A Step Up in Expressiveness of Decidable Fixpoint Logics

Michael Benedikt

University of Oxford

Pierre Bourhis

CNRS CRIStAL, Université Lille 1,
INRIA Lille

Michael Vanden Boom

University of Oxford

Abstract

Guardedness restrictions are one of the principal means to obtain decidable logics — operators such as negation are restricted so that the free variables are contained in an atom. While guardedness has been applied fruitfully in the setting of first-order logic, the ability to add fixpoints while retaining decidability has been very limited. Here we show that one of the main restrictions imposed in the past can be lifted, getting a richer decidable logic by allowing fixpoints in which *the parameters of the fixpoint can be unguarded.* Using automata, we show that the resulting logics have a decidable satisfiability problem, and provide a fine study of the complexity of satisfiability. We show that similar methods apply to decide questions concerning the elimination of fixpoints within formulas of the logic.

Categories and Subject Descriptors Theory of computation [*Logic*]

1. Introduction

We are interested in expressive logics for which static analysis problems such as satisfiability are decidable. One way to achieve decidability is via *guardedness restrictions*. The Guarded Fragment (GF) [2] is a fragment of first-order logic obtained by requiring in existential quantification $\exists x.\phi(x)$ that ϕ be of the form $R(x) \wedge \phi'(x)$, where $R(x)$ is an atom containing all free variables of ϕ', and requiring in universal quantification that ϕ be of the form $R(x) \rightarrow \phi'(x)$, where R is as above. The Guarded Negation Fragment (GNF) [5] is an even more expressive decidable language, allowing unrestricted existential quantification but restricting negation to be of the form $R(x) \wedge \neg\phi'(x)$, with R as above.

Of course, there are other paradigms for decidability within first-order logic, such as restricting to two variables [27]. A particularly attractive feature of guarded logics is that, unlike with the two variable fragment [20], decidability can be extended to give decidable fragments of *least fixpoint logic* (LFP). LFP is the natural extension of first-order logic with a fixpoint constructor. Given a formula $\phi(x_1 \ldots x_m, y_1 \ldots y_n)$ 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 y, we can define a new m-ary relation: the relation is defined as the limit of a monotone sequence $X_0 \ldots$, starting with $X_0 = \emptyset$ and then setting

X_{i+1} to be the set of a such that ϕ holds when extending ρ with the interpretation of X as X_i and x as a. Emphasizing the distinction between x and y, we call x the *fixpoint variables* and 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) [19] extends GF with a fixpoint operator while maintaining decidability. The fixpoint constructor is restricted in two ways: the parameter variables y must be empty, and the fixpoint relation itself cannot be used as a guard. Guarded Negation Fixpoint Logic (GNFP) [5] adds fixpoint constructors to GNF with similar restrictions.

It is known that the second restriction on these fixpoint logics is essential for decidability [19]. But what about the first? It certainly seemed important to the proofs of decidability; [21] states

> 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 [17].

In this paper 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 [17]) 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 3).

The decidability of GFP is proven using an elegant high-level argument [18]: 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 [25]. 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 [19].

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.

Finally, we describe how our translation to automata can be modified to show decidability of the *boundedness problem*, a quantitative analog of satisfiability that asks if a fixpoint can be replaced by a fixed number of unfoldings. Our solution to the boundedness problem can be applied to decide whether certain GNFP-UP-expressible formulas can be rewritten in first-order logic.

2. Preliminaries

We consider signatures σ with a finite set of relations and constants. We write $\text{const}(\sigma)$ for the set of constants in the signature σ.

Least fixed point logic (LFP) over σ is the extension of first-order logic with the following formation rule: if $\phi(y, z, Y, Z)$ is a formula in which Y is a free second-order variable of arity $|y|$ that appears only positively, and t is a tuple of variables and constants of length $|y|$, then $[\mathbf{lfp}_{Y,y} . \phi](t)$ is also a formula. Informally, it asserts that t is in the least fixpoint induced by ϕ. The *parameter variables* of this formula are z — the free variables of ϕ other than y. In this paper, we emphasize the parameters by writing $[\mathbf{lfp}^z_{Y,y} . \phi]$, and reserve $[\mathbf{lfp}_{Y,y} . \phi]$ for the case when there are no parameters.

The semantics are standard: since $\phi(y, z, Y, Z)$ is positive in Y, it induces a monotone operator

$$U \mapsto O^{\mathfrak{A}, v, V}_\phi(U) := \{u : \mathfrak{A}, u, v, U, V \models \phi(y, z, Y, Z)\}$$

on every structure $\mathfrak{A}$ with valuation v for z and V for Z. Hence, this operator has a unique *least fixpoint*, denoted $\phi^\infty(\mathfrak{A}, v, V)$, and this can be obtained as the union of its *fixpoint approximants* $\phi^\beta(\mathfrak{A}, v, V)$ over all ordinals β, where

$$\phi^0(\mathfrak{A}, v, V) := \emptyset$$
$$\phi^{\beta+1}(\mathfrak{A}, v, V) := O^{\mathfrak{A}, v, V}_\phi(\phi^\beta(\mathfrak{A}, v, V))$$
$$\phi^\beta(\mathfrak{A}, v, V) := \bigcup_{\beta' < \beta} \phi^{\beta'}(\mathfrak{A}, v, V) \text{ where } \beta \text{ is a limit ordinal.}$$

The semantics of the least fixpoint formulas is defined such that $\mathfrak{A}, v, V \models [\mathbf{lfp}^z_{Y,y} . \phi](t)$ iff $t \in \phi^\infty(\mathfrak{A}, v, V) = \bigcup_{\beta \in \text{Ord}} \phi^\beta(\mathfrak{A}, v, V)$.

Free and bound variables The notion of free vs. bound second-order variables is standard. In particular, Y is free in $\phi(y, z, Y, Z)$ but bound in $[\mathbf{lfp}^z_{Y,y} . \phi](t)$. 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 z, then $\text{params}(Y) := z$, the *parameters associated with the second-order variable Y*.

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

The parameters in ϕ consists of the union of $\text{params}(Y)$ for all second-order variables Y occurring in ϕ; we let $\text{params}(\phi)$ 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[σ] *formula* ϕ is generated recursively from the following grammar:

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

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

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

A formula ϕ that includes free first-order variables x is x-*guarded* if it is logically equivalent to $\text{gdd}(x) \wedge \phi$. If $\text{free}(\phi) = x$ and ϕ is 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.

