to obtain a finite representation of the provenance even when there is an infinite number of proof trees. The KR community also started to study a new definition of provenance which is not based on proof trees. For the size of provenance, [DMRT14] proposed a notion of compact representation of formula based on circuits. [DGM18] proposes a notion of sampling of provenance based on patterns in order to avoid computing the full provenance.

In this chapter, we present the main results of [ABMS19] and [BBPT22a]. [ABMS19] proposes a new notion of circuits called cycluits that allow us to get a parameterized complexity over the size of the provenance. [BBPT22a] studies different alternative definitions of provenance to try to avoid the problem of infinite proof trees. We analyze the advantages and disadvantages of these different definitions.

4.1 Preliminaries

The notion of Datalog is defined in Subsection 2.2.2. We present other semantics of Datalog that will be used in this chapter.

Semantics

The semantics of Datalog can classically be defined in three ways: through models, fixpoints or derivation trees. All three definitions rely on the notion of homomorphism: a *homomorphism* from a set $\mathcal{A}$ of atoms to a set $\mathcal{B}$ of atoms is a function $h : \mathcal{D}(\mathcal{A}) \to \mathcal{D}(\mathcal{B})$ such that $h(t) = t$ for all $t \in \mathbf{C}$, and $p(t_1, \ldots, t_n) \in \mathcal{A}$ implies $h(p(t_1, \cdots, t_n)) := p(h(t_1), \ldots, h(t_n)) \in \mathcal{B}$. We denote by $h(\mathcal{A})$ the set $\{h(p(t_1, \ldots, t_n)) \mid p(t_1, \ldots, t_n) \in \mathcal{A}\}$. The homomorphism definition is extended to conjunctions of atoms by viewing them as the sets of atoms they contain.

A set I of facts is a *model* of a rule $r := \phi(\boldsymbol{x}, \boldsymbol{y}) \to \psi(\boldsymbol{x})$, denoted by $I \models r$, if every homomorphism h from $\phi(\boldsymbol{x}, \boldsymbol{y})$ to I is also a homomorphism from $\psi(\boldsymbol{x})$ to I; it is a *model* of a Datalog program Σ if $I \models r$ for every $r \in \Sigma$; it is a *model* of a database D if $D \subseteq I$. A fact α is *entailed* by D and Σ, denoted $\Sigma, D \models \alpha$, if $\alpha \in I$ for every model I of Σ and D.

Example 6 *Let Σ contain the rules $B(x) \to A(x)$, $R(x, y) \wedge A(y) \to B(x)$, and $R(x, y) \to R(y, x)$, and $D := \{B(a), B(b), R(a, b), R(b, a)\}$. Each model of D and Σ contains all facts in D as well as $A(a)$ and $A(b)$, which are thus entailed by Σ, D.*

An equivalent way to define the entailment of a fact α by D and Σ is to check if there is a homomorphism from α to a specific model, defined as the *least fixpoint* containing D of the immediate consequence operator: An immediate consequence for D and Σ is either $\alpha \in D$, or α such that there exists a rule $r := \phi(\boldsymbol{x}, \boldsymbol{y}) \to \psi(\boldsymbol{x})$ and a homomorphism h from $\phi(\boldsymbol{x}, \boldsymbol{y})$ to D such that $h(\psi(\boldsymbol{x})) = \alpha$.

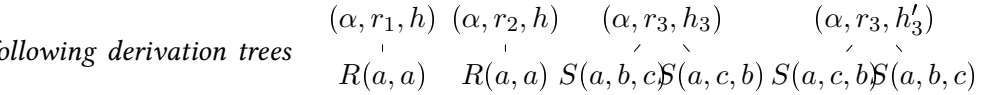
Finally, a third definition relies on derivation trees.

Definition 5 (Derivation Tree) *A* derivation tree t *of a fact α w.r.t. a database D and a program Σ is a finite tree whose leaves are labelled by facts from D and non-leaf nodes are labeled by triples $(p(t_1, \ldots, t_m), r, h)$ where*

- *$p(t_1, \ldots, t_m)$ is a fact over the schema $\mathcal{S}(\Sigma)$;*
- *r is a rule from Σ of the form $\phi(\boldsymbol{x}, \boldsymbol{y}) \to p(\boldsymbol{x})$;*
- *h is a homomorphism from $\phi(\boldsymbol{x}, \boldsymbol{y})$ to the facts of the labels of the node children, such that $h(p(\boldsymbol{x})) = p(t_1, \ldots, t_m)$;*
- *there is a bijection f between the node children and the atoms of $\phi(\boldsymbol{x}, \boldsymbol{y})$, such that for every $q(\boldsymbol{z}) \in \phi(\boldsymbol{x}, \boldsymbol{y})$, $f(q(\boldsymbol{z}))$ is of the form $(h(q(\boldsymbol{z})), r', h')$ or is a leaf labeled by $h(q(\boldsymbol{z}))$.*

Moreover, if $(p(t_1, \cdots, t_m), r, h)$ or $p(t_1, \cdots, t_m)$ is the root of t, then $p(t_1, \cdots, t_m) = \alpha$.

Example 7 *Let Σ contain $r_1 := R(x, y) \to H(x, x)$, $r_2 := R(x, y) \to H(x, y)$ and $r_3 := S(x, y, z) \wedge S(x, z, y) \to H(x, x)$. If $D = \{R(a, a), S(a, b, c), S(a, c, b)\}$, then the fact $\alpha := H(a, a)$ has the following derivation trees*



where $h(x) = h(y) = a$, $h_3(x) = a$, $h_3(y) = b$, $h_3(z) = c$ and $h_3'(x) = a$, $h_3'(y) = c$, $h_3'(z) = b$.

Note that when the program at hand is recursive (i.e., the dependency graph of its predicates contains cycles) a fact may have infinitely many derivation trees.

4.1.1 Annotated Databases

To equip databases with extra information, their facts might be annotated with, e.g., trust levels, clearance degree required to access them, or identifiers to track how they are used.

In the framework of semiring provenance, annotations are elements of algebraic structures known as commutative semirings. A *semiring* $\mathbb{K} = (K, +_\mathbb{K}, \times_\mathbb{K}, 0_\mathbb{K}, 1_\mathbb{K})$ is a set K with distinguished elements $0_\mathbb{K}$ and $1_\mathbb{K}$, equipped with two binary operators: $+_\mathbb{K}$, called the *addition*, which is an associative and commutative operator with identity $0_\mathbb{K}$, and $\times_\mathbb{K}$, called the *multiplication*, which is an associative operator with identity $1_\mathbb{K}$. It also holds that $\times_\mathbb{K}$ distributes over $+_\mathbb{K}$, and $0_\mathbb{K}$ is annihilating for $\times_\mathbb{K}$. When multiplication is commutative, the semiring is said to be *commutative*. We use the convention according to which multiplication is applied before addition to omit parentheses. We omit the subscript of operators and distinguished elements when there is no ambiguity.

Definition 6 *An* annotated database *is a triple* $(D, \mathbb{K}, \lambda)$ *where* D *is a database,* $\mathbb{K} = (K, +_\mathbb{K}, \times_\mathbb{K}, 0_\mathbb{K}, 1_\mathbb{K})$ *is a semiring, and* $\lambda : D \mapsto K \backslash \{0_\mathbb{K}\}$ *maps facts into semiring elements different from* $0_\mathbb{K}$.

Example 8 (Ex. 6 cont'd) *The semiring* $\mathbb{N} = (\mathbb{N}, +, \times, 0, 1)$ *of the natural numbers equipped with the usual operations is used for bag semantics. The tropical semiring* $\mathbb{T} = (\mathbb{R}_+^\infty, \mathsf{min}, +, \infty, 0)$ *is used to compute minimal-cost paths. We define* $\lambda_\mathbb{N} : D \mapsto \mathbb{N}\backslash\{0\}$ *by* $\lambda_\mathbb{N}(B(a)) = 3$, $\lambda_\mathbb{N}(B(b)) = 1$, $\lambda_\mathbb{N}(R(a,b)) = 2$, $\lambda_\mathbb{N}(R(b,a)) = 1$*; And* $\lambda_\mathbb{T} : D \mapsto \mathbb{R}_+$ *by* $\lambda_\mathbb{T}(B(a)) = 10$, $\lambda_\mathbb{T}(B(b)) = 1$, $\lambda_\mathbb{T}(R(a,b)) = 5$, $\lambda_\mathbb{T}(R(b,a)) = 2$.

We next list some possible properties of semirings. A semiring is $+$ *-idempotent* (resp. $\times$ *-idempotent*) if for every $a \in K$, $a + a = a$ (resp. $a \times a = a$). It is *absorptive* if for every $a, b \in K$, $a \times b + a = a$. It is *positive* if for every $a, b \in K$, $a \times b = 0$ if and only if ($a = 0$ or $b = 0$), and $a + b = 0$ if and only if $a = b = 0$. Finally, an important class is that of ω-continuous commutative semirings in which infinite sums are well-defined. Given a semiring, we define the binary relation $\sqsubseteq$ such that $a \sqsubseteq b$ if and only if there exists $c \in K$ such that $a + c = b$. A commutative semiring is ω*-continuous* if $\sqsubseteq$ is a partial order, every (infinite) ω-chain $a_0 \sqsubseteq a_1 \sqsubseteq a_2 \ldots$ has a least upper bound $\sup((a_i)_{i\in\mathbb{N}})$, and for every a, $a + \sup((a_i)_{i\in\mathbb{N}}) = \sup((a + a_i)_{i\in\mathbb{N}})$ and $a \times \sup((a_i)_{i\in\mathbb{N}}) = \sup((a \times a_i)_{i\in\mathbb{N}})$.

The semantics of queries from the positive relational algebra, and in particular of UCQs, over annotated databases is defined inductively on the structure of the query [GKT07]. Intuitively, joint use of data (conjunction) corresponds to multiplication, and alternative use of data (union or projection) corresponds to addition.

Example 9 (Ex. 8 cont'd) *The BCQ* $\exists xy\,(R(x,y) \wedge B(y))$ *is entailed from* $(D, \mathbb{N}, \lambda_\mathbb{N})$ *with multiplicity* $\lambda_\mathbb{N}(R(a,b)) \times \lambda_\mathbb{N}(B(b)) + \lambda_\mathbb{N}(R(b,a)) \times \lambda_\mathbb{N}(B(a)) = 5$*, and from* $(D, \mathbb{T}, \lambda_\mathbb{T})$ *with minimal cost* $\mathsf{min}(\lambda_\mathbb{T}(R(a,b)) + \lambda_\mathbb{T}(B(b)), \lambda_\mathbb{T}(R(b,a)) + \lambda_\mathbb{T}(B(a))) = 6$.

A semantics of Datalog over annotated databases has been defined by [GKT07] using derivation trees, that we shall name the *all-tree semantics*. It associates to each fact α entailed by Σ and D the following sum, where $T_D^\Sigma(\alpha)$ is the set of all derivation trees for α w.r.t. Σ and D and $\Lambda(t) := \prod_{v \text{ is a leaf of } t} \lambda(v)$ is the $\mathbb{K}$-annotation of the derivation tree t (since $\mathbb{K}$ is commutative, the result of the product is well-defined).

$$\mathcal{P}^{\mathsf{AT}}(\Sigma, D, \mathbb{K}, \lambda, \alpha) := \sum_{t \in T_D^\Sigma(\alpha)} \Lambda(t).$$

Since $T_D^\Sigma(\alpha)$ may be infinite, $\mathcal{P}^{\mathsf{AT}}$ is well-defined for all Σ, $(D, \mathbb{K}, \lambda)$ and α only in the case where $\mathbb{K}$ is ω-continuous.

