Exploiting Forwardness: Satisfiability and Query-Entailment in Forward Guarded Fragment

Bartosz Bednarczyk[1,2(✉)]

[1] Computational Logic Group, Technische Universität Dresden, Dresden, Germany
[2] Institute of Computer Science, University of Wrocław, Wrocław, Poland
bartosz.bednarczyk@cs.uni.wroc.pl

Abstract. We study the complexity of two standard reasoning problems for Forward Guarded Logic ($\mathcal{FGF}$), obtained as a restriction of the Guarded Fragment in which variables appear in atoms only in the order of their quantification. We show that $\mathcal{FGF}$ enjoys the higher-arity-forest-model property, which results in ExpTime-completeness of its (finite and unrestricted) knowledge-base satisfiability problem. Moreover, we show that $\mathcal{FGF}$ is well-suited for knowledge representation. By employing a generalisation of Lutz's spoiler technique, we prove that the conjunctive query entailment problem for $\mathcal{FGF}$ remains in ExpTime.

We find that our results are quite unusual as $\mathcal{FGF}$ is, up to our knowledge, the first decidable fragment of First-Order Logic, extending standard description logics like $\mathcal{ALC}$, that offers unboundedly many variables and higher-arity relations while keeping its complexity surprisingly low.

1 Introduction

The *guarded fragment of first-order logic* ($\mathcal{GF}$) is a prominent fragment of first-order logic ($\mathcal{FO}$) that finds application in ontology-based reasoning and in database theory [4,6,24]. In particular, $\mathcal{GF}$ embeds standard modal logics (like K) as well as description logics (DLs) *e.g.* $\mathcal{ALC}$ [8]. The guarded fragment is obtained from $\mathcal{FO}$ by requiring that first-order quantification is appropriately relativised by atoms. It was introduced by Andréka, Németi and van Benthem [1] who proved that its satisfiability problem is decidable. A year later, Grädel [9] proved that $\mathcal{GF}$ has the finite model property and is 2ExpTime-complete. In this work we study the complexity of a certain fragment of $\mathcal{GF}$.

1.1 Our Motivation and Related Work

Our motivation is two-fold. The first comes from applications of $\mathcal{GF}$ to databases and description logics, where query entailment under ontologies plays a vital role. In this scenario a relational database $\mathcal{D}$ and a set of constraints $\mathcal{T}$ (a.k.a. ontology) are given as an input. The input database may not satisfy the given constraints and hence, we look at possible ways of expanding it in a way so that

W. Faber et al. (Eds.): JELIA 2021, LNAI 12678, pp. 1–15, 2021.
https://doi.org/10.1007/978-3-030-75775-5_13

the axioms of $\mathcal{T}$ are finally fulfilled. We are interested in the question whether a query q has a certain answer in the (expanded) database. It boils down to the problem of checking if all models of $(\mathcal{D}, \mathcal{T})$ entail q. Such a question is obviously undecidable in general [3] and the ongoing works concentrate on identifying relevant fragments of $\mathcal{FO}$ for which the problem is decidable [4] and has manageable complexity.

The second motivation is complexity-theoretic. Since the complexity of the Guarded Fragment is relatively high, it is natural to ask whether there exists a fragment of $\mathcal{GF}$ having reasonable complexity while still being expressive enough to capture description logics like $\mathcal{ALC}$. A few such restrictions have already been proposed. Grädel [9] has shown that the complexity of $\mathcal{GF}$ can be lowered to ExpTime either by bounding the number of variables, or the arity of relational symbols. This however, does not seem to be well-suited for applications in database theory, as databases may have arbitrarily large schemas. We would prefer a solution leading to lower complexity that does not restrict the number of variables or the arity of relations. Moreover, Grädel's restriction does not help to lower the complexity of the query entailment problem: his logic captures the DL $\mathcal{ALCI}$, known to have 2ExpTime-hard query entailment problem [18]. Another idea was recently suggested by Kieroński [15]. In [15] the author proposed a family of one-dimensional guarded logics that restrict quantification patterns in $\mathcal{GF}$ by leaving each maximal block of quantifiers in it with at most one free variable. Their satisfiability problem is NExpTime-complete (so probably lower than 2ExpTime) but the complexity of the query entailment problem is still 2ExpTime-hard. The culprit is again the ability to speak about inverses of relations, giving us a way to capture $\mathcal{ALCI}$.

1.2 Our Results

In this work we present a sublogic of $\mathcal{GF}$ that overcomes the problems mentioned in the previous section, which is inspired by Fluted Logic [22,23]. We call our logic the *Forward Guarded Fragment* ($\mathcal{FGF}$) of First-Order Logic. $\mathcal{FGF}$ restricts quantification patterns of $\mathcal{GF}$ in such a way that tuples of variables appearing in atoms are infixes of the sequence of the already quantified-variables (in the order of their quantification). This "forwardness" prohibits the logic from capturing the inverse relations from $\mathcal{ALCI}$ but it still is expressive enough to capture $\mathcal{ALC}$. Moreover, the logic offers a non-trivial use of higher-arity relations, so it can be employed to reason about real-life relational databases.

In the paper we exploit "forwardness" to show that $\mathcal{FGF}$-knowledge-bases enjoy the *higher-arity-forest-model property*, a tailored version of the forest-model property from $\mathcal{GF}$ in which the higher-arity relations link elements from different levels of a tree only in a contiguous ascending order. This property is then employed to establish ExpTime-completeness for the knowledge-base satisfiability problem, which also relies on the fact that there are only exponentially many different relevant types of tuples of the domain elements. The culmination point of the paper is the ExpTime-completeness proof of the CQ entailment

problem, achieved by a generalisation of Lutz's spoiler technique from [19], carefully tailored towards higher-arity relations.

Our proof techniques are similar to those introduced in [9,19]. However, the devil is in the details and higher-arity relations made the problem significantly more difficult. Missing proofs were delegated to the technical report.

2 Preliminaries

In this paper, we employ the standard terminology from finite model theory [17]. Usually, we refer to structures with fraktur letters, and to their universes with the corresponding Roman letters. When working with structures, we always assume that they have non-empty domains. We employ countable *signatures* of individual constants $\mathbf{N_I}$ and predicates (of various positive arities) Σ. The arity of $\mathrm{R} \in \Sigma$ is denoted with $\mathsf{ar}(\mathrm{R})$. We refer to domain elements with $\mathrm{c}, \mathrm{d}, \mathrm{e}, \ldots$ and usually employ $\vec{\mathrm{c}}, \vec{\mathrm{d}}, \vec{\mathrm{e}}, \ldots$ to denote tuples of domain elements. We frequently use variables $x, y, \ldots$ from a countably-infinite set $\mathbf{N_V}$ and individual names $\mathsf{a}, \mathsf{b}, \ldots$ from $\mathbf{N_I}$. We write $\varphi(\vec{x})$ to indicate that all free variables of φ are in $\vec{x}$. A sentence is a formula without free variables. For a unary function f we write $f(\vec{x})$ to denote the tuple resulting from applying f to each element of $\vec{x}$. Given a structure $\mathfrak{A}$ and a set $B \subseteq A$ we define the *restriction* of $\mathfrak{A}$ to B as the structure $\mathfrak{A}\restriction_B$.