Simultaneous fixpoints We will also allow simultaneous fixpoints $[\mathbf{lfp}^z_{Y_i, y_i} .S](t)$ where

$$S = \begin{cases} Y_1, y_1 := \text{gdd}(y_1) \wedge \phi_1(y_1, z, Y_1, \ldots, Y_j, Z) \\ \quad \vdots \\ Y_j; y_j := \text{gdd}(y_j) \wedge \phi_j(y_j, z, Y_1, \ldots, Y_j, Z) \end{cases}$$

is a system of mutually defined equations $Y_i, y_i := \phi_i$ such that $\phi_i \in$ GNFP-UP, and $Y_1, \ldots, Y_j$ occur only positively in ϕ_i. Each ϕ_i utilizes the same parameters z, but different fixpoint variables y_i. Such a system defines a monotone operation on vectors of relations, and $[\mathbf{lfp}^z_{Y_i, y_i} .S](t)$ expresses that t is a tuple in the i-th component of the least fixpoint defined by this operation; Y_i is the distinguished *goal predicate* in $[\mathbf{lfp}^z_{Y_i, y_i} .S](t)$. Allowing simultaneous fixpoints does not change the expressivity of the logic, since they can be eliminated in favor of traditional fixpoints [3], with a possible exponential blow-up in size. However, it is often more convenient to work directly with these simultaneous fixpoints (see Example 4).

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[σ]*-formula:*

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

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: $\neg\exists xz.(\text{gdd}(x, z) \wedge \neg\phi(x, z)) \in$ 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 2. GNFP-UP *is strictly more expressive than* GNFP, *even over finite structures.*

Normal form A *conjunctive query* (CQ) is $\exists y.\psi$ 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[σ] *formula* ϕ or ψ is generated recursively from the following grammars:

$$\phi ::= \bigvee_i \exists y_i. \bigwedge_j \psi_{ij}$$
$$\psi ::= R\,t \mid Y\,t \mid \alpha \wedge \neg\phi \text{ where } \text{free}(\phi) \subseteq \text{free}(\alpha) \mid$$
$$[\mathbf{lfp}^z_{Y_m, y_m}.S](t)$$

where t is a tuple of variables or constants, Rt and α are atoms using a relation in σ or $=$, and S is a system with equations of the form $Y, y := \text{gdd}(y) \wedge \phi(y, z, Y, Z)$, 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 (GNFP-UP)^k[σ] for GNFP-UP formulas of width k.

Other guarded logics with parameters We could consider variants of other guarded logics with parameters. The *unary negation fixpoint logic with unguarded parameters* (UNFP-UP) is the fragment of GNFP-UP where only formulas with at most one free variable are negated, and the fixpoint predicates are monadic but may still carry any number of unguarded parameters. The formula $\phi(x, z)$ in Example 1 is in UNFP-UP.

We could also consider GFP-UP, the variant of GFP with parameters in the fixpoints, but where these parameters still need to be guarded if appearing under a quantification (this mimics the way we have added parameters to GNFP). The formula ϕ in Example 1 is not in GFP-UP since the quantification in $\exists y'.(Ryy' \wedge Yy')$ is not correctly guarded: GFP-UP would require something like $\exists y'.(Gzy' \wedge Yy')$, which includes a guard that covers not only y' but also the parameter z implicit in the fixpoint predicate Y. However, adding parameters to GFP in this way does not increase expressivity:

Proposition 3. *For every answer-guarded* GFP-UP *formula ϕ, we can construct in linear time an equivalent* GFP *formula ϕ' using fixpoint predicates of higher arity.*

This may explain why parameters were not considered further in [19, 21]: when they are introduced to GFP in their full power, it leads to undecidability, and when they are introduced in this more restrictive way, they do not add any expressive power. This is in contrast to GNFP, where we can add parameters in a way that strictly increases expressivity while still retaining decidability.

Organization In Section 3, we give some examples illustrating the expressive power of GNFP-UP. We also define the parameter depth, a key measure of how complicated a GNFP-UP formula is. We then argue in Section 4 that GNFP-UP has tree-like models, which makes satisfiability and boundedness amenable to tree automata techniques. The main technical work is described in Sections 5 and 6 where we describe the automata tools needed for GNFP-UP. Finally, in Sections 7 and 8, we use this automata machinery to study static analysis problems like satisfiability and boundedness.

3. Expressivity examples

In this section, 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.

In order 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 $\text{pdepth}_z(\eta)$ inductively as follows:

$$\text{pdepth}_z(R\,t) = \text{pdepth}_z(Y\,t) := 0$$
$$\text{pdepth}_z(\alpha \wedge \neg\phi) := \text{pdepth}_{z \cap \text{free}(\phi)}(\phi)$$
$$\text{pdepth}_z(\bigvee_i \exists x_i. \bigwedge_j \psi_{ij}) := \max_i p_i \text{ s.t.}$$
$$p_i := \begin{cases} 1 + \max_j \text{pdepth}_{\text{params}(\psi_{ij})}(\psi_{ij}) & \text{if } \exists j. \text{params}(\psi_{ij}) \nsubseteq z \\ \max_j \text{pdepth}_z(\psi_{ij}) & \text{otherwise} \end{cases}$$
$$\text{pdepth}_z([\mathbf{lfp}^{z'}_{X_m, x_m}.S](t)) :=$$
$$\begin{cases} 1 + \max_{\phi_j \in S} \text{pdepth}_{\text{params}(\phi_j)}(\phi_j) & \text{if } \exists j. \text{params}(\phi_j) \nsubseteq z \\ \max_{\phi_j \in S} \text{pdepth}_z(\phi_j) & \text{otherwise} \end{cases}.$$

The parameter depth $\text{pdepth}(\phi)$ for normal form $\phi \in$ GNFP-UP is just $\text{pdepth}_{\text{free}(\phi)}(\phi)$. 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[σ] with $\text{pdepth}(\phi) = 0$ and $\text{params}(\phi) = z$, then we can view ϕ as a GNFP formula without parameters, over the signature σ extended with extra constants 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.

We now give some examples to illustrate the expressivity of GNFP-UP, and this notion of parameter depth. These examples are drawn mostly from query languages used in databases and knowledge representation. Understanding these different logics and query languages is not important for understanding the main results about GNFP-UP (e.g., Theorem 20 and Theorem 24). However, for readers familiar with some of these previously-studied logics, they may give some insight into the sort of properties that can be expressed in GNFP-UP. Indeed, it is interesting to note that many

of the previously-studied logics described below are low in this parameter depth hierarchy.

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 a number of 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) [13] is just a CQ over such expressions.