Example 10 (Ex. 8 cont'd) *The fact $\alpha := A(a)$ is entailed with minimal cost: $\mathcal{P}^{AT}(\Sigma, D, \mathbb{T}, \lambda_\mathbb{T}, \alpha) = \mathsf{min}_{t \in T_D^\Sigma(\alpha)} \Sigma_{v \text{ is a leaf of } t} \lambda_\mathbb{T}(v) = 3$. Since $\mathbb{N}$ is not ω-continuous, $\mathcal{P}^{AT}(\Sigma, D, \mathbb{N}, \lambda_\mathbb{N}, \alpha)$ is not defined.*

4.1.2 Provenance Semirings

Provenance semirings have been introduced to abstract from a particular semiring by associating a unique provenance token to each fact of the database, and building expressions that trace their use. Given a set X of *variables* that annotate the database, a *provenance semiring* $Prov(X)$ is a semiring over a space of provenance expressions with variables from X.

Various such semirings were introduced in the context of relational databases [Gre09]: The most expressive annotations are provided by the *provenance polynomials* semiring $\mathbb{N}[X]$ of polynomials with coefficients from $\mathbb{N}$ and variables from X, and the usual operations. Less general provenance semirings include, for example, the semiring $\mathbb{B}[X]$ of polynomials with Boolean coefficients, and the semiring $PosBool(X)$of positive Boolean expressions.

In the Datalog context, it is important to allow for infinite provenance expressions, as there can be infinitely many derivation trees. A *formal power series* with variables from X and coefficients from K is a mapping that associates to each monomial over X a coefficient in K. A formal power series S can be written as a possibly infinite sum $S = \Sigma_{m \in \mathsf{mon}(X)} S(m)m$ where $\mathsf{mon}(X)$ is the set of monomials over X and $S(m)$ is the coefficient of the monomial m. The set of formal power series with variables from X and coefficients from K is denoted $K[\![X]\!]$. [GKT07] define the *Datalog provenance semiring* as the semiring $\mathbb{N}^\infty[\![X]\!]$ of formal power series with coefficients from $\mathbb{N}^\infty = \mathbb{N} \cup \{\infty\}$.

A *semiring homomorphism* from $\mathbb{K} = (K, +_\mathbb{K}, \times_\mathbb{K}, 0_\mathbb{K}, 1_\mathbb{K})$ to $\mathbb{K}' = (K', +_{\mathbb{K}'}, \times_{\mathbb{K}'}, 0_{\mathbb{K}'}, 1_{\mathbb{K}'})$ is a mapping $h : K \to K'$ such that $h(0_\mathbb{K}) = 0_{\mathbb{K}'}$, $h(1_\mathbb{K}) = 1_{\mathbb{K}'}$, and for all $a, b \in K$, $h(a +_\mathbb{K} b) = h(a) +_{\mathbb{K}'} h(b)$ and $h(a \times_\mathbb{K} b) = h(a) \times_{\mathbb{K}'} h(b)$. A semiring homomorphism between ω-continuous semirings is *ω-continuous* if it preserves least upper bounds: $h(\sup((a_i)_{i \in \mathbb{N}})) = \sup((h(a_i))_{i \in \mathbb{N}})$.

Following [DMRT14] , we say that a provenance semiring $Prov(X)$ *specializes* correctly to a semiring $\mathbb{K}$, if any valuation $\nu : X \to K$ extends uniquely to a (ω-continuous if $Prov(X)$ and $\mathbb{K}$ are ω-continuous) semiring homomorphism $h : Prov(X) \to K$, allowing the computations for $\mathbb{K}$ to factor through the computations for $Prov(X)$. A provenance semiring $Prov(X)$ is *universal for a set of semirings* if it specializes correctly to each semiring of this set. [GKT07] showed that $\mathbb{N}[X]$ is universal for commutative semirings, and $\mathbb{N}^\infty[\![X]\!]$ is universal for commutative ω-continuous semirings.

4.2 Cycluit

In this section at computing efficiently, the notion of Boolean provenance for a particular kind of datalog program called ICG-Datalog program.

Datalog with stratified negation [AHV95] 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:

1. 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 $P_{\zeta(R)}$;
2. 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 $\zeta(R) \leqslant i$;
3. 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. Hence, there is a unique least fixpoint and $I \models P$ is well-defined.

Example 11 *The following stratified Datalog program, with $\sigma = \{R\}$ and $\sigma_{\text{int}} = \{T, Goal\}$, and strata P_1, P_2, tests if there are two elements that are not connected by a directed R-path:*

$$P_1 : T(x,y) \leftarrow R(x,y), \quad T(x,y) \leftarrow R(x,z) \wedge T(z,y) \qquad P_2 : Goal() \leftarrow \neg T(x,y)$$

Definition 7 *Let P be a stratified Datalog program. An intensional literal $A(\mathbf{x})$ or $\neg A(\mathbf{x})$ in a rule body ψ of P is* clique-guarded *if, for any two variables $x_i \neq x_j$ of $\mathbf{x}$, x_i and x_j co-occur in some extensional atom of ψ. P is* intensional-clique-guarded *(ICG) if, for any rule $R(\mathbf{x}) \leftarrow \psi(\mathbf{x}, \mathbf{y})$, every* intensional *literal in ψ is clique-guarded in ψ. The* body size *of P is the maximal number of atoms in the body of its rules, multiplied by its arity.*

We will also use these cycluits as a new powerful tool to compute (Boolean) *provenance information*, i.e., a representation of how the query result depends on the input data:

Definition 8 *A (Boolean)* valuation *of a set S is a function $\nu : S \to \{0,1\}$. A* Boolean function *ϕ on variables S is a mapping that associates to each valuation ν of S a Boolean value in $\{0,1\}$ called the* evaluation *of ϕ according to ν; for consistency with further notation, we write it $\nu(\phi)$. The* 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 $\nu(\phi) = 1$ iff the subinstance $\{F \in I \mid \nu(F) = 1\}$ satisfies Q.*

We can represent Boolean provenance as Boolean formulae [IL84, GKT07], or (more recently) as Boolean circuits [DMRT14, ABS15a]. 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.

We now define the semantics of monotone cycluits. A (Boolean) *valuation* of C is a function $\nu : C_{\text{inp}} \to \{0,1\}$ 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 $\nu(\phi) := \nu(C)$ 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.

The main result is that the computation of provenance of ICG-Datalog is *FPT-linear* in *combined complexity*, when parameterized by the body size of the program and the instance treewidth.

Theorem 8 *Given an ICG-Datalog program P of body size k_{P} and a relational instance I of treewidth k_{I}, we can construct in FPT-linear 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. Further, for fixed k_{I}, this cycluit has treewidth $O(|P|)$.*

The idea of the proof of this theorem relies on the constructions proposed in Subsection 2.1.8 and Section 3.2. By using the tree width of the instance, it is possible to encode of the instance into a tree

encoding. By using this encoding, the ICG-Datalog program can be translated into a particular two way alternating automata called SATWA (details in [ABMS19]) by using a similar idea developped in Section 2.1.8. Finally, we can obtain a circuit to compute the provenance and not the set of answers with a similar approach presented in Section 3.2. However, the differences come by computing the provenance directly by using a two-way alternating tree automata and not a deterministic tree automata.

Theorem 9 *Given an ICG-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 for treewidth k_{I}.*

Theorem 10 *For any fixed alphabet Γ, given a $\overline{\Gamma}$-SATWA A and a Γ-tree $\mathcal{T}$, we can build a stratified cycluit capturing the provenance of A on $\mathcal{T}$ in time $O(|A| \cdot |\mathcal{T}|)$. Moreover, this stratified cycluit has treewidth $O(|A|)$.*

Finally, we look at the expressivity of ICG Datalog program

Guarded negation fragments. ICG-Datalog can express different guarded logics specifically GNF. Indeed, when putting GNF formulae in *GN-normal form* [BtCS15] or even *weak GN-normal form* [BtCV14], we can translate them to ICG-Datalog, and we can use the *CQ-rank* parameter [BtCV14] (that measures the maximal number of atoms in conjunctions) to control the body size parameter.

Recursive languages. The use of fixpoints in ICG-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 [GPW10]:

Second, ICG-Datalog can capture *two-way regular path queries* (2RPQs) [CDLV00, Bar13], a well-known query language in the context of graph databases and knowledge bases:

ICG-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 compilability to automata (and efficient compilation of provenance representations). We do not know whether this extends to the more general classes studied in [BRV14].

However ICG-Datalog cannot express GNFP-UP formulas as ICG-Datalog cannot express general 2CRPQs.

4.3 Alternative Semantics

In this section we propose several natural ways of defining the semantics of Datalog over annotated databases, and investigate their connections. We have seen that the semantics of Datalog can equivalently be defined through models, fixpoints or derivation trees. The semantics we propose also fall into these three approaches. For presentation purposes, we see each semantics as a partial function $\mathcal{P}$ that associates to a Datalog program, annotated database $(D, \mathbb{K}, \lambda)$, and fact α, a semiring element $\mathcal{P}(\Sigma, D, \mathbb{K}, \lambda, \alpha)$.

4.3.1 Model-Based Semantics

We first investigate two provenance semantics based on Datalog's model-theoretic semantics. In both cases, we will define interpretations (I, μ^I) where I is a set of facts and μ^I is a function that annotates facts of I, and formulate requirements for them to be models of Σ and $(D, \mathbb{K}, \lambda)$, extending standard models of Σ and D with fact annotations.

Annotated Model-based

[HK17] define two bag semantics in the context of data exchange: the *incognizant* and *cognizant* semantics. The difference between them arise from the two different semantics of bag union: the incognizant semantics uses the maximum-based union, while the cognizant semantics uses the sum-based union.

In more details, both semantics are based on the following semantics for source-to-target tuple generating dependencies (s-t tgds): a pair (I, J) of source and target instances satisfies an s-t tgd $q_1(\boldsymbol{x}) \rightarrow q_2(\boldsymbol{x})$ if for every answer $\boldsymbol{a}$ to q_1 over I, $\boldsymbol{a}$ is an answer to q_2 over J with at least the same multiplicity. Given a set of s-t tgds Σ and a source I, a target J is an *incognizant solution* for I w.r.t. Σ if (I, J) satisfies every s-t tgd in Σ. It is a *cognizant solution* if for every $r \in \Sigma$, there is a target instance J_r such that (I, J_r) satisfies r and $\uplus J_r \subseteq J$, where $\uplus$ denotes the sum-union of bags (*i.e.*, the multiplicity of each element of the sum-union is equal to the sum of its multiplicities). The incognizant (resp. cognizant) *certain answers* to a query q w.r.t. Σ on I are defined using bag intersection of the answers over the incognizant (resp. cognizant) solutions for I w.r.t. Σ, *i.e.*, the multiplicity of an answer is the minimum of its multiplicities over the solutions. Note that for BCQs, the only possible certain answer is the empty tuple.

For example, consider $\Sigma = \{B(x) \rightarrow A(x), C(x) \rightarrow A(x)\}$ and $D = \{(B(a), 1), (C(a), 1)\}$. Under the incognizant semantics, the multiplicity of the certain answer of the Boolean query $A(a)$ w.r.t. Σ and D is 1 while under the cognizant semantics it is 2. Indeed, $J = \{(A(a), 1)\}$ is an incognizant solution for D w.r.t. Σ as it satisfies both s-t tgds, but is not a cognizant solution as the sum of multiplicities that arise from the two rules is 2.

It is easy to show that the cognizant semantics is equivalent to $\mathcal{P}^{\mathsf{AT}}$ on the counting semiring $\mathbb{N} = (\mathbb{N}, +, \times, 0, 1)$, and thus coincides with the classical bag semantics for Datalog. However, we have seen that the incognizant and cognizant semantics differ. Moreover, note that in the field of ontology-based data access, the bag semantics defined by [NKK$^+$19] for DL-Lite$_R$ coincides with the *incognizant* semantics, thus disagrees with the classical Datalog bag semantics [MPR90, GKT07].