Let $\mathcal{L}$ be a fragment of $\mathcal{FO}$ with its standard syntax and semantics. Given φ with free variables in $\vec{x}$ we say that a tuple of domain elements $\vec{\mathrm{d}}$ from $\mathfrak{A}$ *satisfies* $\varphi(\vec{x})$ iff $\mathfrak{A} \models \varphi[\vec{x}/\vec{\mathrm{d}}]$ holds. An $\mathcal{L}$-theory $\mathcal{T}$ is a finite set of $\mathcal{L}$-formulae over Σ. An $\mathcal{L}$-database is a finite set of facts, *i.e.* expressions of the form $\mathrm{R}(\vec{\mathsf{a}})$, where $\vec{\mathsf{a}}$ is a tuple of individual names. We denote the set of individual names appearing in $\mathcal{D}$ with $\mathsf{ind}(\mathcal{D})$. An $\mathcal{L}$-knowledge-base (a kb for short) is a pair $\mathcal{K} = (\mathcal{D}, \mathcal{T})$ composed of $\mathcal{L}$-database $\mathcal{D}$ and $\mathcal{L}$-theory $\mathcal{T}$. We say that a structure $\mathfrak{A}$ satisfies a theory $\mathcal{T}$ (written: $\mathfrak{A} \models \mathcal{T}$) if it satisfies all of its formulae. Similarly, $\mathfrak{A}$ satisfies a database $\mathcal{D}$ if it satisfies all its facts (with individual names treated as constants). We say that $\mathfrak{A}$ satisfies a kb $\mathcal{K}$ (written: $\mathfrak{A} \models \mathcal{K}$) if it satisfies both its components.

In the *satisfiability* (resp. *knowledge base satisfiability*) problem for a logic $\mathcal{L}$ we ask whether an input formula (resp. knowledge-base) from $\mathcal{L}$ has a *model*.

2.1 Queries

Conjunctive queries (CQs) are conjunctions of positive atoms with variables from $\mathbf{N_V}$. The set of variables appearing in q is denoted with $\mathrm{Var}(q)$ and the number of atoms of q (*i.e.* the size of q) is denoted with $|q|$. The fact that $\mathrm{R}(\vec{x})$ appears in q is indicated with $\mathrm{R}(\vec{x}) \in q$. Whenever some subset $V \subseteq \mathrm{Var}(q)$ is given, with $q\restriction_V$ we denote a sub-query of q where all the atoms containing any variable outside V are removed.

Let $\pi : \mathrm{Var}(q) \to A$ be a *variable assignment*. We write $\mathfrak{A} \models_\pi \mathrm{R}(\vec{x})$ if $\pi(\vec{x}) \in \mathrm{R}^{\mathfrak{A}}$. Similarly, we write $\mathfrak{A} \models_\pi q_1 \wedge q_2$ iff $\mathfrak{A} \models_\pi q_1$ and $\mathfrak{A} \models_\pi q_2$, for some CQs

q_1, q_2. We say that π is a *match* for $\mathfrak{A}$ and q if $\mathfrak{A} \models_\pi q$ holds and that $\mathfrak{A}$ *satisfies* q (denoted with: $\mathfrak{A} \models q$) whenever $\mathfrak{A} \models_\pi q$ for some match π. The definitions are lifted to kbs: q is *entailed* by a kb $\mathcal{K}$ (written: $\mathcal{K} \models q$) if all models $\mathfrak{A}$ of $\mathcal{K}$ satisfy q. When $\mathfrak{A} \models \mathcal{K}$ but $\mathfrak{A} \not\models q$, we call $\mathfrak{A}$ a *countermodel* for $\mathcal{K}$ and q. Note that q is entailed by $\mathcal{K}$ iff there are no countermodels for $\mathcal{K}$ and q. In the *CQ entailment problem* for a logic $\mathcal{L}$ we ask if an input $\mathcal{L}$-kb $\mathcal{K}$ entails an input CQ q.

Observe that a conjunctive query q can be seen as a structure $\mathfrak{H}_q$, with the domain $\mathrm{Var}(q)$, having the interpretation of relations fixed as $\mathrm{R}^{\mathfrak{H}_q} = \{\, \vec{x} \mid \mathrm{R}(\vec{x}) \in q \,\}$. We will call it a *query hypergraph* of q. Hence, any match π for $\mathfrak{A}$ and q can be seen as a homomorphism from $\mathfrak{H}_q$ to $\mathfrak{A}$.

3 Forward Guarded Fragment

We introduce the Forward Guarded Fragment (denoted with $\mathcal{FGF}$) of First-Order Logic defined as the intersection of the Guarded Fragment [1] and the Forward Fragment, sharing the spirit of the Fluted Fragment [23]. We define their syntax first. We stress that the considered logics do not allow for constants and equality.

3.1 Logics

Recall that the *guarded fragment* ($\mathcal{GF}$) is obtained from $\mathcal{FO}$ by requiring that first-order quantification is appropriately relativised by atoms. Formally $\mathcal{GF}$ is the smallest set containing all atomic formulae, closed under boolean connectives and whenever $\varphi(\vec{x}, \vec{y})$ is in $\mathcal{GF}$ and $\alpha(\vec{x}, \vec{y})$ is an atom containing all free variables of φ then both $\forall \vec{y}\ (\alpha(\vec{x}, \vec{y}) \to \varphi(\vec{x}, \vec{y}))$ and $\exists \vec{y}\ (\alpha(\vec{x}, \vec{y}) \wedge \varphi(\vec{x}, \vec{y}))$ are in $\mathcal{GF}$. The atom α is called a *guard*.

Next we define the *forward fragment* ($\mathcal{FF}$) of $\mathcal{FO}$. It is inspired by the Fluted Fragment $\mathcal{FL}$ [23] and the Ordered Fragment of $\mathcal{FO}$ [11]: the main difference is that we allow the variable sequences appearing in formulae to be infixes of the already quantified variables, not only suffixes (as in $\mathcal{FL}$) or prefixes (as in the ordered fragment). Turing our attention to the formal definition of $\mathcal{FF}$, let us fix a sequence $\vec{x_\omega} = x_1, x_2, \ldots$ of variables from $\mathbf{N_V}$. For simplicity, we write $\vec{x}_{i\ldots j}$ to denote the (gap-free!) sequence $x_i, x_{i+1}, \ldots, x_j$. We start by defining the set of $\mathcal{FF}^{[n]}$ formulae over Σ for all natural n:

- an atom $\alpha(\vec{x})$ belongs to $\mathcal{FF}^{[n]}$ if $\vec{x} = \vec{x}_{k\ldots\ell}$ for some infix $[k, \ell]$ of $[1, n]$
- $\mathcal{FF}^{[n]}$ is closed under boolean connectives $\wedge, \vee, \neg, \to$;
- If $\varphi(\vec{x}_{1\ldots n+1})$ is in $\mathcal{FF}^{[n+1]}$ then both $\exists x_{n+1}\ \varphi(\vec{x}_{1\ldots n+1})$ and $\forall x_{n+1}\ \varphi(\vec{x}_{1\ldots n+1})$ belong to $\mathcal{FF}^{[n]}$.