Example 4. *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 $E'(x, y) := [\mathbf{lfp}_{X_0,x}^y \, . \, S](x)$ 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 $Rx_i z$ 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 be introducing variables that are used as parameters in the inner formulas.* GNFP-UP *can also express unions of* C2RPQ*s.*

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 labeled 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 [9]. 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

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

where the extensional predicates EDB$^{\Pi}$ and intensional predicates IDB$^{\Pi}$ are disjoint sets, Rules$^{\Pi}$ consists of formulas of the form $R(x_1 \ldots x_n) \leftarrow \psi(xy)$ where R is an IDB predicate and ψ is a conjunction of atoms, and goal is a distinguished member of IDB$^{\Pi}$. 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 $R(x_1 \ldots x_n) \leftarrow \psi(xy)$, the variables $x_1 \ldots x_n$ 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 [12, 26] 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 $\exists \mathbf{y}.\Pi$ where Π is a Monadic Datalog query, the goal predicate is nullary, and the rules use special symbols $\mathbf{z}$. 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 5 (based on [12])**.** *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*

$$U(y) \leftarrow R z_1 y \qquad U(y) \leftarrow U x \wedge R x y \qquad hit() \leftarrow U z_2 \, .$$

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 $\psi(z_1, z_2) := [\mathbf{lfp}_{hit,\emptyset}^{z_1, z_2} . S]()$ where*

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

Notice that the special symbols become parameters. Because hit is a nullary predicate, the fixpoint is nullary too. It expresses the same property as the formula in Example 1, but is written in an alternative way that mimics the MQ*. It has parameter depth 0.*

We can translate an arbitrary MQ $\exists \mathbf{y}.\Pi$ with special symbols $\mathbf{z}$ using a similar method: the monadic Datalog query becomes a simultaneous fixpoint ψ' in GNFP-UP, with the special symbols $\mathbf{z}$ as parameters, and the special nullary hit predicate as the goal predicate. The MQ itself can then be written in GNFP-UP as $\exists \mathbf{y}.\psi'(\mathbf{z})$. The resulting formula has pdepth at most 1, since $\psi'(\mathbf{z})$ has pdepth 0, and $\exists \mathbf{y}.\psi'(\mathbf{z})$ may project some of the parameters (the previous example only has pdepth 0 because there is no such projection).

In [26], 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. [12] 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.

Transitive closure logic The extension of FO with a transitive closure operator (rather than a full fixpoint operator) was introduced in [23] and has been studied extensively. In a similar vein, we can consider GNF extended with a *(guarded) transitive closure operator* which we denote GNF(TC). The idea is to add to GNF the following formula building rule: if $\phi(\boldsymbol{x}, \boldsymbol{y}) \in$ GNF(TC) for m-tuples $\boldsymbol{x}$ and $\boldsymbol{y}$, then $[\mathbf{TC}_{\boldsymbol{x},\boldsymbol{y}} \, . \, \mathrm{gdd}(\boldsymbol{x}) \wedge \phi(\boldsymbol{x}, \boldsymbol{y})](\boldsymbol{u}, \boldsymbol{v})$ is a formula in GNF(TC). The intended meaning is that $\boldsymbol{u}$ and $\boldsymbol{v}$ are guarded (in the original signature) and $(\boldsymbol{u}, \boldsymbol{v})$ is in the reflexive transitive closure of the binary relation on m-tuples defined by $\mathrm{gdd}(\boldsymbol{x}) \wedge \phi(\boldsymbol{x}, \boldsymbol{y})$.

Example 1 is $\exists x'.(Rxx' \wedge [\mathbf{TC}_{y,y'} \, . \, Ryy'](x', z))$ (the guardedness predicate is omitted since singletons are trivially guarded). CQPDL, and hence C2RPQs, can also be expressed in GNF(TC).

It is straightforward to check that every GNF(TC) formula can be translated in polynomial time to an equivalent GNFP-UP formula, and that such formulas can reach arbitrary pdepth levels.

***Why* GNFP-UP?** The previous examples serve to illustrate the variety of logics that GNFP-UP subsumes. GNFP-UP is useful because it serves as a unifying logic for all of these different formalisms that have some recursive nature.

A major advantage of GNFP-UP over the Datalog-based languages is that the logic has some form of negation. Not only does the presence of negation increase the expressivity of the queries that can be written in this language, but it also means that we can express directly query containment problems in this language, which was not possible in many of these earlier formalisms.

Despite the increased expressivity, we will show that GNFP-UP still has many useful model theoretic and computational properties, including decidable satisfiability and boundedness.

4. Tree-like models and tree encodings

Although GNFP-UP fails to have the finite model property (since it embeds the 2-way μ-calculus), it does have the *tree-like model property*. This says that if there is a model, then there is a model with a tree decomposition of some bounded tree width. A *tree decomposition* of a structure $\mathfrak{A}$ is a tree labelled with atomic facts of $\mathfrak{A}$ such that every atom is present in some label, and for each $a \in \mathrm{dom}(\mathfrak{A})$, the set of nodes (often called *bags*) that mention a form a connected part of the tree. The decomposition has *tree-width* $w-1$ if the number of elements represented in each bag is at most w.

Proposition 6. *Every satisfiable* (GNFP-UP)$^k[\sigma]$ *sentence has a model of tree-width at most* $k + |\mathrm{const}(\sigma)| - 1$.

The proof uses a standard technique, involving an unravelling based on a notion of guarded negation bisimulation.

The first route to deciding satisfiability relies on the tree-like model property of Proposition 6 along with the fact that GNFP-UP can be expressed in a fragment of second-order logic called *guarded second-order logic* (GSO) in which second-order quantification is interpreted only over guarded relations, i.e. relations where every tuple is guarded by some atom in the base signature.

Proposition 7. *Given* $\phi \in$ GNFP-UP$[\sigma]$*, we can construct an equivalent* $\phi' \in$ GSO$[\sigma]$ *in linear time.*

Corollary 8. *Satisfiability for* GNFP-UP *is decidable.*

Proof. GNFP-UP embeds in GSO by Proposition 7 and has bounded tree-width by Proposition 6. Using [21], GSO can be translated into an equivalent MSO formula (over encodings of the tree-like models) which is decidable by [25]. □

This is the easiest route to showing decidability of GNFP-UP, but it is not good for extracting complexity bounds. We next show how to make direct use of the tree-like model property and a translation taking a formula in the logic to tree automata that represent tree-like models of the formula, to determine more precisely the complexity of satisfiability and boundedness for GNFP-UP.

Coding structures Structures of bounded tree-width can be encoded as trees over a finite alphabet that depends only on the signature and the tree-width. Fix some signature σ and some width $k \in \mathbb{N}$. Let U_k be a set of size $2(k + |\mathrm{const}(\sigma)|)$. We refer to these as "names", as they name elements coded in a node. The signature $\tilde{\sigma}_k$ for the encodings is defined as follows.
- For all $a \in U_k$, there is a unary relation $D_a \in \tilde{\sigma}_k$ which indicates that a is a name for an element in the bag.
- For every relation $R \in \sigma$ of arity n and every n-tuple $\boldsymbol{a} \in U_k^n$, there is a unary relation $R_{\boldsymbol{a}} \in \tilde{\sigma}_k$, which indicates that R holds for the tuple of elements indexed by $\boldsymbol{a}$.
- For every constant $z \in \sigma$ and $c \in U_k$, there are unary relations $V_{c/z}$ which indicate that z is interpreted by the element named by c in the given bag.