We hence define a provenance semantics that coincides with these semantics when used with the counting semiring. Since it is based on greatest lower bounds, it is defined on a restricted class of semirings.

Let $\mathbb{K} = (K, +, \times, 0, 1)$ be a commutative ω-continuous semiring such that for every $K' \subseteq K$, the greatest lower bound $\inf(K')$ of K' is well defined (*i.e.*, there exists a unique $z \in K$ such that $z \sqsubseteq x$ for every $x \in K'$ and every z' such that $z' \sqsubseteq x$ for every $x \in K'$ is such that $z' \sqsubseteq z$), Σ be a Datalog program, and $(D, \mathbb{K}, \lambda)$ be an annotated database. We define $\mathbb{K}$-*annotated interpretations* as pairs (I, μ^I) where I is a set of facts, and μ^I is a function from I to K. We say that a $\mathbb{K}$-annotated interpretation (I, μ^I) is a *model* of Σ and $(D, \mathbb{K}, \lambda)$, denoted by $(I, \mu^I) \models (\Sigma, D, \mathbb{K}, \lambda)$, if

1. $D \subseteq I$, and for every $\alpha \in D$, $\lambda(\alpha) \sqsubseteq \mu^I(\alpha)$;
2. for every $\phi(\boldsymbol{x}, \boldsymbol{y}) \rightarrow H(\boldsymbol{x})$ in Σ, whenever there is a homomorphism $h : \phi(\boldsymbol{x}, \boldsymbol{y}) \mapsto I$, then $h(H(\boldsymbol{x})) \in I$ and $\displaystyle\sum_{h' : \phi(\boldsymbol{x},\boldsymbol{y}) \mapsto I, h'(\boldsymbol{x}) = h(\boldsymbol{x})} \prod_{\beta \in h'(\phi(\boldsymbol{x},\boldsymbol{y}))} \mu^I(\beta) \sqsubseteq \mu^I(h(H(\boldsymbol{x})))$.

The *annotated model-based provenance semantics* $\mathcal{P}^{\mathsf{AM}}$ is defined by

$$\mathcal{P}^{\mathsf{AM}}(\Sigma, D, \mathbb{K}, \lambda, \alpha) := \inf(\{\mu^I(\alpha) | (I, \mu^I) \models (\Sigma, D, \mathbb{K}, \lambda)\}).$$

Set-Annotated Model-based

We adapt the work on provenance for the description logics DL-Lite$_R$ and $\mathcal{ELH}^r$ [CLO$^+$19, BOPP20], where the semiring is assumed to be a $\times$-idempotent provenance semiring $Prov(X)$ and rules are also annotated. Annotated models of annotated knowledge bases are defined as set of facts annotated with sets of monomials from $Prov(X)$. Given a fact α and a monomial m over X, $(\Sigma, D, Prov(X), \lambda_X) \models (\alpha, m)$ holds when m belongs to the annotation set of α in every models of Σ and $(D, Prov(X), \lambda_X)$.

To obtain an analog provenance semantics for Datalog, we define interpretations which associate facts with (possibly infinite) *sets of annotations*, and formulate the requirements for them to be models of Σ and $(D, \mathbb{K}, \lambda)$.

Let $\mathbb{K} = (K, +, \times, 0, 1)$ be a commutative ω-continuous semiring, Σ be a Datalog program, and $(D, \mathbb{K}, \lambda)$ be an annotated database. We define $\mathbb{K}$*-set-annotated interpretations* as pairs (I, μ^I) where I is a set of facts, and μ^I is a function from I to the *power-set of* K. We say that a $\mathbb{K}$-set-annotated interpretation (I, μ^I) is a model of Σ and $(D, \mathbb{K}, \lambda)$, denoted by $(I, \mu^I) \models (\Sigma, D, \mathbb{K}, \lambda)$, if

1. $D \subseteq I$, and for every $\alpha \in D$, $\lambda(\alpha) \in \mu^I(\alpha)$;
2. for every $\phi(\boldsymbol{x}, \boldsymbol{y}) \to H(\boldsymbol{x})$ in Σ, whenever there is a homomorphism $h : \phi(\boldsymbol{x}, \boldsymbol{y}) \mapsto I$, then $h(H(\boldsymbol{x})) \in I$ and if $h(\phi(\boldsymbol{x}, \boldsymbol{y})) = \beta_1 \wedge \cdots \wedge \beta_n$, $\{\Pi_{i=1}^n k_i \mid (k_1, \ldots, k_n) \in \mu^I(\beta_1) \times \cdots \times \mu^I(\beta_n)\} \subseteq \mu^I(h(H(\boldsymbol{x})))$.

The *set-annotated model-based provenance semantics* $\mathcal{P}^{\mathsf{SAM}}$ is defined by

$$\mathcal{P}^{\mathsf{SAM}}(\Sigma, D, \mathbb{K}, \lambda, \alpha) := \sum_{k \in \bigcap_{(I,\mu^I) \models (\Sigma, D, \mathbb{K}, \lambda)} \mu^I(\alpha)} k.$$

Connections between semantics

Let $\sqsubseteq$ be the binary relation between provenance semantics such that $\mathcal{P} \sqsubseteq \mathcal{P}'$ if and only if $\mathcal{P}(\Sigma, D, \mathbb{K}, \lambda, \alpha) \sqsubseteq \mathcal{P}'(\Sigma, D, \mathbb{K}, \lambda, \alpha)$ for every Σ, $(D, \mathbb{K}, \lambda)$ and α on which $\mathcal{P}$ and $\mathcal{P}'$ are well-defined.

Proposition 6 *The following holds:*

$$\mathcal{P}^{AM} \sqsubseteq \mathcal{P}^{AT} \text{ and } \mathcal{P}^{SAM} \sqsubseteq \mathcal{P}^{AT}.$$

Moreover $\mathcal{P}^{\mathsf{AM}}$ and $\mathcal{P}^{\mathsf{SAM}}$ are incomparable.

Despite of their inherently different approaches, $\mathcal{P}^{\mathsf{AM}}$, $\mathcal{P}^{\mathsf{SAM}}$ and $\mathcal{P}^{\mathsf{AT}}$ coincide on a large class of semirings.

Proposition 7 *If $\mathbb{K}$ is a commutative + -idempotent ω-continuous semiring, then for every Σ, $(D, \mathbb{K}, \lambda)$, and α, $\mathcal{P}^{AM}(\Sigma, D, \mathbb{K}, \lambda, \alpha) = \mathcal{P}^{SAM}(\Sigma, D, \mathbb{K}, \lambda, \alpha) = \mathcal{P}^{AT}(\Sigma, D, \mathbb{K}, \lambda, \alpha)$.*

Additional insights on the connection between definitions can be gained by considering the provenance semiring $\mathbb{N}^\infty[\![X]\!]$: the monomials with non-zero coefficients are the same with all semantics but their coefficients may differ ($\mathcal{P}^{\mathsf{AT}}$ leading to the highest coefficients by Proposition 6).

Proposition 8 *Let λ_X be an injective function from D to X.*

- *A monomial occurs in* $\mathcal{P}^{AT}(\Sigma, D, \mathbb{N}^\infty[\![X]\!], \lambda_X, \alpha)$ *if and only if it occurs in* $\mathcal{P}^{AM}(\Sigma, D, \mathbb{N}^\infty[\![X]\!], \lambda_X, \alpha)$.
- $\mathcal{P}^{SAM}(\Sigma, D, \mathbb{N}^\infty[\![X]\!], \lambda_X, \alpha)$ *is obtained by setting all non-zero coefficients to* 1 *in* $\mathcal{P}^{AT}(\Sigma, D, \mathbb{N}^\infty[\![X]\!], \lambda_X, \alpha)$.

4.3.2 Execution- and Tree-Based Semantics

We saw that when annotations are present there is more than one way to define a model-based semantics for Datalog and that it differs from the all-tree semantics. We now investigate definitions based on classical Datalog evaluation algorithms.

We extend the notion of *immediate consequence operator* describing the application of rules onto facts, with the computation of annotation. To this end, we introduce the *annotation aware immediate consequence operator* T_Σ. Applying T_Σ on a set of annotated facts $(I, \mathbb{K}, \lambda)$ results in $(I_{T_\Sigma}, \mathbb{K}, \lambda_{T_\Sigma})$ where I_{T_Σ} is the result of applying the immediate consequence operator to Σ and I, and λ_{T_Σ} annotates facts in I_{T_Σ} with the relational provenance (over $(I, \mathbb{K}, \lambda)$) of the UCQ formed by the bodies of the rules that create them. Formally,

$$I_{T_\Sigma} := \{H(\boldsymbol{a}) \mid I \models \exists \boldsymbol{y}\, \phi(\boldsymbol{a}, \boldsymbol{y})\,,\ \phi(\boldsymbol{x}, \boldsymbol{y}) \to H(\boldsymbol{x}) \in \Sigma\}$$
$$\lambda_{T_\Sigma}(H(\boldsymbol{a})) := \sum_{\substack{h(\boldsymbol{x})=\boldsymbol{a},\, I \models h(\phi(\boldsymbol{x},\boldsymbol{y})) \\ \phi(\boldsymbol{x},\boldsymbol{y}) \to H(\boldsymbol{x}) \in \Sigma}} \prod_{\beta \in h(\phi(\boldsymbol{x},\boldsymbol{y}))} \lambda(\beta)$$

We define a union operator for annotated databases (over the same semiring): $(I, \mathbb{K}, \lambda) \cup (I', \mathbb{K}, \lambda') := (I \cup I', \mathbb{K}, \lambda'')$ where $\lambda''(\alpha) := \lambda(\alpha) + \lambda'(\alpha)$ where we slightly abuse notation by setting $\lambda(\alpha) = 0$ if $\alpha \notin I$, and $\lambda'(\alpha) = 0$ if $\alpha \notin I'$.

Naive Evaluation / All Trees

In the naive evaluation algorithm, all rules are applied in parallel until a fixpoint is reached. The 'annotation aware' version of it is as follows: We set $I^0_{\mathsf{n}}(\Sigma, D, \mathbb{K}, \lambda) := (D, \mathbb{K}, \lambda)$, and define inductively $I^{i+1}_{\mathsf{n}}(\Sigma, D, \mathbb{K}, \lambda) := T_\Sigma(I^i_{\mathsf{n}}(\Sigma, D, \mathbb{K}, \lambda)) \cup (D, \mathbb{K}, \lambda)$. Note that the subscript n of I_{n} is an abbreviation for 'naive', and the superscript i indicates how many times T_Σ was applied.

Let $(I^i_{\mathsf{n}}, \mathbb{K}, \lambda^i_{\mathsf{n}})$ denote $I^i_{\mathsf{n}}(\Sigma, D, \mathbb{K}, \lambda)$. We say that $I^i_{\mathsf{n}}(\Sigma, D, \mathbb{K}, \lambda)$ *converges* if there is some k such that $I^\ell_{\mathsf{n}} = I^k_{\mathsf{n}}$ for every $\ell \geqslant k$, and $\sup(\lambda^i_{\mathsf{n}}(\alpha))$ exists for every $\alpha \in I^k_{\mathsf{n}}$.