We define $\mathcal{FF}$ as the set $\mathcal{FF}^{[0]}$, which is composed exclusively of sentences. We stress that $\mathcal{FF}$ was not studied in the literature before but it can be polynomially reduced to the Fluted Fragment $\mathcal{FL}$.

Finally, we define the *forward guarded fragment* ($\mathcal{FGF}$) as $\mathcal{GF} \cap \mathcal{FF}$, thus combining both mentioned restrictions. To gain more intuitions on $\mathcal{FGF}$, we encourage the reader to consult the following correct $\mathcal{FGF}$ formula φ_1^{ok} as well as three incorrect formulae $\varphi_{1\text{–}3}^{bad}$:

$$\varphi_1^{ok} = \forall x_1\ \mathrm{A}(x_1) \to \exists x_2\ \big[\mathrm{S}(x_1,x_2) \wedge \neg\mathrm{U}(x_1,x_2) \wedge \neg\mathrm{A}(x_2) \wedge \forall x_3 \forall x_4\ (\mathrm{T}(x_1,x_2,x_3,x_4) \to \mathrm{P}(x_2,x_3,x_4) \wedge \mathrm{A}(x_4))\big]$$

$$\varphi_1^{bad} = \forall x_1\ \mathrm{R}(x_1,x_1), \quad \varphi_2^{bad} = \forall x_1 \forall x_2\ \mathrm{S}(x_1,x_2) \to \mathrm{R}(x_2,x_1),$$
$$\varphi_3^{bad} = \forall x_1 \forall x_2 \forall x_3\ \mathrm{R}(x_1,x_2) \wedge \mathrm{R}(x_2,x_3) \to \mathrm{R}(x_1,x_3)$$

Note that all of the aforementioned incorrect formulae are not in $\mathcal{FGF}$ due to the fact that sequences of variables appearing in atoms are not infixes of $x_1, \ldots, x_k$, with k being the number of the last quantified variable. One can also observe that there is another reason for the third formula to be incorrect: the quantifiers in φ_3^{bad} are not guarded, *i.e.* the atom $\alpha(x_1,x_2,x_3)$ after the last quantifier is missing. The atom $\mathrm{S}(x_1,x_2)$ in φ_2^{bad} is an example of a correct guard. The formula φ_1^{bad} demonstrates why the equality predicate is disallowed in $\mathcal{FGF}$.

3.2 Simplified Forms and Forward Types

While working with $\mathcal{FGF}$ formulae it is convenient to convert them into an appropriate normal form. The proof goes via a routine renaming.

Lemma 1. *For any $\mathcal{FGF}$-kb $\mathcal{K} = (\mathcal{D}, \mathcal{T})$ we can compute (in polynomial time) an equi-satisfiable kb $\mathcal{K}_{simpl} = (\mathcal{D}_+, \{\varphi_\forall, \varphi_{\forall\exists}\})$ (over an extended signature) with*

$$\varphi_\forall = \bigwedge_{i=0}^{m_\forall} \forall \vec{x}_{1\ldots k_i}\ \mathrm{R}_{\forall_i}(\vec{x}_{1\ldots k_i}) \to \psi_{\forall_i}(\vec{x}_{1\ldots k_i})$$

$$\varphi_{\forall\exists} = \bigwedge_{i=0}^{m_{\forall\exists}} \forall \vec{x}_{1\ldots k_i}\ \mathrm{R}_{\forall\exists_i}(\vec{x}_{1\ldots k_i}) \to \exists \vec{x}_{k_i+1\ldots k_i+\ell_i}\ \mathrm{S}_{\forall\exists_i}(\vec{x}_{1\ldots k_i+\ell_i}) \wedge \psi_{\forall\exists_i}(\vec{x}_{1\ldots k_i+\ell_i}),$$

where (possibly decorated) R, S and ψ denote, respectively, predicates and quantifier-free $\mathcal{FGF}$ formulae. We refer to such a $\mathcal{K}_{simpl}$ as a simplified $\mathcal{K}$.

We next introduce a notion of a *forward type* useful to reason about $\mathcal{FGF}$-definable properties. Fix finite signature Σ and positive n. A (Σ,n)*-forward type* is an $\mathcal{FO}$ formula with n free-variables $\vec{x}_{1\ldots n}$ s.t. for all symbols $\mathrm{R} \in \Sigma$ of arity ℓ not bigger than n and for all $1 \leq i \leq n{+}1{-}\ell$ a type contains as a conjunct either $\mathrm{R}(\vec{x}_{i\ldots i+\ell-1})$ or its negation. We write $\mathsf{tp}^{\Sigma}_{\mathfrak{A}}(\vec{\mathrm{d}})$ to denote the *unique* forward type satisfying $\mathfrak{A} \models \mathsf{tp}^{\Sigma}_{\mathfrak{A}}(\vec{\mathrm{d}})$. We also say that $\vec{\mathrm{d}}$ *realises* the forward type $\mathsf{tp}^{\Sigma}_{\mathfrak{A}}(\vec{x})$. By elementary counting we can see that the number of (Σ,n)-forward types is exponential in $|\Sigma|{+}n$ while their sizes are only polynomial.

Lemma 2. *Up to isomorphism, there are at most $2^{|\Sigma|\cdot n^2}$ (Σ,n)-forward types. Moreover, each (Σ,n)-forward type has at most $|\Sigma| \cdot n$ conjuncts.*

Finally, by unfolding definitions, one can show that whenever two tuples have equal forward types then they satisfy the same formulae from simplified kbs.

3.3 Higher-Arity-Forest-(Counter)Model Property

Here we introduce the notion of higher-arity forests, which are forest reflecting the essence of forwardness. We say that a structure $\mathfrak{F}$ is a *higher-arity forest* (HAF) if its domain is a prefix-closed subset of sequences from $\mathbb{N}^+$ and for all relational symbols R of arity k we have that $\vec{\mathrm{d}} \in \mathrm{R}^{\mathfrak{F}}$ implies:

- either all the elements from $\vec{\mathrm{d}}$ are natural numbers (= one-element sequences)
- or $\vec{\mathrm{d}} = (\mathrm{c}_1, \ldots, \mathrm{c}_\ell, \mathrm{e}_1, \mathrm{e}_2, \ldots \mathrm{e}_{\ell'})$, where each member of $\vec{\mathrm{c}}$ is a number and there exist numbers $n_1, n_2, \ldots, n_{\ell'}$ such that $\mathrm{e}_i = \mathrm{c}_\ell \cdot n_1 \cdot \ldots \cdot n_i$ for all $\ell' \geq i \geq 0$
- or $\vec{\mathrm{d}} = (\mathrm{d}_1, \ldots, \mathrm{d}_k)$, with $\mathrm{d}_1 \notin \mathbb{N}$, such that for each index i there exist a number n_i such that $\mathrm{d}_{i+1} = \mathrm{d}_i \cdot n_i$.