Tree decompositions and the corresponding encodings can generally have unbounded (possibly infinite) degree. We apply the first-child, next-sibling transformation (based on an arbitrary ordering of the children) to the standard encoding, so that we can use binary trees for our encodings. This transformation takes a tree with arbitrary branching degree and constructs a binary tree as follows: it maps the root of the original tree to the root of a new binary tree; then, starting from the root, each node's leftmost child in the original tree is mapped to its left child in the binary tree, and its next sibling to the right in the original tree is mapped to its right child.

Hence, for our binary tree encodings, a node u can be identified by a word in $\{0, 1\}^*$, and the *biological children* of u are the nodes $u01^*$ (these are the nodes that would have been children of u in the tree decomposition before the first-child next-sibling transformation). The *biological parent* of $v \neq \epsilon$ is the unique u such that $v \in u01^*$. A *biological neighbor* is a biological child or biological parent. For these binary tree encodings, we also add to $\tilde{\sigma}_k$ unary predicates P_i for $i \in \{0, 1\}$ which indicate the node is the i-th child of its parent.

This transformation to a binary tree encoding is essential in certain automaton constructions (e.g. Theorem 13) when the automaton needs to record in its state the direction it came from. From now on, we use $\tilde{\sigma}_k$-tree to refer to an infinite full binary tree over $\tilde{\sigma}_k$.

Decoding structures If a $\tilde{\sigma}_k$-tree satisfies certain consistency properties, then it can be decoded into a σ-structure that has tree-width $k + |\mathrm{const}(\sigma)| - 1$.

Let $\mathrm{names}(v) := \{a \in U_k : D_a v\}$ denote the set of *names* used for elements in bag v.

A *consistent* $\tilde{\sigma}_k$-*tree* is a $\tilde{\sigma}_k$-tree such that every node v satisfies
- $|\mathrm{names}(v)| \leq k + |\mathrm{const}(\sigma)|$;
- for all $R_{\boldsymbol{a}} \in \tilde{\sigma}_k$, if $R_{\boldsymbol{a}} v$ then $\boldsymbol{a} \subseteq \mathrm{names}(v)$;
- $P_i v$ holds iff v is the i-th child of its parent;
- for all constants $z \in \sigma$, there is exactly one node w and one $c \in \mathrm{names}(w)$ such that $V_{c/z} w$.

Given a consistent tree $\mathcal{T}$, we say nodes u and v are *a-connected* if there is a sequence of nodes $u = w_0, w_1, \ldots, w_j = v$ such that w_{i+1} is a biological neighbor of w_i, and $a \in \mathrm{names}(w_i)$ for all $i \in \{0, \ldots, j\}$. We write $[v, a]$ for the equivalence class of a-connected nodes of v. For $\boldsymbol{a} = a_1 \ldots a_n$, we often abuse notation and write $[v, \boldsymbol{a}]$ for the tuple $[v, a_1], \ldots, [v, a_n]$

The *decoding* of $\mathcal{T}$ is the σ-structure $\mathfrak{D}(\mathcal{T})$ with universe $\{[v, a] : v \in \text{dom}(\mathcal{T}) \text{ and } a \in \text{names}(v)\}$ such that for each constant z, we have $z^{\mathfrak{D}(\mathcal{T})} := [v, c]$ for the unique v, c such that $V_{c/z} v$ holds, and for each relation R, we have $R^{\mathfrak{D}(\mathcal{T})}([v_1, a_1], \ldots, [v_j, a_j])$ iff there is $w \in \text{dom}(\mathcal{T})$ such that $R_{\boldsymbol{a}} w$ holds and $[w, a_i] = [v_i, a_i]$ for all i.

Free variables For formulas with free variables, the trees are extended with additional information about the valuations for these free variables. These trees use an extended signature where for each free first-order variable z and each $c \in U_k$, we introduce a predicate $V_{c/z}$, and for each second-order variable Z of arity n and each $\boldsymbol{a} \in U_k^n$, we introduce a predicate $Z_{\boldsymbol{a}}$. We refer to these as free variable markers or encodings of valuations for free variables.

We write $z^{\rightarrow}$ for the sets of predicates associated with z. For a set of free variables $z = \{z_1 \ldots z_n\}$, we write $z^{\rightarrow}$ for $z_1^{\rightarrow} \cup \cdots \cup z_n^{\rightarrow}$. We sometimes abuse notation and write $z^{\rightarrow}$ for both the predicates and the valuation of those predicates. Similar conventions apply to the free second-order variables.

Given a $\tilde{\sigma}_k$-tree $\mathcal{T}$, we write $(\mathcal{T}, z^{\rightarrow}, \mathbf{Z}^{\rightarrow})$ for a tree over the signature $\tilde{\sigma}_k \cup z^{\rightarrow} \cup \mathbf{Z}^{\rightarrow}$. We abuse the terminology and say $(\mathcal{T}, z^{\rightarrow}, \mathbf{Z}^{\rightarrow})$ is a consistent $\tilde{\sigma}_k$-tree if $\mathcal{T}$ is a consistent $\tilde{\sigma}_k$-tree, and for each $z \in z^{\rightarrow}$, there is exactly one v and one $c \in \text{names}(v)$ such that $V_{c/z} v$ holds, and for each $Z_{\boldsymbol{a}} \in \mathbf{Z}^{\rightarrow}$, if $Z_{\boldsymbol{a}} v$ then $\boldsymbol{a} \subseteq \text{names}(v)$.

5. Automata tools

We make use of automata running on infinite binary trees. In this section, we briefly recall some definitions and properties (please consult, e.g., [24, 29] for more information). We will need to use 2-way automata that can move both up and down as they process the tree, so we highlight some less familiar properties about the relationship between 2-way and 1-way versions of these automata.

Trees The input structures are infinite full binary trees $\mathcal{T}$ over a finite set of propositions Σ. In other words, these are structures over a signature with binary relations for the left and right child relation, and unary relations for the propositions. We also assume there are propositions indicating whether each node is a left child, right child, or the root. We write $\mathcal{T}(v)$ for the set of propositions that hold at node v.