Proposition 9 *For every* $\Sigma, D, \mathbb{K}, \lambda$, *if* $\mathbb{K}$ *is* ω*-continuous then* $I^i_{\mathsf{n}}(\Sigma, D, \mathbb{K}, \lambda)$ *converges.*

In this case, we define $I^\infty_{\mathsf{n}} := I^k_{\mathsf{n}}$ and $\lambda^\infty_{\mathsf{n}} := \sup_{i \to \infty} \lambda^i_{\mathsf{n}}$. The *naive execution provenance semantics* $\mathcal{P}^{\mathsf{NE}}$ is defined by

$$\mathcal{P}^{\mathsf{NE}}(\Sigma, D, \mathbb{K}, \lambda, \alpha) := \begin{cases} \lambda^\infty_{\mathsf{n}}(\alpha) & \alpha \in I^\infty_{\mathsf{n}} \\ 0 & \text{otherwise} \end{cases}$$

and is equivalent to the all-tree semantics.

Proposition 10 *It holds that* $\mathcal{P}^{NE} = \mathcal{P}^{AT}$.

Optimized Naive Evaluation / Minimal Depth Trees

We consider an optimized version of the naive algorithm that stops as soon as the desired fact is derived. We define the 'annotation aware' version of this algorithm by $I^0_{\mathsf{o},\alpha}(\Sigma, D, \mathbb{K}, \lambda) := (D, \mathbb{K}, \lambda)$, and

$$I^{i+1}_{\mathsf{o},\alpha}(\Sigma, D, \mathbb{K}, \lambda) := \begin{cases} T_\Sigma(I^i_{\mathsf{o},\alpha}(\Sigma, D, \mathbb{K}, \lambda)) \cup (D, \mathbb{K}, \lambda) & \alpha \notin I^i_{\mathsf{o},\alpha} \\ I^i_{\mathsf{o},\alpha}(\Sigma, D, \mathbb{K}, \lambda) & \text{otherwise} \end{cases}$$

where $I^i_{\mathsf{o},\alpha}$ is such that $I^i_{\mathsf{o},\alpha}(\Sigma, D, \mathbb{K}, \lambda) := (I^i_{\mathsf{o},\alpha}, \mathbb{K}, \lambda^i_{\mathsf{o},\alpha})$.

Proposition 11 *For every $\Sigma, D, \mathbb{K}, \lambda$, and α such that $\Sigma, D \models \alpha$, there exists $k \geqslant 0$ such that $I^k_{\mathsf{o},\alpha}(\Sigma, D, \mathbb{K}, \lambda) = I^\ell_{\mathsf{o},\alpha}(\Sigma, D, \mathbb{K}, \lambda)$ for every $\ell \geqslant k$.*

With k as provided by Proposition 11, we define the *optimized execution provenance semantics* $\mathcal{P}^{\mathsf{OE}}$ by:

$$\mathcal{P}^{\mathsf{OE}}(\Sigma, D, \mathbb{K}, \lambda, \alpha) := \begin{cases} \lambda^k_{\mathsf{o},\alpha}(\alpha) & \alpha \in I^k_{\mathsf{o},\alpha} \\ 0 & \text{otherwise} \end{cases}$$

We show that an equivalent tree-based semantics can be obtained by considering only minimal depth trees for the desired fact. This approach has been considered useful, for example to present a 'small proof' for debugging [ZSS20]. Formally, let $\mathtt{depth}(t)$ denote the depth of tree t. We say that $t \in T^\Sigma_D(\alpha)$ is *of minimal depth* if for every $t' \in T^\Sigma_D(\alpha)$ it holds that $\mathtt{depth}(t) \leqslant \mathtt{depth}(t')$. The *minimal depth tree provenance semantics* $\mathcal{P}^{\mathsf{MDT}}$ is defined by

$$\mathcal{P}^{\mathsf{MDT}}(\Sigma, D, \mathbb{K}, \lambda, \alpha) := \sum_{\substack{t \in T^\Sigma_D(\alpha) \\ \text{is of minimal depth}}} \Lambda(t)$$

and is equivalent to the optimized naive execution.

Proposition 12 *It holds that $\mathcal{P}^{OE} = \mathcal{P}^{MDT}$.*

Seminaive Evaluation / Hereditary Minimal Depth Trees

In the seminaive evaluation algorithm, facts are derived only once. We introduce a new consequence operator Δ_Σ that derives only new facts and is defined as follows: $\Delta_\Sigma(I, \mathbb{K}, \lambda) := (I_{\Delta_\Sigma}, \mathbb{K}, \lambda_{\Delta_\Sigma})$ where $T_\Sigma(I, \mathbb{K}, \lambda) := (I_{T_\Sigma}, \mathbb{K}, \lambda_{T_\Sigma})$, $I_{\Delta_\Sigma} := I_{T_\Sigma} \backslash I$, and λ_{Δ_Σ} is the restriction of λ_{T_Σ} to I_{Δ_Σ}. We can now define the annotation aware version of the seminaive evaluation: $I^0_{\mathsf{sn}}(\Sigma, D, \mathbb{K}, \lambda) := (D, \mathbb{K}, \lambda)$ and $I^{i+1}_{\mathsf{sn}}(\Sigma, D, \mathbb{K}, \lambda) := I^i_{\mathsf{sn}}(\Sigma, D, \mathbb{K}, \lambda) \cup \Delta_\Sigma(I^i_{\mathsf{sn}}(\Sigma, D, \mathbb{K}, \lambda))$.

Proposition 13 *propsemi For every $\Sigma, D, \mathbb{K}, \lambda$, there exists $k \geqslant 0$ such that $I^k_{\mathsf{sn}}(\Sigma, D, \mathbb{K}, \lambda) = I^\ell_{\mathsf{sn}}(\Sigma, D, \mathbb{K}, \lambda)$ for every $\ell \geqslant k$.*

Note that, unlike in Proposition 9, we do not require $\mathbb{K}$ to be ω-continuous. With k provided by Proposition 13, the *seminaive execution provenance semantics* $\mathcal{P}^{\mathsf{SNE}}$ is defined by

$$\mathcal{P}^{\mathsf{SNE}}(\Sigma, D, \mathbb{K}, \lambda, \alpha) := \begin{cases} \lambda^k_{\mathsf{sn}} & \alpha \in I^k_{\mathsf{sn}} \\ 0 & \text{otherwise} \end{cases}$$

To capture this with the tree-based approach we need to further restrict all subtrees to be of minimal depth. Formally, a derivation tree $t \in T_D^\Sigma(\alpha)$ is a *hereditary minimal-depth (derivation) tree* if for every node n of t labeled by (β, r, h), the subtree t_β with root n is a minimal-depth derivation tree for β. The *hereditary minimal depth tree provenance semantics* $\mathcal{P}^{\mathsf{HMDT}}$ is defined by

$$\mathcal{P}^{\mathsf{HMDT}}(\Sigma, D, \mathbb{K}, \lambda, \alpha) := \sum_{\substack{t \in T_D^\Sigma(\alpha) \\ \text{is hereditary minimal-depth}}} \Lambda(t)$$

and is equivalent to the seminaive execution.

Proposition 14 *It holds that* $\mathcal{P}^{SNE} = \mathcal{P}^{HMDT}$.

4.3.3 Non-Recursive Tree-Based Semantics

Both execution-based semantics $\mathcal{P}^{\mathsf{OE}}$ and $\mathcal{P}^{\mathsf{SNE}}$ take into account finite subsets of derivation trees (and hence converge). Is there a more informative tree-based semantics (*i.e.*, one that takes into account a bigger subset of derivation trees) that still converges? We present such a semantics based on the intuition that deriving a fact from itself is redundant.

Formally, a *non-recursive (derivation) tree* is a derivation tree that does not contain two nodes labeled with the same fact and such that one is the descendant of the other. The *non-recursive tree provenance semantics* $\mathcal{P}^{\mathsf{NRT}}$ is defined by

$$\mathcal{P}^{\mathsf{NRT}}(\Sigma, D, \mathbb{K}, \lambda, \alpha) := \sum_{\substack{t \in T_D^\Sigma(\alpha) \\ \text{is non-recursive}}} \Lambda(t).$$

Connections between semantics

Next proposition follows from the fact that hereditary minimal-depth trees are of minimal-depth and non recursive. The sets of minimal depth trees and non-recursive trees are incomparable, so that $\mathcal{P}^{\mathsf{NRT}} \not\sqsubseteq \mathcal{P}^{\mathsf{MDT}}$ and $\mathcal{P}^{\mathsf{MDT}} \not\sqsubseteq \mathcal{P}^{\mathsf{NRT}}$.

Proposition 15 *The following hold:*

$$\mathcal{P}^{HMDT} \sqsubseteq \mathcal{P}^{NRT} \sqsubseteq \mathcal{P}^{AT} \quad \textit{and} \quad \mathcal{P}^{HMDT} \sqsubseteq \mathcal{P}^{MDT} \sqsubseteq \mathcal{P}^{AT}$$

Moreover $\mathcal{P}^{\mathsf{NRT}}$ and $\mathcal{P}^{\mathsf{AT}}$ coincide on specific semirings.

Proposition 16 *For every* $\Sigma, D, \mathbb{K}, \lambda$ *and* α*, if* $\mathbb{K}$ *is a commutative absorptive* ω*-continuous semiring, then* $\mathcal{P}^{NRT}(\Sigma, D, \mathbb{K}, \lambda, \alpha) = \mathcal{P}^{AT}(\Sigma, D, \mathbb{K}, \lambda, \alpha)$.

If $\mathbb{K}$ is not absorptive, there exists Σ, $(D, \mathbb{K}, \lambda)$ and α such that $\mathcal{P}^{\mathsf{NRT}}(\Sigma, D, \mathbb{K}, \lambda, \alpha) \neq \mathcal{P}^{\mathsf{AT}}(\Sigma, D, \mathbb{K}, \lambda, \alpha)$, even in the case where $\mathbb{K}$ is $+$-idempotent and $\times$-idempotent: Let Σ consist of the rule $A(x) \wedge B(x) \to A(x)$ and $D = \{A(a), B(a)\}$. Then $\mathcal{P}^{\mathsf{NRT}}(\Sigma, D, \mathbb{K}, \lambda, A(a)) = \lambda(A(a))$ while $\mathcal{P}^{\mathsf{AT}}(\Sigma, D, \mathbb{K}, \lambda, A(a)) = \lambda(A(a)) + \lambda(A(a)) \times \lambda(B(a))$.

The other semantics differ even under strong restrictions.

4.3.4 Basics Properties

We provide a framework allowing to compare the provenance semantics presented in the previous section. It is clear that they all fulfill the following definition.

Definition 9 (Provenance semantics) *A* provenance semantics *is a partial function that assigns to a Datalog program Σ, annotated database $(D, \mathbb{K}, \lambda)$ and fact α, an element $\mathcal{P}(\Sigma, D, \mathbb{K}, \lambda, \alpha)$ in K such that:*

1. *$\Sigma, D \not\models \alpha$ implies $\mathcal{P}(\Sigma, D, \mathbb{K}, \lambda, \alpha) = 0_{\mathbb{K}}$.*
2. *If $\mathbb{K}$ is positive, $\mathcal{P}(\Sigma, D, \mathbb{K}, \lambda, \alpha) = 0_{\mathbb{K}}$ implies $\Sigma, D \not\models \alpha$.*

We call the semiring domain *of $\mathcal{P}$ the maximal set S of semirings such that $\mathcal{P}(\Sigma, D, \mathbb{K}, \lambda, \alpha)$ is defined for every $\mathbb{K} \in S$, and every Σ, $(D, \mathbb{K}, \lambda)$ and α.*

Intuitively, Definition 9 means that the semantics reflects fact (non)-entailment. It is extremely permissive: We could define such a semantics that associates to each entailed fact a random semiring element different from zero, and does not bring any information beyond facts entailment. In the sequel, we state and discuss a number of properties that may be expected to be satisfied by a provenance semantics.