The elements from $F \cap \mathbb{N}$ are simply the *roots* of $\mathfrak{F}$. A forest with a single root is called a *tree*. We also use the prefix ordering $\prec_{\mathsf{pref}}$ to speak about children, parents, siblings in the usual (graph-theoretic) way. Observe that, intuitively, higher-arity forests are just forests in which relations either arbitrarily traverse roots or connect other elements but only in a level-by-level ascending order.

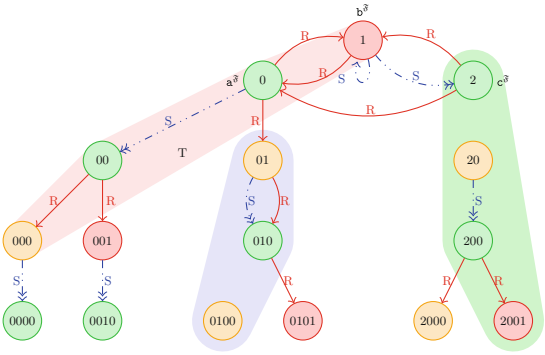


Fig. 1. An example higher-arity forest. The coloured areas in the picture indicates higher-arity relations, *e.g.* the red area means $\mathrm{T}(1, 0, 00, 000)$. (Color figure online)

A model $\mathfrak{A}$ of a kb $\mathcal{K} = (\mathcal{D}, \mathcal{T})$ is a *HAF model* iff $\mathfrak{A}$ is a HAF with the set of roots being equal to the set of interpretations of individuals from $\mathsf{ind}(\mathcal{D})$ in $\mathfrak{A}$.

We show $\mathcal{FGF}$ enjoys the *HAF-model property*, useful to design an ExpTime decision procedure for deciding $\mathcal{FGF}$. In the proof we take any model $\mathfrak{A}$ of $\mathcal{K}$ and construct an infinite sequence of forest of growing sizes. The first of them is simply $\mathfrak{A}$ restricted to the interpretation of database constants. The others are obtained as follows: whenever some forest $\mathfrak{F}$ contains a tuple $\vec{\mathrm{d}}$ of elements does not have a witness to satisfy a conjunct of $\varphi_{\forall\exists}$ we expand the domain of $\mathfrak{F}$ with a fresh copy of its original witnesses taken from $\mathfrak{A}$ and connect it to $\vec{\mathrm{d}}$, mimicking the connections in $\mathfrak{A}$. The limit of this process will be a HAF-model of $\mathcal{K}$.

Lemma 3. *Any satisfiable simplified $\mathcal{FGF}$ kb $\mathcal{K}$ has a HAF model. Moreover, if there is a countermodel for $\mathcal{K}$ and a CQ q then there is also a HAF countermodel.*

3.4 ExpTime-Completeness of the kb Satisfiability Problem

The notion of forward types and higher-arity forests will now be employed to design an alternating PSpace procedure for deciding the satisfiability for $\mathcal{FGF}$ knowledge bases. Since APSpace = ExpTime [5] we derive an ExpTime upper bound for $\mathcal{FGF}$. The matching lower bound is inherited from $\mathcal{ALC}$ [2]. The forthcoming algorithm is a variant of Grädel's algorithm for $\mathcal{GF}$ [9].

We sketch the main ideas. As a preliminary step, we first transform the input $\mathcal{K}$ into $\mathcal{K}_{smpl} = (\mathcal{D}_+, \{\varphi_\forall, \varphi_{\forall\exists}\})$. Then the rest of the procedure is responsible for constructing a higher-arity forest-model $\mathfrak{F}$ of $\mathcal{K}_{smpl}$. We start from guessing the "roots" $\mathfrak{R}$ of $\mathfrak{F}$. Note that we cannot simply guess $\mathfrak{R}$: once Σ contains an n-ary predicate, such a predicate might be composed of $|R|^n$ different tuples and thus we cannot fully store it in polynomial space. Fortunately we do not need to do it. It turns out that for the feasibility of our procedure it suffices to keep only the forward types of tuples appearing in $\mathcal{D}_+$ (the number of which is bounded polynomially, see: Lemma 2). Since the guessed structure is of polynomial size, we can perform the standard $\mathcal{FO}$ model-checking algorithm [25] to ensure that $\mathfrak{R}$ satisfies both $\mathcal{D}_+$ and $\varphi_\forall$. It could be, however, that $\varphi_{\forall\exists}$ is not satisfied (yet). We then iterate over all conjuncts λ from $\varphi_{\forall\exists}$, universally choosing a tuple $\vec{\mathrm{d}}$ of elements for whose the antecedent of λ is satisfied but the consequent of λ is not. For such a tuple we introduce fresh elements $\vec{\mathrm{e}}$ and guess the forward type of $\vec{\mathrm{d}} \cdot \vec{\mathrm{e}}$. Next, we check that $\vec{\mathrm{d}} \cdot \vec{\mathrm{e}}$ indeed satisfies λ and whether its type does not violate $\varphi_\forall$ (we reject otherwise). Finally, we recursively repeat the procedure for the substructure containing only $\vec{\mathrm{d}} \cdot \vec{\mathrm{e}}$. The procedure accepts when the number of steps exceeds the total number of (Σ, n)-forward-types – by pigeonhole principle it follows that one of the (Σ, n)-forward-types necessarily occurs twice, so if the procedure has not rejected the input yet it means that we can safely repeat the process over and over, making exactly the same choices as it did before.

Our pseudo-code and its correctness proof are available in the full paper. From it we conclude the first main theorem of the paper. Since $\mathcal{GF}$ has the finite model property [9] (even in the presence of constants that can simulate DBs) our algorithm for $\mathcal{FGF}$ can also be applied to the finite-model reasoning.

Theorem 4. *Kb (finite) satisfiability problem for $\mathcal{FGF}$ is* ExpTime*-complete.*

4 Query Answering

This section provides a worst-case complexity-optimal algorithm for deciding query entailment over $\mathcal{FGF}$ knowledge-bases. The main technique employed here is a generalisation of the so-called *spoiler technique* by Lutz [19, Sec. 3], carefully tailored to work over structures having relations of arity greater than 2.

We first give a rather informal explanation of the technique. We recall that to decide $\mathcal{K} \models q$ it suffices to check the existence of a HAF countermodel for $\mathcal{K}$ and q (see: Lemma 3). In the ideal situation, we would know how to prepare a knowledge-base $\mathcal{K}_{\neg q}$ that characterises the class of all HAF countermodels for q. Note that the existence of $\mathcal{K}_{\neg q}$ would immediately imply that any model of

$\mathcal{K} \cup \mathcal{K}_{\neg q}$ is, by definition, also a countermodel for $\mathcal{K}$ and q. The problematic part is, of course, the construction of $\mathcal{K}_{\neg q}$. To decide satisfiability of $\mathcal{K} \cup \mathcal{K}_{\neg q}$ we would like axioms of $\mathcal{K}_{\neg q}$ to be written in $\mathcal{FGF}$, which seems to be challenging since the matches of q may have arbitrary complex shapes. On the positive side, there is a simple way of detecting matches of tree-shaped queries, based on the well-known *rolling-up technique* [13, Sec. 4]: we basically describe tree-shaped matches as unary predicates by defining their trees in a bottom-up manner and then we enforce their emptiness in all models of $\mathcal{K}_{\neg q}$. Here we exploit the fact that countermodels can be made HAFs and combine the rolling-up technique with so-called *splittings*, that detects query matches of arbitrary shape over forests. In order to block such matches, we parallelise the construction of $\mathcal{K}_{\neg q}$. Rather than construing one huge kb we divide it into smaller chunks $\mathcal{K}_s$ called *spoilers* with an intuitive meaning that the consistency of any of $\mathcal{K} \cup \mathcal{K}_s$ *spoils* the entailment $\mathcal{K} \models q$. Once we show that each spoiler is of polynomial size and there are only exponentially many of them, we can reduce the entailment question to exponentially many satisfiability checks for kbs of polynomial size (hence in ExpTime by Theorem 4), deducing the ExpTime-completeness of CQ entailment problem for $\mathcal{FGF}$.