Tree automata An *alternating parity tree automaton* $\mathcal{A}$ is a tuple $\langle \Sigma, Q, q_0, \delta, \Omega \rangle$ where Σ is a finite set of propositions, Q is a finite set of states, $q_0 \in Q$ is the initial state, $\delta : Q \times \mathcal{P}(\Sigma) \to \mathcal{B}^+(\text{Dir} \times Q)$ is the transition function with directions $\text{Dir} \subseteq \{\mathbf{l}, \mathbf{r}, \uparrow\}$, and $\Omega : Q \to P$ is the priority function with a finite set of *priorities* $P \subseteq \mathbb{N}$.

The transition function maps a state and input letter to a positive boolean formula over propositions $\text{Dir} \times Q$, denoted $\mathcal{B}^+(\text{Dir} \times Q)$. This formula indicates possible next moves for the automaton. We can assume that these formulas are written in disjunctive normal form. Running the automaton $\mathcal{A}$ on some input tree $\mathcal{T}$ is best thought of in terms of an *acceptance game*. Positions in the game are of the form $(q, v) \in Q \times \text{dom}(\mathcal{T})$. In position (q, v), Eve chooses a disjunct θ in $\delta(q, \mathcal{T}(v))$. Then Adam chooses a conjunct (d, q') in θ and the game continues from position (q', v'), where v' is the node in direction d from v. In other words, the disjunct θ chosen by Eve specifies the possible copies of the automaton that could be launched by Adam from v or a neighbor of v.

A play $(q_0, v_0)(q_1, v_1) \ldots$ in the game is winning for Eve if it satisfies the *parity condition*: the maximum priority occurring infinitely often in $\Omega(q_0)\Omega(q_1) \ldots$ is even. A *strategy* for Eve is a function that given the history of the play and the current position in the game, determines Eve's choice in the game. Note that we allow the automaton to be started from arbitrary positions v_0 in the tree, rather than just the root; we will often indicate this by saying that the automaton is *launched* from v_0. We say that $\mathcal{A}$ *accepts* $\mathcal{T}$ *starting from* v_0 if Eve has a strategy such that all plays consistent

with the strategy starting from (q_0, v_0) are winning. $L(\mathcal{A})$ denotes the *language* of trees accepted by $\mathcal{A}$ starting from the root.

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 (omitting the standard proofs).

First, the automata that we are using are closed under union and intersection (of their languages).

Proposition 9. *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 PTime 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 10. *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 11. *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 [30] that 2-way alternating parity tree automata can be converted to equivalent 1-way nondeterministic automata, with an exponential blow-up.

Theorem 12 ([30]). *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., [10, 12]).

Theorem 13. *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 $v \in \text{dom}(\mathcal{T})$,

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

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 $\mathcal{A}'$.

Proof sketch. $\mathcal{A}$ simulates $\mathcal{A}'$ by guessing in a backwards fashion an initial part of a run of $\mathcal{A}'$ on the path from v to the root and then processing the rest of the tree in a normal downwards fashion. The subtlety is that the automaton $\mathcal{A}$ is reading a tree without valuations for $P_1, \ldots, P_j$ so once the automaton leaves node v, if it were to cross this position again, it would be unable to correctly simulate $\mathcal{A}'$. To avoid this, we only send downwards copies of the automaton in directions that are not on the path from the root to v. □

We remark that this construction can be adapted for an alternating parity automaton as input, but $\mathcal{A}$ is exponential in the size of the input automaton $\mathcal{A}'$, rather than linear.

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

Theorem 14 ([16],[30]). *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.*

6. Automata for GNFP-UP

In this section, 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 t is in the least fixpoint $[\mathbf{lfp}_{Y,y}^z \,. \phi]$ 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 y being tested in the fixpoint, with the initial position being t. In general, in position y, one round of the game consists of the following:

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

Adam wins if the game continues forever.

The idea is that if 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. In fact, we start with *localized versions* of this automaton because we need to launch different versions based on Adam's challenges. A *local assignment* b/y for $b = b_1 \ldots b_n \in U_k^n$ and $y = y_1 \ldots y_n$ is a mapping such that $y_i \mapsto b_i$. A node v in $\mathcal{T}$ with $b \subseteq \text{names}(v)$ and a local assignment b/y, specifies a valuation for $y^\rightarrow$. We say it is local since the free variable markers for y would all appear locally in v. If we have an automaton $\mathcal{A}$ running on trees with free variable markers for y, we say that we *localize $\mathcal{A}$ to b/y* if we apply the localization theorem (Theorem 13) 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 y correspond to the elements $[v, b]$, but it no longer relies on free variable markers for 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 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 $y' \in Y$, 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 y' (again, we know that 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 in a similar way.

In order 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 12 and 13, 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 $\theta \in$ GNFP-UP$[\sigma]$. First, it is straightforward to construct an automaton that checks consistency:

Proposition 15. *There is a 2-way alternating parity tree automaton C that checks whether or not a $\tilde{\sigma}_k$-tree (possibly extended with additional free variable markers for z and Z) is consistent. The size of C is at most exponential in $(|\sigma| + |z| + |Z|) \cdot |U_k|^k$.*

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 actually 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 16. *For normal form $\theta \in$ (GNFP-UP)$^k[\sigma]$ with $\text{pdepth}(\theta) \geq 1$, we can construct a 2-way alternating parity tree automaton $\mathcal{A}_\theta$ such that for all consistent $\tilde{\sigma}_k$-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 $(pdepth(\theta)+1)$-exponential in $|\theta|$.

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

For brevity, in this theorem and in the remainder of the paper, we usually give only bounds on the output size, not the running time of the algorithms. However the proofs will show that the worst-case running time is bounded by a polynomial in the output size, i.e. the running time of Theorem 16 is $(pdepth(\theta) + 1)$-ExpTime. We emphasize that this means that *for fixed pdepth, the construction can be done in elementary time.*

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 16 is by induction on $|\theta|$, and constructs localized 2-way automata for subformulas of θ. For GNFP subformulas, it is known from [8] how to construct 2-way automata:[1]

Lemma 17 ([8]). *Let $\psi(\boldsymbol{y}, \boldsymbol{Z}) \in GNFP^k[\sigma']$ 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 $w \in dom(\mathcal{T})$ with $\boldsymbol{b} \subseteq names(w)$,*

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

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 $m = |\psi|$, $l = |const(\sigma')|$, and $r = rank_{CQ}(\psi)$. 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 $\psi \in (GNFP\text{-}UP)^k[\sigma]$ with $params(\psi) \subseteq \boldsymbol{z}$, define the *augmented signature* $\sigma_{\boldsymbol{z},\psi}$ to be the signature σ together with additional constants $\boldsymbol{z} \in \boldsymbol{z}$ and subformula predicates F_η for subformulas η with $params(\eta) \subseteq \boldsymbol{z}$. For such η, the arity of F_η is usually $|free(\eta) \setminus params(\eta)|$; 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 $\eta \in (GNFP\text{-}UP)^k[\sigma]$ with $free(\eta) \subseteq \boldsymbol{yz}$ and $params(\eta) \subseteq \boldsymbol{z}$. We define $transform_{\boldsymbol{z}}(\eta) \in GNFP^k[\sigma_{\boldsymbol{z},\eta}]$ inductively as follows:

$transform_{\boldsymbol{z}}(R\,\boldsymbol{t}) := R\,\boldsymbol{t} \qquad transform_{\boldsymbol{z}}(Y\,\boldsymbol{t}) := Y\,\boldsymbol{t}$

$transform_{\boldsymbol{z}}(\alpha \wedge \neg\phi) := \alpha \wedge \neg transform_{\boldsymbol{z} \cap free(\phi)}(\phi)$

$transform_{\boldsymbol{z}}([\mathbf{lfp}_{X,\boldsymbol{x}}^{\boldsymbol{z}'}.S](\boldsymbol{t})) :=$

$\begin{cases} F_{[\mathbf{lfp}_{X,\boldsymbol{x}}^{\boldsymbol{z}'}.S](\boldsymbol{t})}\,\boldsymbol{t} & \text{if there is } \phi_j \in S \text{ with } params(\phi_j) \not\subseteq \boldsymbol{z} \\ [\mathbf{lfp}_{X,\boldsymbol{x}}.S'](\boldsymbol{t}) & \text{o.w.} \end{cases}$

where S' is the result of applying $transform_{\boldsymbol{z}}$ to each $\phi_j \in S$

$transform_{\boldsymbol{z}}(\bigvee_i \exists \boldsymbol{x}_i. \bigwedge_j \psi_{ij}) :=$

$\begin{cases} F_{\bigvee_i \exists \boldsymbol{x}_i. \bigwedge_j \psi_{ij}}\,\boldsymbol{y} & \text{if there is } i, j \text{ such that } params(\psi_{ij}) \not\subseteq \boldsymbol{z} \\ \bigvee_i \exists \boldsymbol{x}_i. \bigwedge_j transform_{\boldsymbol{z} \cap free(\psi_{ij})}(\psi_{ij}) & \text{o.w.} \end{cases}$

---

[1] [8] used a different encoding of the tree-like models, but the adaptation to the encoding here requires only minor technical changes.

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.

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 $free(\eta') \setminus params(\eta')$. 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 in the construction, described in Lemma 18 below, 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.

Lemma 18. *Let $\phi(\boldsymbol{y}, \boldsymbol{z}, \boldsymbol{Z})$ be a subformula of $\theta \in (GNFP\text{-}UP)^k[\sigma]$ with $params(\phi) \subseteq \boldsymbol{z}$. For each local assignment $\boldsymbol{b}/\boldsymbol{y}$, we can construct a 2-way alternating parity tree automaton $\mathcal{B}_\phi^{\boldsymbol{b}/\boldsymbol{y}}$ such that for all consistent $\tilde{\sigma}_k$-trees $(\mathcal{T}, \boldsymbol{z}^\rightarrow, \boldsymbol{Z}^\rightarrow)$ and for all nodes $w \in dom(\mathcal{T})$ with $\boldsymbol{b} \subseteq names(w)$,*

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

For $pdepth_{\boldsymbol{z}}(\phi) \geq 1$, there is a polynomial function f independent of ϕ such that the size of all such localized automata is at most $exp_f^{pdepth_{\boldsymbol{z}}(\phi)}(f(mn) \cdot 2^{f(klr)})$ where $m = |\phi|$, $n = |\sigma|$, $l = |const(\sigma)|$, and $r = rank_{CQ}(\phi)$. The number of priorities is linear in $|\phi|$. For $pdepth_{\boldsymbol{z}}(\phi) = 0$, the bounds match Lemma 17.

Proof sketch. The proof is by induction on $|\phi|$.

Assume $\phi' := transform_{\boldsymbol{z}}(\phi)$ is helpful. This always holds for the smallest (atomic) formulas, so this covers the base of the induction. We construct $\mathcal{B}_\phi^{\boldsymbol{b}/\boldsymbol{y}}$ to simulate the automaton $\mathcal{A}_{\phi'}^{\boldsymbol{b}/\boldsymbol{y}}$ from Lemma 17, while allowing Eve to guess valuations for the F_η relations from ϕ'. Since ϕ' is helpful, we know that every F_η relation in ϕ' is for some formula η that is strictly smaller than ϕ, and hence the inductive hypothesis ensures there is a corresponding automaton for every suitable local assignment. During the simulation of $\mathcal{A}_{\phi'}^{\boldsymbol{b}/\boldsymbol{y}}$, if Eve asserts $F_{\eta(\boldsymbol{x})}\,\boldsymbol{a}$ at w for some $\boldsymbol{a} \subseteq names(w)$, then Adam can challenge this by launching the localized automaton for the intersection of $\mathcal{B}_{\eta(\boldsymbol{x})}^{\boldsymbol{a}/\boldsymbol{x}}$ and $\mathcal{B}_{gdd(\boldsymbol{x})}^{\boldsymbol{a}/\boldsymbol{x}}$ from w; likewise, if Eve does not assert $F_{\eta(\boldsymbol{x})}\,\boldsymbol{a}$ at w for some $\boldsymbol{a} \subseteq names(w)$, then Adam can challenge this by launching the localized automaton for the dual of the intersection of $\mathcal{B}_{\eta(\boldsymbol{x})}^{\boldsymbol{a}/\boldsymbol{x}}$ and $\mathcal{B}_{gdd(\boldsymbol{x})}^{\boldsymbol{a}/\boldsymbol{x}}$ from w. Correctness follows from Lemma 17, and the fact that the inductive hypothesis ensures the subautomata for F_η in ϕ' are correct. There is no exponential blow-up in this case.

Next, assume that $\phi' := transform_{\boldsymbol{z}}(\phi)$ is unhelpful. There are two possible cases.

The first case is when ϕ is a UCQ-shaped formula $\bigvee_i \exists \boldsymbol{x}_i. \bigwedge_j \psi_{ij}$ where variables from some $\boldsymbol{x}_i$ are used as parameters in $\bigwedge_j \psi_{ij}$. To start, we consider each CQ-shaped formula separately, so fix some $\exists \boldsymbol{x}_i. \bigwedge_j \psi_{ij}$. We use the inductive hypothesis to obtain 2-way alternating automata for each conjunct ψ_{ij}, using the empty local assignment. This is possible since the size of these conjuncts must be strictly less than the size of ϕ. Because we are using the empty local assignment, these automata operate on trees with markers for all of the free variables: $\boldsymbol{x}_i \cup \boldsymbol{y} \cup \boldsymbol{z}$. Take the intersection of these automata using Proposition 9. Then take the intersection with the automaton from Proposition 15 checking consistency. This yields a 2-way alternating automaton corresponding to $\bigwedge_j \psi_{ij}$. We then convert this

automaton to a nondeterministic version using Theorem 12 and project away the information about x_i using Proposition 10. This yields an equivalent (1-way) nondeterministic parity automaton for the CQ. Next, we localize this automaton to the desired variables y using Theorem 13. Finally, to construct the automaton $\mathcal{B}_\phi^{b/y}$, we take the union of the individual CQ automata using Proposition 9. The conversion from a 2-way alternating automaton to a nondeterministic automaton is the costly step in this process, resulting in an exponential blow-up. This matches the claimed size bound since $pdepth(\psi_{ij}) < pdepth_z(\phi)$.