When not stated otherwise, $\mathcal{P}$, Σ, D, $\mathbb{K}$, λ and α denote respectively an arbitrary provenance semantics, Datalog program, database, commutative semiring $(K, +, \times, 0, 1)$, function from D to $K\backslash\{0\}$, and fact. We phrase properties as conditions, and say that $\mathcal{P}$ *satisfies* a property if it satisfies the condition. We also denote by λ_X an injective function $\lambda_X : D \mapsto X$.

4.3.5 Compatibility with Classical Notions

Property 1 is a sanity check: if a Datalog program amounts to a UCQ, the provenance should be the same as the one defined for relational databases [GKT07]. A Datalog program Σ is *UCQ-defined* if its rules are of the form $\phi(\boldsymbol{x}, \boldsymbol{y}) \rightarrow H(\boldsymbol{x})$ where H is a predicate that does not occur in the body of any rule. In this case, the equivalent UCQ Q^{Σ} of Σ is $\bigcup_{\phi(\boldsymbol{x},\boldsymbol{y})\rightarrow H(\boldsymbol{x})\in\Sigma} \exists \boldsymbol{y} \phi(\boldsymbol{x}, \boldsymbol{y})$.

Property 1 (Algebra Consistency) *If Σ is UCQ-defined with rule head $H(\boldsymbol{x})$ and $H \notin \mathcal{S}(D)$, then for every tuple $\boldsymbol{a}$ of same arity as $\boldsymbol{x}$, the relational provenance of $Q^{\Sigma}(\boldsymbol{a})$ is equal to $\mathcal{P}(\Sigma, D, \mathbb{K}, \lambda, H(\boldsymbol{a}))$.*

While Property 1 considers the behavior of a provenance semantics on a restricted class of queries, we can alternatively consider its behavior on a specific semiring. Boolean provenance has a very natural definition, based on the database subsets that entail the query, and is widely used, notably for probabilistic databases [Sen17], but also for ontology-mediated query explanation (e.g., in Datalog$^{+/-}$ or description logics [CLMV19, CLMV20]). It is formalized with the semiring $PosBool(X)$.

Property 2 (Boolean Compatibility)

$$\mathcal{P}(\Sigma, D, PosBool(X), \lambda_X, \alpha) = \bigvee_{\substack{D'\subseteq D \\ \Sigma, D'\models\alpha}} \bigwedge_{\beta\in D'} \lambda_X(\beta)$$

Property 2 expresses 'insensibility' to syntax, that is, every provenance semantics that satisfies Property 2 agrees on equivalent programs (*i.e.*, those that have the same models) for the semiring $PosBool(X)$. This is related to ideas from [Gre09] on the provenance of equivalent UCQs.

4.3.6 Compatibility with Specialization

Semiring provenance has been introduced to abstract from the particular semiring at hand, and factor the computations in some provenance semiring which specializes correctly to any semiring of interest. The next property allows one to do so, and is thus highly desirable.

Property 3 (Commutation with Homomorphisms) *If there is a semiring homomorphism h from $\mathbb{K}_1$ to $\mathbb{K}_2$, then $h(\mathcal{P}(\Sigma, D, \mathbb{K}_1, \lambda, \alpha)) = \mathcal{P}(\Sigma, D, \mathbb{K}_2, h \circ \lambda, \alpha)$.*

We call Property 3 restricted to ω-continuous homomorphisms *Commutation with ω-Continuous Homomorphisms.*

Specializing correctly is all the more useful when $\mathcal{P}$ is well-defined for a lot of semirings, in particular on all commutative or at least all commutative ω-continuous semirings.

Property 4 (Any (ω-Continuous) Semiring) *$\mathcal{P}$ satisfies the* Any Semiring Property *(resp.* Any ω-Continuous Semiring Property*) if the semiring domain of $\mathcal{P}$ contains the set of all commutative (resp. commutative ω-continuous) semirings.*

4.3.7 Joint and Alternative Use of the Data

How is the actual usage of the data reflected in the provenance semantics? The next property formalizes that multiplication reflects joint use of the data, and addition alternative use. For the rest of this section, we set goal to be a nullary predicate not in $\mathcal{S}(\Sigma) \cup \mathcal{S}(D)$.

Property 5 (Joint and Alternative Use) *For all tuples of facts $(\alpha_1^1, \cdots, \alpha_{n_1}^1), \ldots, (\alpha_1^m, \cdots, \alpha_{n_m}^m)$, it holds that*

$$\mathcal{P}(\Sigma', D, \mathbb{K}, \lambda, \mathit{goal}) = \Sigma_{i=1}^{m} \Pi_{j=1}^{n_i} \mathcal{P}(\Sigma, D, \mathbb{K}, \lambda, \alpha_j^i)$$

where $\Sigma' = \Sigma \cup \{\bigwedge_{j=1}^{n_i} \alpha_j^i \rightarrow \mathit{goal} \mid 1 \leqslant i \leqslant m\}$.

We weaken the above by referring to each mode separately:

Property 6 (Joint Use) *For all facts $\alpha_1, \cdots, \alpha_n$,*

$$\mathcal{P}(\Sigma', D, \mathbb{K}, \lambda, \mathit{goal}) = \Pi_{j=1}^{n} \mathcal{P}(\Sigma, D, \mathbb{K}, \lambda, \alpha_j)$$

where $\Sigma' = \Sigma \cup \{\bigwedge_{j=1}^{n} \alpha_j \rightarrow \mathit{goal}\}$.

Property 7 (Alternative Use) *For all facts $\alpha_1, \cdots, \alpha_m$,*

$$\mathcal{P}(\Sigma', D, \mathbb{K}, \lambda, \mathit{goal}) = \Sigma_{i=1}^{m} \mathcal{P}(\Sigma, D, \mathbb{K}, \lambda, \alpha_i)$$

where $\Sigma' = \Sigma \cup \{\alpha_i \rightarrow \mathit{goal} \mid 1 \leqslant i \leqslant m\}$.

4.3.8 Fact Roles in Entailment.

After considering how facts can be combined or used alternatively to entail a result, we ponder their possible roles w.r.t. the entailment. Property 8 asserts that the original annotation of a fact takes part in the provenance of its entailment.

Property 8 (Self) *If* $\alpha \in D$, *then* $\lambda(\alpha) \sqsubseteq \mathcal{P}(\Sigma, D, \mathbb{K}, \lambda, \alpha)$.

Moreover, if a database fact cannot be alternatively derived using the rules, then its provenance should be exactly its original annotation. To phrase this property we use the *grounding* Σ_D of Σ w.r.t. D, defined by $\Sigma_D = \{h(\phi(\boldsymbol{x}, \boldsymbol{y})) \rightarrow h(H(\boldsymbol{x})) \mid \phi(\boldsymbol{x}, \boldsymbol{y}) \rightarrow H(\boldsymbol{x}) \in \Sigma, h : \boldsymbol{x} \cup \boldsymbol{y} \mapsto \mathcal{D}(D)\}$. It holds that $\Sigma, D \models \alpha$ if and only if $\Sigma_D, D \models \alpha$.

Property 9 (Parsimony) *If* $\alpha \in D$ *does not occur in any rule head in* Σ_D *then* $\mathcal{P}(\Sigma, D, \mathbb{K}, \lambda, \alpha) = \lambda(\alpha)$.

Property 10 states that $\mathcal{P}$ reflects the necessity of a fact for the entailment. We say that $\beta \in D$ is *necessary* to $\Sigma, D \models \alpha$ if $\Sigma, D \backslash \{\beta\} \not\models \alpha$, and denote by Nec the set of such facts.

Property 10 (Necessary Facts) *There exists* $e \in K$ *such that* $\mathcal{P}(\Sigma, D, \mathbb{K}, \lambda, \alpha) = \Pi_{\beta \in Nec} \lambda(\beta) \times e$.

A fact is *usable* to $\Sigma, D \models \alpha$ if it occurs in some derivation tree in $T_D^{\Sigma}(\alpha)$. Usable facts are related to the notion of *lineage* [CWW00] and can be defined without resorting to derivation trees. Intuitively, if a fact is not usable to derive another fact, it should not have any influence on its provenance.

Property 11 (Non-Usable Facts) *For every* λ' *that differs from* λ *only on facts that are not usable to* $\Sigma, D \models \alpha$, *it holds that* $\mathcal{P}(\Sigma, D, \mathbb{K}, \lambda, \alpha) = \mathcal{P}(\Sigma, D, \mathbb{K}, \lambda', \alpha)$.

4.3.9 Data Modification

The last two properties indicate how provenance is impacted when facts are inserted or deleted.

Property 12 (Insertion) *For every* $(D', \mathbb{K}, \lambda')$ *such that* $D \cap D' = \varnothing$,
$\mathcal{P}(\Sigma, D, \mathbb{K}, \lambda, \alpha) + \mathcal{P}(\Sigma, D', \mathbb{K}, \lambda', \alpha)$

$$\sqsubseteq \mathcal{P}(\Sigma, D \cup D', \mathbb{K}, \lambda \cup \lambda', \alpha).$$

Maintaining provenance upon fact deletion is very useful in practice. We formalize this using a provenance semiring, which allows us to keep track of the facts. A partial evaluation of a provenance expression $p(X)$ over variables X is an expression obtained from $p(X)$ by replacing some of the variables by a given value.

Property 13 (Deletion) *For every provenance semiring* $Prov(X)$ *and* $D' \subseteq D$, *if* λ' *is the restriction of* λ_X *to* D' *and* $\Delta = D \backslash D'$, *then* $\mathcal{P}(\Sigma, D', Prov(X), \lambda', \alpha)$ *is equal to the partial evaluation of* $\mathcal{P}(\Sigma, D, Prov(X), \lambda_X, \alpha)$ *obtained by setting the annotations of facts in* Δ *to* 0: $\mathcal{P}(\Sigma, D, Prov(X), \lambda_X, \alpha)[\{\lambda_X(x) = 0\}_{x \in \Delta}]$.

4.3.10 Semantics Analysis w.r.t. Properties

In this subsection, we analyze the semantics proposed in Section 4.3 w.r.t. the properties introduced in Section 4.3.4. The properties each semantics satisfies are summarized in Table 4.1. Proofs of the positive cases are given in the appendix of [BBPT22b]

	$\mathcal{P}^{\mathsf{AT}}$	$\mathcal{P}^{\mathsf{NRT}}$	$\mathcal{P}^{\mathsf{MDT}}$	$\mathcal{P}^{\mathsf{HMDT}}$	$\mathcal{P}^{\mathsf{AM}}$	$\mathcal{P}^{\mathsf{SAM}}$
Algebra Consistency	✓	✓	✓	✓		
Boolean Compat.	✓	✓			✓	✓
Com. with Hom.		✓	✓	✓		
Com. with ω-Cont.	✓	✓	✓	✓		
Any Semiring		✓	✓	✓		
Any ω-Cont. Sem.	✓	✓	✓	✓		✓
Joint and Alt. Use	✓	✓				
Joint Use	✓	✓		✓	✓	
Alternative Use	✓	✓				
Self	✓	✓	✓	✓	✓	✓
Parsimony	✓	✓	✓	✓	✓	✓
Necessary Facts	✓	✓	✓	✓		✓
Non-Usable Facts	✓	✓	✓	✓	✓	✓
Insertion	✓	✓				
Deletion	✓	✓			✓	✓

Table 4.1: Does a property hold for a provenance semantics?

Chapter 5

Conclusion

This manuscript has focused on my work about the dialogue between Logic, Tree and Circuit and some of my main theoretical results about them.