4.1 Rolling-Up: Detecting Matches of Tree-Shaped Queries

We consider a modification of the *rolling-up technique* that transforms tree-shaped queries into $\mathcal{FGF}$. In our scenario, the name "tree-shaped" indicates that the underlying hypergraph $\mathfrak{H}_q$ of a query q is a (connected) higher-arity tree. Henceforth we always assume that whenever $\mathrm{R}(\vec{x}_{1\ldots k}) \in q$ then also $\mathrm{R}_i(\vec{x}_{1\ldots i}) \in q$ for fresh relation names R_i. We call such CQs *closed* and by the *closure* of q, denoted with $\mathsf{cl}(q)$, we mean the query obtained from q by extending q in a minimal way to make it closed. Note that the entailment problem of CQs and closed CQs over $\mathcal{FGF}$ kbs coincides, since we can always extend the input kb with fresh relations R_i and the rules $\forall \vec{x}_{1\ldots \mathsf{ar}(\mathrm{R})}\ \mathrm{R}(\vec{x}_{1\ldots \mathsf{ar}(\mathrm{R})}) \leftrightarrow \bigwedge_{i=1}^{\mathsf{ar}(\mathrm{R})} \mathrm{R}_i(\vec{x}_{1\ldots i})$ for all non-unary predicates R appearing in q. Abusing slightly the notation, we call the kbs extended in the above way their *q-closures*.

In what follows we are going to construct, for every variable $v \in \mathrm{Var}(q)$, a unary predicate $\mathrm{Subt}_q^v(x)$ with the indented meaning that $\mathrm{d} \in (\mathrm{Subt}_q^v)^{\mathfrak{A}}$ holds whenever the subtree of $\mathfrak{H}_q$ rooted at the variable v can be mapped below d in $\mathfrak{A}$. In order to adjust the rolling-up technique to non-binary relations that may appear in trees, we employ additional non-binary predicates $\mathrm{Subt}_q^{\vec{v},u}(\vec{x}, y)$ that do the same job as $\mathrm{Subt}_q^u(y)$ but in contrast they memorise the path $\vec{v}$ leading to u, so the higher-arity relations can be retrieved from the construction.

An inductive definition is given next. The main idea behind it is to traverse the input tree in a bottom-up manner, describing its shape in $\mathcal{FGF}$, and gradually "rolling-up" the input tree into smaller chunks until its root is reached.

Definition 5. *For a given closed tree-shaped CQ q and any sequence of variables $\vec{v}u$ from* $\mathrm{Var}(q)$ *(that follows the level-by-level order in $\mathfrak{H}_q$) we define an*

($|\vec{v}|+1$)-ary predicate $\mathrm{Subt}_q^{\vec{v},u}(\vec{x}_{1\dots|\vec{v}|+1})$ *as follows. The empty conjunction is treated as* $\top$*.*

1. *We initially set* $\mathrm{Subt}_q^{\vec{v},u}(\vec{x}_{1\dots|\vec{v}|+1})$ *to be equal:*

$$\bigwedge_{\mathrm{R}(\vec{v}_{k\dots|\vec{v}|}\,u)\in q} \mathrm{R}(\vec{x}_{k\dots|\vec{v}|+1}) \wedge \bigwedge_{\mathrm{A}(u)\in q} \mathrm{A}(x_{|\vec{v}|+1})$$

2. *Additionally, when* u *is not a leaf of* $\mathfrak{H}_q$*, we supplement the above formula with some extra conjuncts for each children variable* $w \in \mathrm{Var}(q)$ *of* u *in* $\mathfrak{H}_q$*. Take a longest suffix* $\vec{v}_{\mathsf{suff}}$ *of* $\vec{v}$ *for which* $\mathrm{R}(\vec{v}_{\mathsf{suff}}, u, w) \in q$ *(if there is no such suffix then keep* $\vec{v}_{\mathsf{suff}}$ *empty) and append the formula:*

$$\exists x_{|\vec{v}|+2}\ \mathrm{Subt}_q^{\vec{v}_{\mathsf{suff}},u,w}(\vec{x}_{|\vec{v}|-|\vec{v}_{\mathsf{suff}}|+1\dots|\vec{v}|+2})$$

We use $\mathrm{Match}_q(x)$ as a shorthand for $\mathrm{Subt}_q^{x_r}(x)$ with x_r being the root of $\mathfrak{H}_q$. We stress that due to the closedness of q and the fact that we keep the variables appropriately ordered, the definition of $\mathrm{Match}_q(x)$ is in $\mathcal{FGF}$.

From the presented construction we can easily see that the size (*i.e.* the number of atoms) of Match_q is polynomial in $|q|$. The next lemma, claiming correctness of the presented definition, can be shown by induction.

Lemma 6. *For any higher-arity forest* $\mathfrak{A}$ *and a closed tree-shaped conjunctive query* q *we have* $(\mathrm{Match}_q)^{\mathfrak{A}} \neq \emptyset$ *iff there exists a homomorphism* $\mathfrak{h} : \mathfrak{H}_q \to \mathfrak{A}$*.*

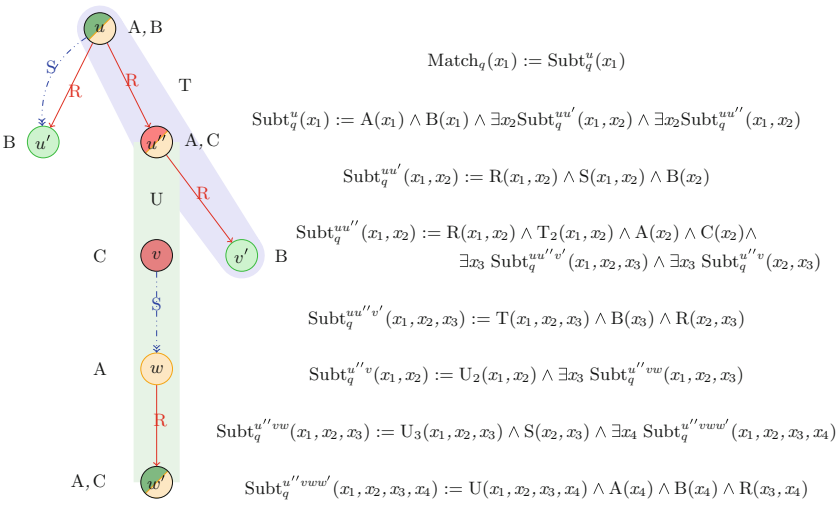


Fig. 2. An example CQ q together with the resulting rolling-up predicates. In the picture we omitted additional relations appearing in q due to its closedness. Moreover, in the definitions of predicates Subt_q we omitted $\mathrm{S}_1, \mathrm{R}_1, \mathrm{T}_1, \mathrm{U}_1$.