The second case is when ϕ is a fixpoint formula where fixpoint variables are used as parameters in the body of the fixpoint. Suppose it is of the form $[\mathbf{lfp}_{X,x}^{z'} . gdd(x) \wedge \chi(xz', XZ)](t)$ where $params(\chi) \cap x \neq \emptyset$, so $params(\chi) \not\subseteq z$; the construction is similar for a simultaneous fixpoint. The formula χ in the body of the fixpoint is strictly smaller, so we can apply the inductive hypothesis to get an automaton for this part. We want this automaton to be localized to x so we can test for tuples in the fixpoint by launching copies of the automaton for some local assignment. However, the inductive hypothesis does not directly yield this, since some variables from x are used as parameters. Hence, we must use the inductive hypothesis to get a 2-way automaton for $\mathcal{B}_{\chi'}^{0/0}$, apply Theorem 12 to get an equivalent 1-way nondeterministic automaton (resulting in an exponential blow-up), and then localize to some a/x using Theorem 13. Once we have these localized automata $\mathcal{B}_\chi^{a/x}$ for the body of the fixpoint, we can construct an automaton that captures the fixpoint game described at the beginning of this section. $\square$

Theorem 16 easily follows from this lemma.

Using exactly the same conversion rules from [5], it can be shown that an arbitrary GNFP-UP sentence θ' can be converted to an equivalent normal form θ with size exponential in $|\theta|$, but width and CQ-rank linear in $|\theta|$. Hence, we have the following corollary.

Corollary 19. *For $\theta' \in$ GNFP-UP$[\sigma]$ (not necessarily in normal form), we can construct an automaton $\mathcal{A}_{\theta'}$ of $(pdepth(\theta') + 1)$-exponential size.*

7. Satisfiability and containment

Because of the tree-like model property for GNFP-UP, we can use the automaton construction and ExpTime emptiness testing (Theorem 14) to decide satisfiability of GNFP-UP.

Theorem 20. *Satisfiability for $\theta \in$ GNFP-UP is decidable in $(pdepth(\theta) + 2)$-ExpTime.*

Applications We can apply Theorem 20 to obtain results about the query languages described in Section 3: e.g., containment of m-nested MQs is in $(m + 2)$-ExpTime. This result was known already from [12]. However, an advantage of this GNFP-UP framework is that we can introduce additional features such as relativizing the results to sentences in our logics, without affecting the complexity.

Corollary 21. *Containment of m-nested MQs relative to boolean frontier-guarded Datalog queries (or any boolean queries translatable to GNFP in PTime) is decidable in $(m + 2)$-ExpTime.*

We can extend our approach to even allow some unguarded logics on the left-hand side of the containment (as is done in [12], which considers only unrelativized containment).

Corollary 22. *Containment of a Datalog query in an m-nested MQ is in $(m + 2)$-ExpTime, even relative to a boolean frontier-guarded Datalog query or a GNFP sentence.*

We can also derive results about satisfiability of CQPDL with respect to GNFP sentences. Although CQPDL can reach arbitrary pdepth levels, we can construct an equi-satisfiable formula of low pdepth. This yields the following new result, extending results about (nested) C2RPQs from [9].

Corollary 23. *Satisfiability of CQPDL sentences (or boolean combinations of CQPDL sentences and GNFP sentences) is decidable in 2-ExpTime.*

Lower bounds This connection with flag and check queries also demonstrates that our general $(m + 2)$-ExpTime bound on satisfiability for GNFP-UP formulas of pdepth m is optimal, since containment of boolean Datalog queries in m-nested MQs is actually $(m + 2)$-ExpTime hard [12].

8. Extending to boundedness

Thus far, we have concentrated on showing that satisfiability is decidable for GNFP-UP, using techniques based on automata. In this section, we point out that we can extend this automata machinery to help answer additional questions like boundedness.

The *boundedness problem* for a logic $\mathcal{L}$ over σ asks:

Given a formula $\phi(x, X) \in \mathcal{L}[\sigma]$ positive in some second-order variable X of arity $|x|$, is there a natural number n such that for all σ-structures $\mathfrak{A}$, $\phi^n(\mathfrak{A}) = \phi^{n+1}(\mathfrak{A})$?

In other words, the boundedness problem asks whether there is a uniform natural number bound on the number of iterations needed to reach the least fixpoint induced by $\phi(x, X)$. For formulas $\phi(x, z, X)$ with some designated set of parameters z, it is also natural to ask the following variant of the boundedness problem: is there a natural number n such that for all σ-structures $\mathfrak{A}$ and for all valuations $z \mapsto c$, does $\phi^n(\mathfrak{A}, c) = \phi^{n+1}(\mathfrak{A}, c)$?

There is a large body of work studying boundedness for various logics, particularly Datalog queries (see [1, 15, 22]). Boundedness is undecidable when ϕ is a negation-free first-order formula. However, for many guarded logics, boundedness has been shown to be decidable [6, 8, 11] (some of the decidability results rely on unpublished work due to Colcombet, referred to as ILT in [6]).

Using similar techniques, we can analyze boundedness for GNFP-UP. As was the case for our analysis of satisfiability, the key is to take advantage of the fact that we can restrict to structures of bounded tree-width, and then use tree automata to help solve the problem. For boundedness a variant of the prior construction (with the same bounds) can generate a special type of tree automaton with counters called a *cost automaton* (see [8, 14] for more information). This cost automaton defines a function that maps a consistent tree $\mathcal{T}$ to the least $n \in \mathbb{N} \cup \{\infty\}$ such that every tuple in $\phi^\infty(\mathfrak{D}(\mathcal{T}))$ is in $\phi^n(\mathfrak{D}(\mathcal{T}))$ — there is a single counter that is incremented each time the fixpoint is unfolded. The range of the function defined by this cost automaton is bounded by a natural number across all consistent trees iff ϕ is bounded.