Over the last five years, I have also worked on applying these results to different cases. Thanks to my colleagues in SPIRALS team, I have found several use cases for applying and extending my work. In the context Software engineering, models are a key approach in order to configure applications built from different composents in particular in the context of distributed computation. In this context, an approach has been proposed to describe different configurations of an application by using a language of description called Feature Model [MP14]. Interestingly, this language is equivalent to the notion of even trees presented in [SA07]. Despite most of the querying problems over Feature Models are NP-hard, by using a compilation method through d-DNNF thanks to the system d4, we are investigating practical query processing over Feature Model. Another model for describing distribution systems is based on logic to describe the properties of the systems. Unfortunately, it is complex to check the consistency or the redundancy of these logical descriptions. As the language is very expressive, the problem is undecidable and only partial approaches to solve this problem. We are investigating if the works in satisfiability in logic such as Description Logics, GNFO can be used in this context to give exact answers. Within another axe of the team, I have also started different security problems such as security on access control [BBtC$^+$21, BBJT21, BBJT19], analyzing AdBlocker through logical approaches.

Through other collaborations, I have started to work on Digital Humanities: on similarity of theatre plays based on the notion of parametrized edition distance on words by co-advising Aaron Boussidan with Philippe Gambette. I have been working with Ekaterina Nechaeva, Simon Bliudze and Lionel Seinturier on the management of uncertain data in History in particular through the lens of provenance.

With Philippe Gambette, Sarah Berkemer and Lionel Seinturier, we are investigating the use of Datalog to query phylogenetic networks used in Bio Computing.

More generally, my current research aims to build more trusted systems in particular through the lens of data management: by adapting general techniques in database management to specific problems from other fields as presented earlier, by understanding how to create collaborative systems which respect the law. On this last point, I have worked with Juliette Sénéchal and Lionel Seinturier for a year on understanding European Legal texts and their impacts on building systems in particular through data management. In particular, we have answered different concertations of the CNIL, the French authority monitoring the use of the data in society regarding the regulations. These concertations have been done to improve their notes to help users how to manage their data.

Building systems that manage data in a trustworthy manner is what will lead my research in the next years.

Bibliography

[ABBV18] Antoine Amarilli, Michael Benedikt, Pierre Bourhis, and Michael Vanden Boom. Query answering with transitive and linear-ordered data. *J. Artif. Intell. Res.*, 63:191–264, 2018.

[ABCM24] Antoine Amarilli, Pierre Bourhis, Florent Capelli, and Mikaël Monet. Ranked enumeration for MSO on trees via knowledge compilation. In Graham Cormode and Michael Shekelyan, editors, *27th International Conference on Database Theory, ICDT 2024, March 25-28, 2024, Paestum, Italy*, volume 290 of *LIPIcs*, pages 25:1–25:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[ABJM17] Antoine Amarilli, Pierre Bourhis, Louis Jachiet, and Stefan Mengel. A circuit-based approach to efficient enumeration. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 111:1–111:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[ABJM19] Antoine Amarilli, Pierre Bourhis, Louis Jachiet, and Stefan Mengel. A Circuit-based approach to efficient enumeration, 2019. Extended version with proofs. `https://arxiv.org/abs/1702.05589`.

[ABMN19a] Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Constant-delay enumeration for nondeterministic document spanners. In Pablo Barceló and Marco Calautti, editors, *22nd International Conference on Database Theory, ICDT 2019, March 26-28, 2019, Lisbon, Portugal*, volume 127 of *LIPIcs*, pages 22:1–22:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[ABMN19b] Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Enumeration on trees with tractable combined complexity and efficient updates. In Dan Suciu, Sebastian Skritek, and Christoph Koch, editors, *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 89–103. ACM, 2019.

[ABMN21] Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Constant-delay enumeration for nondeterministic document spanners. *ACM Trans. Database Syst.*, 46(1):2:1–2:30, 2021.

[ABMS19] Antoine Amarilli, Pierre Bourhis, Mikaël Monet, and Pierre Senellart. Evaluating datalog via tree automata and cycluits. *Theory Comput. Syst.*, 63(7):1620–1678, 2019.

[ABS15a] Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Provenance circuits for trees and treelike instances. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 56–68. Springer, 2015.

[ABS15b] Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Provenance circuits for trees and treelike instances (extended version), November 2015. CoRR abs/1511.08723.

[ABV18] Serge Abiteboul, Pierre Bourhis, and Victor Vianu. Explanations and transparency in collaborative workflows. In Jan Van den Bussche and Marcelo Arenas, editors, *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018*, pages 409–424. ACM, 2018.

[AHU74] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The design and analysis of computer algorithms*. Addison-Wesley, 1974.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. 1995.

[AN01] André Arnold and Damian Niwiński. *Rudiments of mu-calculus*. North Holland, 2001.

[ANvB98] H. Andréka, I. Németi, and J. van Benthem. Modal languages and bounded fragments of predicate logic. *JPL*, 27(3):217–274, 1998.

[AVFY00] Serge Abiteboul, Victor Vianu, Bradley S. Fordham, and Yelena Yesha. Relational transducers for electronic commerce. *J. Comput. Syst. Sci.*, 61(2):236–269, 2000.

[AVV97] Serge Abiteboul, Moshe Y. Vardi, and Victor Vianu. Fixpoint logics, relational machines, and computational complexity. *J. ACM*, 44(1):30–56, 1997.

[Bag06] Guillaume Bagan. MSO queries on tree decomposable structures are computable with linear delay. In Zoltán Ésik, editor, *Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings*, volume 4207 of *Lecture Notes in Computer Science*, pages 167–181. Springer, 2006.

[Bar13] Pablo Barceló. Querying graph databases. In *PODS*, 2013.

[BBGS20] Michael Benedikt, Pierre Bourhis, Georg Gottlob, and Pierre Senellart. Monadic datalog, tree validity, and limited access containment. *ACM Trans. Comput. Log.*, 21(1):6:1–6:45, 2020.

[BBJT19] Michael Benedikt, Pierre Bourhis, Louis Jachiet, and Michaël Thomazo. Reasoning about disclosure in data integration in the presence of source constraints. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 1551–1557. ijcai.org, 2019.

[BBJT21] Michael Benedikt, Pierre Bourhis, Louis Jachiet, and Efthymia Tsamoura. Balancing expressiveness and inexpressiveness in view design. *ACM Trans. Database Syst.*, 46(4):15:1–15:40, 2021.

[BBPT22a] Camille Bourgaux, Pierre Bourhis, Liat Peterfreund, and Michaël Thomazo. Revisiting semiring provenance for datalog. In Gabriele Kern-Isberner, Gerhard Lakemeyer, and Thomas Meyer, editors, *Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning, KR 2022, Haifa, Israel, July 31 - August 5, 2022*, 2022.

[BBPT22b] Camille Bourgaux, Pierre Bourhis, Liat Peterfreund, and Michaël Thomazo. Revisiting semiring provenance for Datalog, 2022. arxiv.org/abs/2202.10766.

[BBS12] Michael Benedikt, Pierre Bourhis, and Pierre Senellart. Monadic datalog containment. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, volume 7392 of *Lecture Notes in Computer Science*, pages 79–91. Springer, 2012.

[BBtC13] Vince Bárány, Michael Benedikt, and Balder ten Cate. Rewriting guarded negation queries. In *MFCS*, 2013.

[BBtC+21] Michael Benedikt, Pierre Bourhis, Balder ten Cate, Gabriele Puppis, and Michael Vanden Boom. Inference from visible information and background knowledge. *ACM Trans. Comput. Log.*, 22(2):13:1–13:69, 2021.

[BBV16] Michael Benedikt, Pierre Bourhis, and Michael Vanden Boom. A step up in expressiveness of decidable fixpoint logics. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 817–826. ACM, 2016.

[BBV17] Michael Benedikt, Pierre Bourhis, and Michael Vanden Boom. Characterizing definability in decidable fixpoint logics. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 107:1–107:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[BBV19] Michael Benedikt, Pierre Bourhis, and Michael Vanden Boom. Definability and interpolation within decidable fixpoint logics. *Log. Methods Comput. Sci.*, 15(3), 2019.

[BCK+14] Achim Blumensath, Thomas Colcombet, Denis Kuperberg, Pawel Parys, and Michael Vanden Boom. Two-way cost automata and cost logics over infinite trees. In *CSL-LICS*, 2014.

[BCOS14] Meghyn Bienvenu, Diego Calvanese, Magdalena Ortiz, and Mantas Simkus. Nested regular path queries in description logics. In *KR*, 2014.

[BDM16] Pierre Bourhis, Daniel Deutch, and Yuval Moskovitch. Analyzing data-centric applications: Why, what-if, and how-to. In *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016*, pages 779–790. IEEE Computer Society, 2016.

[BDM20] Pierre Bourhis, Daniel Deutch, and Yuval Moskovitch. Equivalence-invariant algebraic provenance for hyperplane update queries. In David Maier, Rachel Pottinger, AnHai Doan,

Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo, editors, *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, pages 415–429. ACM, 2020.

[BGJR21] Pierre Bourhis, Alejandro Grez, Louis Jachiet, and Cristian Riveros. Ranked enumeration of MSO logic on words. In Ke Yi and Zhewei Wei, editors, *24th International Conference on Database Theory, ICDT 2021, March 23-26, 2021, Nicosia, Cyprus*, volume 186 of *LIPIcs*, pages 20:1–20:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[BKR15] Pierre Bourhis, Markus Krötzsch, and Sebastian Rudolph. Reasonable highly expressive query languages - IJCAI-15 distinguished paper (honorary mention). In Qiang Yang and Michael J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 2826–2832. AAAI Press, 2015.

[BL16] Pierre Bourhis and Carsten Lutz. Containment in monadic disjunctive datalog, mmsnp, and expressive description logics. In Chitta Baral, James P. Delgrande, and Frank Wolter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016*, pages 207–216. AAAI Press, 2016.

[BMMP16] Pierre Bourhis, Marco Manna, Michael Morak, and Andreas Pieris. Guarded-based disjunctive tuple-generating dependencies. *ACM Trans. Database Syst.*, 41(4):27:1–27:45, 2016.

[BMS08] Henrik Björklund, Wim Martens, and Thomas Schwentick. Optimizing conjunctive queries over trees using schema information. In *MFCS*, 2008.

[BMS13] Henrik Björklund, Wim Martens, and Thomas Schwentick. Validity of tree pattern queries with respect to schema information. In *MFCS*, 2013.

[BO96] Gerth Stolting Brodal and Chris Okasaki. Optimal purely functional priority queues. *Journal of Functional Programming*, 6(6), 1996.

[BOPP20] Camille Bourgaux, Ana Ozaki, Rafael Peñaloza, and Livia Predoiu. Provenance for the description logic elhr. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 1862–1869. ijcai.org, 2020.

[BPV04] Andrey Balmin, Yannis Papakonstantinou, and Victor Vianu. Incremental validation of XML documents. *ACM Trans. Database Syst.*, 29(4):710–751, 2004.

[BRV14] Pablo Barceló, Miguel Romero, and Moshe Y Vardi. Does query evaluation tractability help query containment? In *PODS*, 2014.

[BtCCV15] Michael Benedikt, Balder ten Cate, Thomas Colcombet, and Michael Vanden Boom. The complexity of boundedness for guarded logics. In *LICS*, 2015.

[BtCS15] Vince Bárány, Balder ten Cate, and Luc Segoufin. Guarded negation. *J. ACM*, 62(3):22:1–22:26, 2015.

[BtCV14] Michael Benedikt, Balder ten Cate, and Michael Vanden Boom. Effective interpolation and preservation in guarded logics. In *LICS*, 2014.

[CDG+02] Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. Tree automata techniques and applications. Available at `http://www.grappa.univ-lille3.fr/tata/`, 2002.

[CDLV00] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzeniri, and Moshe Y. Vardi. Containment of conjunctive regular path queries with inverse. In *KR*, 2000.

[CGKV88] Stavros S. Cosmadakis, Haim Gaifman, Paris C. Kanellakis, and Moshe Y. Vardi. Decidable optimization problems for database logic programs (preliminary report). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 477–490. ACM, 1988.