The presented rolling-up technique shows us how to detect matches of tree-shaped queries. Its direct consequence is the forthcoming theorem telling us that such query matches can be effectively blocked and giving us a robust reduction from query entailment problem for tree-shaped queries to kb satisfiability problem.

Theorem 7. *Let $\mathcal{K} = (\mathcal{D}, \mathcal{T})$ be a closed satisfiable kb and let q be a closed tree-shaped CQ. Then $\mathcal{K} \not\models q$ iff the kb $\mathcal{K} \cup \{\forall x_1 \neg \mathrm{Match}_q(x_1)\}$ is satisfiable.*

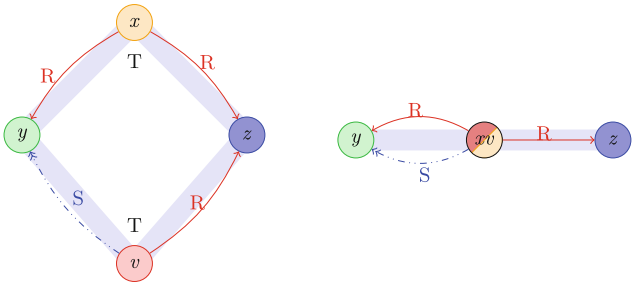
Unfortunately, the above theorem does not transfer beyond tree-shaped CQs since our match-detecting mechanism is too weak. To detect matches of arbitrary CQs, we introduce the notions of forks and splittings.

4.2 Fork Rewritings: Describing Different Collapsings of a Query

Observe that a connected conjunctive query can induce several different query matches, depending on how its variables "glue" together. We formalise this concept with the forthcoming notion of fork rewritings [19, p. 4]. Moreover, as it will turn out soon, the only relevant trees for detecting query matches are exactly those trees being subtrees of the maximal fork rewritings.

Definition 8. *Let q, q' be conjunctive queries. We say that q' is obtained from q by fork elimination, and denote this fact with $q \rightsquigarrow_{\mathsf{fe}} q'$, if q' can be obtained from q by selecting two atoms $\mathrm{R}(\vec{z}, \vec{y_1}, x)$, $\mathrm{S}(\vec{y_2}, x)$ of q (where $\vec{z}$ might be empty, R and S are not necessarily different and $|\vec{y_1}| = |\vec{y_2}|$ holds) and componentwise identifying the tuples $\vec{y_1}, \vec{y_2}$. We also say that q' is a fork rewriting of q if q' is obtained from q by applying fork elimination on q possibly multiple times. When the fork elimination process is applied exhaustively on q we say that the resulting query, denoted with $\mathsf{maxfr}(q)$, is a maximal fork rewriting of q.*

Example 9. Consider a CQ $q = \mathrm{R}(x, y) \wedge \mathrm{S}(v, y) \wedge \mathrm{R}(x, z) \wedge \mathrm{R}(v, z) \wedge \mathrm{T}(y, x, z) \wedge \mathrm{T}(y, v, z)$ with atoms $\alpha_{1\text{–}6}$. Note that q has three forks: (α_1, α_2), (α_3, α_4) and (α_5, α_6). By eliminating any of them we obtain the maximal fork rewriting of q, namely $\mathsf{maxfr}(q) = \mathrm{R}(xv, y) \wedge \mathrm{R}(xv, z) \wedge \mathrm{S}(xv, y) \wedge \mathrm{T}(y, xv, z)$ with fresh xv.



By employing a special naming schemes for variables and by induction over the number of fork eliminations we can show the following lemma:

Lemma 10. *Every CQ q has the unique (up to renaming)* $\mathsf{maxfr}(q)$.

A rather immediate application of Definition 8 is that entailment of a fork rewriting of a query implies entailment of the input query itself. The proof goes via an induction over the number of fork eliminations.

Lemma 11. *Let q, q' be conjunctive queries, such that q' is obtained from q by fork elimination, and let $\mathfrak{A}$ be a structure. Then $\mathfrak{A} \models q'$ implies $\mathfrak{A} \models q$.*

4.3 Splittings: Describing Query Matches in an Abstract Way

The next notion, namely *splittings* [19, p. 4], are partitions of query variables that provide an abstract way to reason on how (a fork rewriting of) a conjunctive query matches a forest structure, without referring to either to a concrete forest or to a concrete match. Intuitively, when a query q matches a forest, its match induces a partition of variables $x \in \mathrm{Var}(q)$, according to the following scenarios:

- either x is mapped to one of the roots of the intended forest,
- or x, together with some other variables, constitute to a subtree dangling from one of the forests' roots,
- or otherwise x is mapped somewhere far inside the forest, not being directly connected to the forests' roots.

Splittings capture the above intuitions. Their definition is provided below.

Definition 12. *A* splitting Π_q *w.r.t.* $\mathcal{K} = (\mathcal{T}, \mathcal{D})$ *of* q *is a tuple*

$$\Pi_q = (\mathrm{Roots}, \mathrm{name}, \mathrm{SubTree}_1, \mathrm{SubTree}_2, \ldots, \mathrm{SubTree}_n, \text{root-}of, \mathrm{Trees}),$$

where the sets $\mathrm{Roots}, \mathrm{SubTree}_1, \ldots, \mathrm{SubTree}_n, \mathrm{Trees}$ *induce a partition of* $\mathrm{Var}(q)$, $\mathrm{name} : \mathrm{Roots} \to \mathsf{ind}(\mathcal{D})$ *is a function naming the roots and* $\text{root-}of : \{1, 2, \ldots, n\} \to \mathrm{Roots}$ *assigns to each* $\mathrm{SubTree}_i$ *an element from* Roots. *Moreover, Π_q satisfies:*

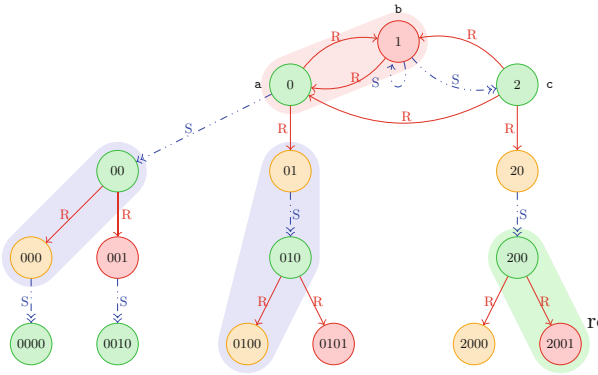
(a) the query $q{\restriction_{\mathrm{Trees}}}$ is a variable-disjoint union of tree-shaped queries,
(b) the queries $q{\restriction_{\mathrm{SubTree}_i}}$ are tree-shaped for all indices $i \in \{1, 2, \ldots, n\}$,
(c) for any atom $\mathrm{R}(\vec{x}) \in q$ the variables from $\vec{x}$ either belong to the same set or $\vec{x} = (\vec{y}, u, v, \vec{z})$ [with possibly empty $\vec{y}, \vec{z}$], where:
  - *all variables from $\vec{y}, u$ belong to* Roots,
  - *there is an index $i \in \{1, 2, \ldots, n\}$ witnessing* $\text{root-}of(i) = u$,
  - *$v \in \mathrm{SubTree}_i$ is the root of $q{\restriction_{\mathrm{SubTree}_i}}$ and variables from $\vec{z}$ are in* $\mathrm{SubTree}_i$.