In general, it is not known how to decide if the range of the function defined by a cost automaton over infinite trees is bounded. However, for special types of cost automata — like the automata that come from analyzing boundedness for guarded logics — this is known to be decidable [8] (in cases where there is a reliance on cost automaton results that have been claimed before but not published, we mark this with ILT as in [6]).[2] Hence, by combining our automaton construction with results from [8] and ILT, we can show:

Theorem 24. *Assuming ILT, the boundedness problem is decidable for x-guarded $\phi(x, X) \in$ GNFP-UP$[\sigma]$ in non-elementary time (elementary time for fixed pdepth).*

In the special case of x-guarded $\phi(x, z, X) \in$ GNF$[\sigma]$, boundedness is decidable in elementary time (without assuming ILT).

---

[2] ILT stands for "infinite limitedness theorem", a statement about cost automata on infinite trees. More details can be found in [8].

FO *definability* For certain logics, boundedness coincides with FO definability; e.g., we have the following corollary of Theorem 24.

Corollary 25. *It is decidable whether* $[\mathbf{lfp}_{X,\boldsymbol{x}}^z.\phi]$ *can be written in* FO *for* $\boldsymbol{x}$*-guarded* $\phi(\boldsymbol{x}, \boldsymbol{z}, X) \in \mathrm{GNF}[\sigma]$.

Furthermore, whenever this holds the fixpoint $[\mathbf{lfp}_{X,\boldsymbol{x}}^z.\phi]$ *can in fact be written in* GNF.

Proof. If $\phi \in$ GNF is bounded, then the fixpoint is equivalent to the n-th approximant, for some $n \in \mathbb{N}$. We can write out the formula ϕ^n for this approximant, where $\phi^0 := \bot$, and $\phi^n := \phi[\phi^{n-1}(\boldsymbol{y})/X\boldsymbol{y}]$. This is in GNF, and witnesses the FO definability of $[\mathbf{lfp}_{X,\boldsymbol{x}}^z.\phi]$. The other direction follows from the Barwise-Moschovakis theorem [7]. □

A similar result is not known for $\phi \in \mathrm{GNFP\text{-}UP}[\sigma]$, because of the presence of additional fixpoints in ϕ — the boundedness problem only concerns the elimination of the outermost fixpoint.

As another application, we can combine automata techniques from this paper with other cost automata results in [8, 10] to prove the following result about negation-free CQPDL and FO definability.

Theorem 26. *It is decidable whether or not a negation-free* CQPDL *sentence relative to a* GNF *sentence can be expressed in* FO.

In particular, we can decide whether a conjunctive regular path query can be expressed in FO*, and similarly for their nested and two-way variants (e.g.* C2RPQ*s).*

The ability to reduce C2RPQ querying to first-order querying is interesting, since when this occurs we can use standard database techniques to evaluate graph queries with recursion.

9. Conclusion

We have explained how fixpoint logics can be increased in expressivity while retaining decidability, by allowing *unguarded parameters*. We have also undertaken a fine-grained analysis of the complexity of static analysis problems for the resulting logics.

A limitation of our analysis is that it restricts to reasoning over *all* structures, both finite and infinite. In contrast, [4, 28] have shown that unparameterized guarded fixpoint logics have decidable satisfiability problems over finite structures. It is not clear if the technique of [4] extends to deal with unguarded parameters.

The results about testing first-order definability of fixpoint logics and recursive queries (e.g. Theorem 26) do not include complexity bounds. We conjecture that the cost automaton results could be analyzed and refined further (as was done in [8]) to extract at least an elementary bound (and, ideally, a 2-ExpTime bound). For cases without negation, like FO definability of C2RPQs, it may well be possible to extend the more elementary automata-theoretic approach used to decide boundedness for monadic Datalog [15], avoiding the use of cost automata altogether.

Acknowledgments

References

[1] S. Abiteboul. Boundedness is undecidable for datalog programs with a single recursive rule. *IPL*, 32(6):281–287, 1989.

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

[3] A. Arnold and D. Niwiński. *Rudiments of mu-calculus*. North Holland, 2001.

[4] V. Bárány and M. Bojańczyk. Finite satisfiability for guarded fixpoint logic. *IPL*, 112(10):371–375, 2012.

[5] V. Bárány, B. ten Cate, and L. Segoufin. Guarded negation. In *ICALP*, 2011.

[6] V. Bárány, B. ten Cate, and M. Otto. Queries with guarded negation. *PVLDB*, 5(11):1328–1339, 2012.

[7] J. Barwise and Y. N. Moschovakis. Global inductive definability. *JSL*, 43(3):521–534, 1978.

[8] M. Benedikt, B. ten Cate, T. Colcombet, and M. Vanden Boom. The complexity of boundedness for guarded logics. In *LICS*, 2015.

[9] M. Bienvenu, D. Calvanese, M. Ortiz, and M. Simkus. Nested regular path queries in description logics. In *KR*, 2014.

[10] A. Blumensath, T. Colcombet, D. Kuperberg, P. Parys, and M. Vanden Boom. Two-way cost automata and cost logics over infinite trees. In *CSL-LICS*, 2014.

[11] A. Blumensath, M. Otto, and M. Weyer. Decidability results for the boundedness problem. *LMCS*, 10(3), 2014.

[12] P. Bourhis, M. Krötzsch, and S. Rudolph. Reasonable highly expressive query languages. In *IJCAI*, 2015.

[13] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Containment of conjunctive regular path queries with inverse. In *KR*, 2000.

[14] T. Colcombet and C. Löding. Regular cost functions over finite trees. In *LICS*, 2010.

[15] S. S. Cosmadakis, H. Gaifman, P. C. Kanellakis, and M. Y. Vardi. Decidable optimization problems for database logic programs. In *STOC*, 1988.

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

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

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

[19] E. Grädel and I. Walukiewicz. Guarded fixed point logic. In *LICS*, 1999.

[20] E. Grädel, M. Otto, and E. Rosen. Undecidability results on two-variable logics. In *STACS*, 1997.

[21] E. Grädel, C. Hirsch, and M. Otto. Back and forth between guarded and modal logics. *ACM TOCL*, 3(3):418–463, 2002.

[22] G. G. Hillebrand, P. C. Kanellakis, H. G. Mairson, and M. Y. Vardi. Undecidable boundedness problems for datalog programs. *J. Log. Program.*, 25(2):163–190, 1995.

[23] N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68(1-3):86–104, 1986.

[24] C. Löding. Automata on infinite trees. Available at http://www.automata.rwth-aachen.de/ loeding/inf-tree-automata.pdf.

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

[26] S. Rudolph and M. Krötzsch. Flag & check: data access with monadically defined queries. In *PODS*, 2013.

[27] D. Scott. A decision method for validity of sentences in two variables. *JSL*, 27(477), 1962.

[28] L. Segoufin and B. ten Cate. Unary negation. *LMCS*, 9(3), 2013.

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

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