[Chl86] Bogdan S. Chlebus. Domino-tiling games. *J. Comput. Syst. Sci.*, 32(3):374–392, 1986.

[CLMV19] İsmail İlkan Ceylan, Thomas Lukasiewicz, Enrico Malizia, and Andrius Vaicenavicius. Explanations for query answers under existential rules. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 1639–1646. ijcai.org, 2019.

[CLMV20] İsmail İlkan Ceylan, Thomas Lukasiewicz, Enrico Malizia, and Andrius Vaicenavicius. Explanations for ontology-mediated query answering in description logics. In Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang, editors, *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 672–679. IOS Press, 2020.

[CLO+19] Diego Calvanese, Davide Lanti, Ana Ozaki, Rafael Peñaloza, and Guohui Xiao. Enriching ontology-based data access with provenance. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 1616–1623. ijcai.org, 2019.

[CV92] Surajit Chaudhuri and Moshe Y. Vardi. On the equivalence of recursive and nonrecursive Datalog programs. In *PODS*, 1992.

[CV97] Surajit Chaudhuri and Moshe Y. Vardi. On the equivalence of recursive and nonrecursive Datalog programs. *JCSS*, 54(1), 1997.

[CWW00] Yingwei Cui, Jennifer Widom, and Janet L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Trans. Database Syst.*, 25(2):179–227, 2000.

[DFKM22] Daniel Deutch, Nave Frost, Benny Kimelfeld, and Mikaël Monet. Computing the shapley value of facts in query answering. In Zachary G. Ives, Angela Bonifati, and Amr El Abbadi, editors, *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, pages 1570–1583. ACM, 2022.

[DGM18] Daniel Deutch, Amir Gilad, and Yuval Moskovitch. Efficient provenance tracking for datalog using top-k queries. *VLDB J.*, 27(2):245–269, 2018.

[DH00] Giovanna D'Agostino and Marco Hollenberg. Logical Questions Concerning The mu-Calculus: Interpolation, Lyndon and Los-Tarski. *JSL*, 65(1):310–332, 2000.

[DHK22] Shaleen Deep, Xiao Hu, and Paraschos Koutris. Ranked enumeration of join queries with projections. *arXiv preprint arXiv:2201.05566*, 2022.

[DK19] Shaleen Deep and Paraschos Koutris. Ranked enumeration of conjunctive query results. *arXiv preprint arXiv:1902.02698*, 2019.

[DMRT14] Daniel Deutch, Tova Milo, Sudeepa Roy, and Val Tannen. Circuits for datalog provenance. In Nicole Schweikardt, Vassilis Christophides, and Vincent Leroy, editors, *Proc. 17th International Conference on Database Theory (ICDT), Athens, Greece, March 24-28, 2014*, pages 201–212. OpenProceedings.org, 2014.

[EJ88] E. Allen Emerson and Charanjit S. Jutla. The complexity of tree automata and logics of programs (extended abstract). In *FOCS*, 1988.

[FRU+18] Fernando Florenzano, Cristian Riveros, Martin Ugarte, Stijn Vansummeren, and Domagoj Vrgoc. Constant delay algorithms for regular document spanners. In *PODS*, 2018.

[GHO02] Erich Grädel, Colin Hirsch, and Martin Otto. Back and forth between guarded and modal logics. *ACM TOCL*, 3(3):418–463, 2002.

[GJ22] Étienne Grandjean and Louis Jachiet. Which arithmetic operations can be performed in constant time in the RAM model with addition? *arXiv preprint arXiv:2206.13851*, 2022.

[GKT07] Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In Leonid Libkin, editor, *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 11-13, 2007, Beijing, China*, pages 31–40. ACM, 2007.

[GPW10] Georg Gottlob, Reinhard Pichler, and Fang Wei. Monadic Datalog over finite structures of bounded treewidth. *TOCL*, 12(1), 2010.

[Grä99] Erich Grädel. On the restraining power of guards. *JSL*, 64(4):1719–1742, 1999.

[Grä02] Erich Grädel. Guarded fixed point logics and the monadic theory of countable trees. *TCS*, 288(1):129 – 152, 2002.

[Gre09] Todd J. Green. Containment of conjunctive queries on annotated relations. In Ronald Fagin, editor, *Database Theory - ICDT 2009, 12th International Conference, St. Petersburg, Russia, March 23-25, 2009, Proceedings*, volume 361 of *ACM International Conference Proceeding Series*, pages 296–309. ACM, 2009.

[GS97] Ferenc Gécseg and Magnus Steinby. "Tree Languages". In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, chapter 1, pages 1–68. Springer Verlag, 1997.

[GW99] Erich Grädel and Igor Walukiewicz. Guarded fixed point logic. In *LICS*, 1999.

[HK17] André Hernich and Phokion G. Kolaitis. Foundations of information integration under bag semantics. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017.

[IL84] Tomasz Imielinski and Witold Lipski, Jr. Incomplete information in relational databases. *J. ACM*, 31(4), 1984.

[JLMS18] Jean Christoph Jung, Carsten Lutz, Mauricio Martel, and Thomas Schneider. Querying the unary negation fragment with regular path expressions. In Benny Kimelfeld and Yael Amsterdamer, editors, *21st International Conference on Database Theory, ICDT 2018, March 26-29, 2018, Vienna, Austria*, volume 98 of *LIPIcs*, pages 15:1–15:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[JS13] Abhay Kumar Jha and Dan Suciu. Knowledge compilation meets database theory: Compiling queries to decision diagrams. *Theory Comput. Syst.*, 52(3):403–440, 2013.

[JW95] David Janin and Igor Walukiewicz. Automata for the modal mu-calculus and related results. In *MFCS*, 1995.

[JW96] David Janin and Igor Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In *CONCUR*, 1996.

[KMN22] Sarah Kleest-Meißner, Jonas Marasus, and Matthias Niewerth. MSO queries on trees: Enumerating answers under updates using forest algebras. *CoRR*, abs/2208.04180, 2022.

[KNP+24] Mahmoud Abo Khamis, Hung Q. Ngo, Reinhard Pichler, Dan Suciu, and Yisu Remy Wang. Convergence of datalog over (pre-) semirings. *J. ACM*, 71(2):8:1–8:55, 2024.

[KPT06] Phokion G. Kolaitis, Jonathan Panttaja, and Wang Chiew Tan. The complexity of data exchange. In Stijn Vansummeren, editor, *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 26-28, 2006, Chicago, Illinois, USA*, pages 30–39. ACM, 2006.

[KS13] Wojciech Kazana and Luc Segoufin. Enumeration of monadic second-order queries on trees. *ACM Trans. Comput. Log.*, 14(4):25:1–25:12, 2013.

[KV17] Adrien Koutsos and Victor Vianu. Process-centric views of data-driven business artifacts. *J. Comput. Syst. Sci.*, 86:82–107, 2017.

[LM14] Katja Losemann and Wim Martens. MSO queries on trees: enumerating answers under updates. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 67:1–67:10. ACM, 2014.

[MP14] Andreas Metzger and Klaus Pohl. Software product line engineering and variability management: achievements and challenges. In James D. Herbsleb and Matthew B. Dwyer, editors, *Proceedings of the on Future of Software Engineering, FOSE 2014, Hyderabad, India, May 31 - June 7, 2014*, pages 70–84. ACM, 2014.

[MPR90] Inderpal Singh Mumick, Hamid Pirahesh, and Raghu Ramakrishnan. The magic of duplicates and aggregates. In Dennis McLeod, Ron Sacks-Davis, and Hans-Jörg Schek, editors, *16th International Conference on Very Large Data Bases, August 13-16, 1990, Brisbane, Queensland, Australia, Proceedings*, pages 264–277. Morgan Kaufmann, 1990.

[Nev02] Frank Neven. Automata theory for XML researchers. *SIGMOD Record*, 31(3):39–46, 2002.

[Nie18] Matthias Niewerth. MSO queries on trees: Enumerating answers under updates using forest algebras. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 769–778. ACM, 2018.

[NKK+19] Charalampos Nikolaou, Egor V. Kostylev, George Konstantinidis, Mark Kaminski, Bernardo Cuenca Grau, and Ian Horrocks. Foundations of ontology-based data access under bag semantics. *Artif. Intell.*, 274:91–132, 2019.

[NS18] Matthias Niewerth and Luc Segoufin. Enumeration of MSO queries on strings with constant delay and logarithmic updates. In Jan Van den Bussche and Marcelo Arenas, editors, *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018*, pages 179–191. ACM, 2018.

[PFKK19] Liat Peterfreund, Dominik D. Freydenberger, Benny Kimelfeld, and Markus Kröll. Complexity bounds for relational algebra over document spanners. In Dan Suciu, Sebastian Skritek, and Christoph Koch, editors, *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 320–334. ACM, 2019.

[Rab69] Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. Soc.*, 141:1–35, 1969.

[RK13] Sebastian Rudolph and Markus Krötzsch. Flag & check: data access with monadically defined queries. In *PODS*, 2013.

[SA07] Pierre Senellart and Serge Abiteboul. On the complexity of managing probabilistic XML data. In Leonid Libkin, editor, *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 11-13, 2007, Beijing, China*, pages 283–292. ACM, 2007.

[SdBBA19] Andy Shih, Guy Van den Broeck, Paul Beame, and Antoine Amarilli. Smoothing structured decomposable circuits. In *NeurIPS*, 2019.

[Sen17] Pierre Senellart. Provenance and probabilities in relational databases. *SIGMOD Rec.*, 46(4):5–15, 2017.

[StC13] Luc Segoufin and Balder ten Cate. Unary negation. *Log. Methods Comput. Sci.*, 9(3), 2013.

[TGR22] Nikolaos Tziavelis, Wolfgang Gatterbauer, and Mirek Riedewald. Any-k algorithms for enumerating ranked answers to conjunctive queries. *arXiv preprint arXiv:2205.05649*, 2022.

[Tho97] Wolfgang Thomas. Languages, Automata, and Logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*. 1997.

[TW68] James W. Thatcher and Jesse B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Math. Systems Theory*, 2(1), 1968.

[Var98] Moshe Y. Vardi. Reasoning about the past with two-way automata. In *ICALP*, 1998.

[Wid96] Jennifer Widom. The starburst active database rule system. *IEEE Trans. Knowl. Data Eng.*, 8(4):583–595, 1996.

[ZSS20] David Zhao, Pavle Subotic, and Bernhard Scholz. Debugging large-scale datalog: A scalable provenance evaluation strategy. *ACM Trans. Program. Lang. Syst.*, 42(2):7:1–7:35, 2020.

Appendices

Publications from February 2011 to June 2024

The following is my list of publications, since my PhD defense in February 2011. The majority have taken place either in a supporting role to a doctorate student, for example, during my postdoctorate, or directly in a supervising role since I obtained my position as a CNRS researcher.