(d) For any index $i \in \{1, 2, \ldots, n\}$ there is an atom $\mathrm{R}(\vec{y}, \text{root-}of(i), x_i) \in q$ [where $\vec{y}$ is possibly empty] with x_i being the root of $q{\restriction_{\mathrm{SubTree}_i}}$.

It helps to think that a splitting consists of named roots, corresponding to the database part of the model, together with some of their subtrees and of some auxiliary trees lying somewhere far from the roots.

Example 13. Consider a HAF $\mathfrak{A}$ with roots $\mathtt{a}, \mathtt{b}, \mathtt{c}$ and a (non-tree-shaped) CQ:

$$q = (\mathrm{A}(x_0) \wedge \mathrm{R}(x_0, x_1) \wedge \mathrm{R}(x_1, x_0) \wedge \mathrm{B}(x_1)) \wedge (\mathrm{S}(x_0, x_{00}) \wedge \mathrm{R}(x_{00}, x_{000})) \wedge (\mathrm{R}(x_0, x_{01}) \wedge \mathrm{S}(x_{01}, x_{010}) \wedge \mathrm{R}(x_{010}, x_{0100})) \wedge (\mathrm{A}(x_{200}) \wedge \mathrm{R}(x_{200}, x_{2001}) \wedge \mathrm{B}(x_{2001}))\,.$$



$\mathrm{Roots} = \{ x_0, x_1 \}$

$\mathrm{SubTree}_1 = \{ x_{00}, x_{000} \}$

$\mathrm{SubTree}_2 = \{ x_{01}, x_{010}, x_{0100} \}$

$\mathrm{Trees} = \{ x_{200}, x_{2001} \}$

$\mathrm{name}(x_0) = \mathtt{a}, \mathrm{name}(x_1) = \mathtt{b}$

$\text{root-}of(1) = x_0, \text{root-}of(2) = x_0$

Fig. 3. Splitting Π_q of q, compatible with $\mathfrak{A}$. Coloured areas partition variables.

We conclude the section by showing that splittings indeed correspond to query matches over forests. In order to do it, we first introduce an auxiliary definition of *compatibility* of a splitting with a HAF. Intuitively, the first item detects distant trees with the rolling-up technique, the second one describes the connections between roots and the last one detects subtrees dangling from some root.

Definition 14. *Let $\mathcal{K}$ be a closed $\mathcal{FGF}$ knowledge-base, q be a closed CQ and $\mathfrak{A}$ a HAF model of $\mathcal{K}$. A splitting Π_q w.r.t $\mathcal{K}$ of q is* compatible *with $\mathfrak{A}$ if it satisfies all the conditions below:*

(A) for all connected components $\hat{q}$ of Trees *there is a* $\mathrm{d} \in A$ *s.t.* $\mathrm{d} \in (\mathrm{Match}_{\hat{q}})^{\mathfrak{A}}$,
(B) for all $\mathrm{R}(\vec{x}) \in q$ *with all* $x_i \in \mathrm{Roots}$ *we have* $\left(\mathrm{name}(x_1)^{\mathfrak{A}}, \ldots, \mathrm{name}(x_{|\vec{x}|})^{\mathfrak{A}}\right) \in \mathrm{R}^{\mathfrak{A}}$,
(C) Take all indices $i \in \{1, 2, \ldots, n\}$ *and let v_i be the root variable of* $q{\restriction_{\mathrm{SubTree}_i}}$. *Take any $\vec{u}$ composed only of* Roots *with the last element* $\text{root-}of(i)$, *s.t.* $\mathrm{R}(\vec{u}, v_i) \in q$. *Then the tuple* $\left(\mathrm{name}(u_1)^{\mathfrak{A}}, \ldots, \mathrm{name}(u_{|\vec{u}|})^{\mathfrak{A}}\right)$ *satisfies*

$$\exists x_{|\vec{u}|+1}\ \mathrm{Subt}^{\vec{u}, v_i}_{q{\restriction_{\{\vec{u}, v_i\} \cup \mathrm{SubTree}_i}}}(\vec{x}_{1 \ldots |\vec{u}|+1})$$

We stress that the difficulties in Item (C) comes from a possible presence of higher-arity relations that link other roots before reaching $\text{root-}of(i)$.

The lemma below gathers the notions presented so far.

Lemma 15. *Let $\mathcal{K}$ be a closed $\mathcal{FGF}$-kb, q a closed CQ and a HAF model $\mathfrak{A}$ of $\mathcal{K}$. Then $\mathfrak{A} \models q$ iff there is a fork rewriting q' of q and a splitting $\Pi_{q'}$ w.r.t. $\mathcal{K}$ of q' compatible with $\mathfrak{A}$.*

4.4 Spoilers: Blocking Query Matches

Spoilers are knowledge bases dedicated to blocking compatibility of a given splitting. We define them similarly to Definition 14, in a way that there will be a tight correspondence between the cases below and those from Definition 14.

Definition 16. *Let q be a closed CQ, $\mathcal{K}$ be a closed $\mathcal{FGF}$-kb and let $\Pi_q = (\text{Roots}, \text{name}, \text{SubTree}_1, \ldots, \text{SubTree}_n, \text{root-}of, \text{Trees})$ be a splitting w.r.t $\mathcal{K}$ of q. A* spoiler $\mathcal{K}_{\neg\Pi_q} = (\mathcal{D}_{\neg\Pi_q}, \mathcal{T}_{\neg\Pi_q})$ *for Π_q is an $\mathcal{FGF}$-kb satisfying one of:*

(A) $\forall x\ \neg\text{Match}_{\hat{q}}(x) \in \mathcal{T}_{\neg\Pi_q}$ *for some tree-shaped query $\hat{q}$ from* Trees,
(B) $\neg\text{R}(\text{name}(x_1), \ldots, \text{name}(x_k)) \in \mathcal{D}_{\neg\Pi_q}$ *for some atom* $\text{R}(\vec{x}) \in q$ *with all x_i in* Roots,
(C) there is an index $i \in \{1, 2, \ldots, n\}$, a tuple of variables $\vec{u}$ composed only of Roots *with the last element* root-*of*(i)*, s.t.* $\text{R}(\vec{u}, v_i) \in q$*, where v_i is the root variable of $q\restriction_{\text{SubTree}_i}$, but*

$$\left(\neg\exists x_{|\vec{u}|+1}\ \text{Subt}^{\vec{u},v_i}_{q\restriction_{\vec{u}\cup\{v_i\}\cup\text{SubTree}_i}}(\vec{x}_{1\ldots|\vec{u}|+1})\right)\left(\text{name}(u_1), \ldots, \text{name}(u_{|\vec{u}|})\right) \in \mathcal{D}_{\neg\Pi_q}.$$

The definition of spoilers is now lifted to the case for the whole closed CQs.