Journal articles

- Michael Benedikt, Pierre Bourhis, Balder ten Cate, Gabriele Puppis, Michael Vanden Boom: Inference from Visible Information and Background Knowledge. ACM Trans. Comput. Log. 22(2): 13:1-13:69 (2021)
- Antoine Amarilli, Pierre Bourhis, Stefan Mengel, Matthias Niewerth: Constant-Delay Enumeration for Nondeterministic Document Spanners. ACM Trans. Database Syst. 46(1): 2:1-2:30 (2021)
- Pierre Bourhis, Juan L. Reutter, Domagoj Vrgoc: JSON: Data model and query languages. Inf. Syst. 89: 101478 (2020)
- Sihem Amer-Yahia, Senjuti Basu Roy, Lei Chen, Atsuyuki Morishima, James Abello Monedero, Pierre Bourhis, François Charoy, Marina Danilevsky, Gautam Das, Gianluca Demartini, Shady Elbassuoni, David Gross-Amblard, Emilie Hoareau, Munenari Inoguchi, Jared B. Kenworthy, Itaru Kitahara, Dongwon Lee, Yunyao Li, Ria Mae Borromeo, Paolo Papotti, Raghav Rao, Sudeepa Roy, Pierre Senellart, Keishi Tajima, Saravanan Thirumuruganathan, Marion Tommasi, Kazutoshi Umemoto, Andrea Wiggins, Koichiro Yoshida: Making AI Machines Work for Humans in FoW. SIGMOD Rec. 49(2): 30-35 (2020)
- Antoine Amarilli, Pierre Bourhis, Stefan Mengel, Matthias Niewerth: Constant-Delay Enumeration for Nondeterministic Document Spanners. SIGMOD Rec. 49(1): 25-32 (2020)
- Pierre Bourhis, Juan L. Reutter, Domagoj Vrgoc: JSON: Data model and query languages. Inf. Syst. 89: 101478 (2020)
- Michael Benedikt, Pierre Bourhis, Georg Gottlob, Pierre Senellart: Monadic Datalog, Tree Validity, and Limited Access Containment. ACM Trans. Comput. Log. 21(1): 6:1-6:45 (2020)
- Pierre Bourhis, Gianluca Demartini, Shady Elbassuoni, Emilie Hoareau, H. Raghav Rao: Ethical Challenges in the Future of Work. IEEE Data Eng. Bull. 42(4): 55-64 (2019)
- Michael Benedikt, Pierre Bourhis, Michael Vanden Boom: Definability and Interpolation within Decidable Fixpoint Logics. Log. Methods Comput. Sci. 15(3) (2019)
- Antoine Amarilli, Pierre Bourhis, Mikaël Monet, Pierre Senellart: Evaluating Datalog via Tree Automata and Cycluits. Theory Comput. Syst. 63(7): 1620-1678 (2019)
- Antoine Amarilli, Michael Benedikt, Pierre Bourhis, Michael Vanden Boom: Query Answering with Transitive and Linear-Ordered Data. J. Artif. Intell. Res. 63: 191-264 (2018)

- Pierre Bourhis, Marco Manna, Michael Morak, Andreas Pieris: Guarded-Based Disjunctive Tuple-Generating Dependencies. ACM Trans. Database Syst. 41(4): 27:1-27:45 (2016)
- Pierre Bourhis, Gabriele Puppis, Cristian Riveros, Slawek Staworko: Bounded Repairability for Regular Tree Languages. ACM Trans. Database Syst. 41(3): 18 (2016)
- Serge Abiteboul, Pierre Bourhis, Victor Vianu: Highly Expressive Query Languages for Unordered Data Trees. Theory Comput. Syst. 57(4): 927-966 (2015)
- Pierre Bourhis, Gabriele Puppis, Cristian Riveros: Which XML Schemas are Streaming Bounded Repairable? Theory Comput. Syst. 57(4): 1250-1321 (2015)
- Michael Benedikt, Pierre Bourhis, Clemens Ley: Analysis of Schemas with Access Restrictions. ACM Trans. Database Syst. 40(1): 5 (2015)
- Michael Benedikt, Pierre Bourhis, Clemens Ley: Querying Schemas With Access Restrictions. PVLDB 5(7): 634-645 (2012)
- Serge Abiteboul, Pierre Bourhis, Victor Vianu: Comparing workflow specification languages: A matter of views. ACM Trans. Database Syst. 37(2): 10 (2012)

International conferences

- Antoine Amarilli, Pierre Bourhis, Florent Capelli, Mikaël Monet: Ranked Enumeration for MSO on Trees via Knowledge Compilation. ICDT 2024: 25:1-25:18
- Pierre Bourhis, Aaron Boussidan, Philippe Gambette: On Distances Between Words with Parameters. CPM 2023: 6:1-6:23
- Camille Bourgaux, Pierre Bourhis, Liat Peterfreund, Michaël Thomazo: Revisiting Semiring Provenance for Datalog. KR 2022
- Pierre Bourhis, Alejandro Grez, Louis Jachiet, Cristian Riveros: Ranked Enumeration of MSO Logic on Words. ICDT 2021 20:1-20:19
- Pierre Bourhis, Daniel Deutch, Yuval Moskovitch: Equivalence-Invariant Algebraic Provenance for Hyperplane Update Queries. SIGMOD Conference 2020: 415-429
- Michael Benedikt, Pierre Bourhis, Louis Jachiet, Efthymia Tsamoura: Balancing Expressiveness and Inexpressiveness in View Design. KR 2020: 109-118
- Pierre Bourhis, Loïc Hélouët, Zoltán Miklós, Rituraj Singh: Data Centric Workflows for Crowdsourcing. Petri Nets 2020: 24-45
- Meghyn Bienvenu, Pierre Bourhis: Mixed-World Reasoning with Existential Rules under Active Domain Semantics ICJAI 2019
- Pierre Bourhis, Michel Leclère, Marie-Laure Mugnier, Sophie Tison, Federico Ulliana, Lily Gallois: Oblivious and Semi-Oblivious Boundedness for Existential Rules IJCAI 2019
- Michael Benedikt, Pierre Bourhis, Louis Jachiet, Michael Thomazo: Reasoning about disclosure in data integration in the presence of source constraints IJCAI 2019
- Antoine Amarilli, Pierre Bourhis, Stefan Mengel, Matthias Niewerth: Enumeration on Trees with Tractable Combined Complexity and Efficient Updates. PODS 2019
- Antoine Amarilli, Pierre Bourhis, Stefan Mengel and Matthias Niewerth Constant-Delay Enumeration for Nondeterministic Document Spanners ICDT 2019
- Serge Abiteboul, Pierre Bourhis, Victor Vianu: Explanations and Transparency in Collaborative Workflows. PODS 2018: 409-424
- Antoine Amarilli, Pierre Bourhis, Stefan Mengel:
- Antoine Amarilli, Pierre Bourhis, Stefan Mengel: Enumeration on Trees under Relabelings. ICDT 2018: 5:1-5:18

- Serge Abiteboul, Pierre Bourhis, Victor Vianu: Explanations and Transparency in Collaborative Workflows. PODS 2018: 409-424
- Michael Benedikt, Pierre Bourhis, Michael Vanden Boom: Characterizing Definability in Decidable Fixpoint Logics. (Best Paper Award Track B) ICALP 2017: 107:1-107:14
- Antoine Amarilli, Pierre Bourhis, Louis Jachiet, Stefan Mengel: A Circuit-Based Approach to Efficient Enumeration. ICALP 2017: 111:1-111:15
- Meghyn Bienvenu, Pierre Bourhis, Marie-Laure Mugnier, Sophie Tison, Federico Ulliana: Ontology-Mediated Query Answering for Key-Value Stores. IJCAI 2017: 844-851
- Pierre Bourhis, Michael Morak, Andreas Pieris: Making Cross Products and Guarded Ontology Languages Compatible. IJCAI 2017: 880-886
- Pierre Bourhis, Juan L. Reutter, Fernando Suárez, Domagoj Vrgoc: JSON: Data model, Query languages and Schema specification. PODS 2017: 123-135
- Antoine Amarilli, Pierre Bourhis, Mikaël Monet, and Pierre Senellart. Combined Tractability of Query Evaluation via Tree Automata and Cycluits. ICDT 2017
- Pierre Bourhis, Juan L. Reutter, Fernando Suarez and Domagoj Vrgoc, JSON: data model, query languages and schema specification PODS 2017

- Michael Benedikt, Pierre Bourhis, Gabriele Puppis and Balder Ten Cate. Querying Visible and Invisible Information. LICS 2016
- Michael Benedikt, Pierre Bourhis and Michael Vanden Boom. A step up in expressiveness of decidable fixpoint logics. LICS 2016
- Pierre Bourhis, Daniel Deutch, Yuval Moskovitch: Analyzing data-centric applications: Why, what-if, and how-to. ICDE 2016: 779-790
- Serge Abiteboul, Pierre Bourhis, Victor Vianu: A Formal Study of Collaborative Access Control in Distributed Datalog. ICDT 2016: 10:1-10:17
- Antoine Amarilli, Michael Benedikt, Pierre Bourhis, Michael Vanden Boom: Query Answering with Transitive and Linear-Ordered Data. IJCAI 2016: 893-899
- Pierre Bourhis, Carsten Lutz: Containment in Monadic Disjunctive Datalog, MMSNP, and Expressive Description Logics. KR 2016: 207-216
- Antoine Amarilli, Pierre Bourhis, Pierre Senellart: Tractable Lineages on Treelike Instances: Limits and Extensions. PODS 2016: 355-370
- Eleanor Ainy, Pierre Bourhis, Susan B. Davidson, Daniel Deutch, Tova Milo: Approximated Summarization of Data Provenance. CIKM 2015: 483-492
- Pierre Bourhis, Markus Krötzsch, Sebastian Rudolph: Reasonable Highly Expressive Query Languages - IJCAI-15 Distinguished Paper (Honorary Mention). IJCAI 2015: 2826-2832
- Antoine Amarilli, Pierre Bourhis, Pierre Senellart: Provenance Circuits for Trees and Treelike Instances. ICALP (2) 2015: 56-68
- Pierre Bourhis, Michael Morak, Andreas Pieris: Towards Efficient Reasoning Under Guarded-Based Disjunctive Existential Rules. MFCS (1) 2014: 99-110
- Pierre Bourhis, Gabriele Puppis, Cristian Riveros: Which DTDs are streaming bounded repairable? ICDT 2013: 57-68
- Serge Abiteboul, Pierre Bourhis, Anca Muscholl, Zhilin Wu: Recursive queries on trees and data trees. ICDT 2013: 93-104
- Vince Bárány, Michael Benedikt, Pierre Bourhis: Access patterns and integrity constraints revisited. ICDT 2013: 213-224
- Pierre Bourhis, Michael Morak, Andreas Pieris: The Impact of Disjunction on Query Answering

Under Guarded-Based Existential Rules. IJCAI 2013: 796-802
- Michael Benedikt, Pierre Bourhis, Pierre Senellart: Monadic Datalog Containment. ICALP (2) 2012: 79-91
- Serge Abiteboul, Pierre Bourhis, Victor Vianu: Highly expressive query languages for unordered data trees. ICDT 2012: 46-60
- Serge Abiteboul, Pierre Bourhis, Victor Vianu: Comparing workflow specification languages: a matter of views. ICDT 2011: 78-89
- Serge Abiteboul, Pierre Bourhis, Bogdan Marinoiu: Efficient maintenance techniques for views over active documents. EDBT 2009: 1076-1087
- Serge Abiteboul, Pierre Bourhis, Bogdan Marinoiu: Satisfiability and relevance for queries over active documents. PODS 2009: 87-96
- Serge Abiteboul, Pierre Bourhis, Alban Galland, Bogdan Marinoiu: The AXML Artifact Model. (invited paper) TIME 2009: 11-17

Demonstrations in international conferences

- Pierre Bourhis, Daniel Deutch and Yuval Moskovitch POLYTICS: Provenance-Based Analytics of Data-Centric Applications ICDE 2017
- Eleanor Ainy, Pierre Bourhis, Susan B. Davidson, Daniel Deutch, Tova Milo: PROX: Approximated Summarization of Data Provenance. EDBT 2016: 620-623
- Bogdan Marinoiu, Serge Abiteboul, Pierre Bourhis, Alban Galland: AXART - Enabling Collaborative Work with AXML Artifacts. PVLDB 3(2): 1553-1556 (2010)
- Serge Abiteboul, Bogdan Marinoiu, Pierre Bourhis: Distributed Monitoring of Peer-to-Peer Systems. ICDE 2008: 1572-1575