Definition 17. *A* super-spoiler *for a closed CQ q and a closed $\mathcal{FGF}$ kb $\mathcal{K}$ is a minimal (in the sense the of number of axioms) $\mathcal{FGF}$ kb $\mathcal{K}_{\neg q}$ s.t. for all fork rewritings q' of q and all splittings $\Pi_{q'}$ w.r.t $\mathcal{K}$ of q', $\mathcal{K}_{\neg q}$ is a spoiler for $\Pi_{q'}$.*

The following crucial property of super-spoilers is shown next.

Lemma 18. *Let $\mathcal{K}$ be a closed $\mathcal{FGF}$ kb and let q be a closed CQ. Then $\mathcal{K} \not\models q$ iff there is a super-spoiler $\mathcal{K}_{\neg q}$ for q and $\mathcal{K}$ such that $\mathcal{K} \cup \mathcal{K}_{\neg q}$ is satisfiable.*

We now bound the total number and the sizes of super-spoilers. It is easy to see that there are only exponentially many super-spoilers, since the facts that appear in super-spoilers are also present in the input knowledge base. The challenging part is to show that super-spoilers are of polynomial size in $|\mathcal{K}|+|q|$. In order to do it, we observe that all trees that appear in spoilers are actually subtrees of the maximal fork rewriting of q. Trivially, there are only polynomially many subtrees of $\mathsf{maxfr}(q)$, so we are done. Finally, we will see that candidates for super-spoilers can be enumerated in exponential time.

Lemma 19. *Let $\mathcal{K}$ be closed $\mathcal{FGF}$ kb and q be a closed CQ. The following properties hold true: (a) super-spoilers have sizes polynomial in $|\mathcal{K}|+|q|$; (b) there are only exponentially many (in $|\mathcal{K}|+|q|$) candidates for super-spoilers; (c) super-spoilers can be enumerated in time exponential in $|\mathcal{K}| + |q|$.*

From the presented lemma we can deduce an algorithm for solving CQ entailment over $\mathcal{FGF}$ kbs. As a preliminary step we "close" both input CQ q and input kb $\mathcal{K}$. Second, we exhaustively enumerate all possible candidates $\mathcal{K}_{\neg q}$ for being a super-spoiler for $\mathcal{K}$ and q. Note that the enumeration process can be done in exponential time due to Lemma 19. After ensuring that $\mathcal{K}_{\neg q}$ is indeed a super-spoiler, we test whether $\mathcal{K} \cup \mathcal{K}_{\neg q}$ is satisfiable. The satisfiability test can be

performed in ExpTime due to the polynomial size of $\mathcal{K}_{\neg q}$ and Theorem 4. If some $\mathcal{K} \cup \mathcal{K}_{\neg q}$ is satisfiable, by Lemma 18, we conclude $\mathcal{K} \not\models q$. Otherwise we have that $\mathcal{K} \models q$. The overall process can be implemented in ExpTime, thus we conclude the second main theorem of the paper.

Theorem 20. *CQ entailment problem for $\mathcal{FGF}$ is* ExpTime*-complete.*

Note that the lower bounds are inherited from kb satisfiability problem. For readers interested in CQ entailment over finite models we can also infer ExpTime-completness of the finitary version of the problem. A (non-trivial) argument is that $\mathcal{GF}$ is *finite controllable* [7] (a CQ is entailed over all models iff it is entailed over finite models), which obviously applies also to $\mathcal{FGF}$. Hence, we obtain:

Corollary 21. *CQ finite entailment problem for $\mathcal{FGF}$ is* ExpTime*-complete.*

In the real-life applications, we usually measure the *data complexity* of both satisfiability and entailment problems, *i.e.* the case when the size of the input theory and query is treated as a constant and only $|\mathcal{D}|$ matters. The upper bound follows from $\mathcal{GF}$ [7] and the lower bound holds already for $\mathcal{ALC}$.

Corollary 22. *(Finite) satisfiability and CQ (finite) entailment problems for $\mathcal{FGF}$ are, respectively,* NP*-complete and co*NP*-complete in data-complexity.*

5 Conclusions and Future Work

In the paper we introduced a novel logic $\mathcal{FGF}$ that combines ideas of guarded quantification and forwardness. By exploiting the HAF-model property of the logic we have shown that both kb satisfiability problems and CQ entailment problems are ExpTime-complete, also in the finite.

Our results are quite encouraging and there is a lot of space for future research. We conclude by discussing some interesting open problems.

- Understanding model theory of $\mathcal{FGF}$. One can develop an appropriate notion of bisimulation for $\mathcal{FGF}$ and show an analogous of Van Benthem & Rosen characterisation theorem in the spirit of [10,20]. In the light of [12] it would be interesting to investigate Craig Interpolation and Beth Definability for $\mathcal{FGF}$.
- Understanding extensions of $\mathcal{FGF}$ with counting, constants or transitivity. We conjuncture that the extensions of $\mathcal{FGF}$ with counting quantifiers à la [21] or constants are decidable and can be shown with techniques from Sect. 3.4. Another idea is to $\mathcal{FGF}$ with transitive guards, denoted with $\mathcal{FGF}$+TG, that captures the DL $\mathcal{SH}$. Its two-variable fragment is known to be ExpSpace-complete (without database though) [14]. We believe that the combination of our techniques and those from [14,16] can be applied to infer an ExpSpace upper bound for kb sat problem for the full logic. Finally, CQ entailment for $\mathcal{GF}$+TG is undecidable [7], but we hope that it is not the case for $\mathcal{FGF}$+TG.

Acknowledgements. The author apologises for all mistakes and grammar issues that appear in the paper. He thanks A. Karykowska and P. Witkowski for proofreading, E. Kieroński for his help with the introduction, W. Faber for deadline extension and anonymous JELIA's reviewers for many useful comments.

This work was supported by the ERC Consolidator Grant No. 771779 (DeciGUT).

References

1. Andréka, H., Németi, I., van Benthem, J.: Modal languages and bounded fragments of predicate logic. J. Philos. Logic (1998)
2. Baader, F., Horrocks, I., Lutz, C., Sattler, U.: An Introduction to Description Logic. Cambridge University Press, Cambridge (2017)
3. Beeri, C., Vardi, M.Y.: The Implication Problem for Data Dependencies. In: ICALP (1981)
4. Calí, A., Gottlob, G., Kifer, M.: Taming the infinite chase: query answering under expressive relational constraints. J. Artif. Intell. Res. (2013)
5. Chandra, A.K., Kozen, D., Stockmeyer, L.J.: Alternation. J. ACM (1981)
6. Figueira, D., Figueira, S., Baque, E.P.: Finite Controllability for Ontology-Mediated Query Answering of CRPQ. KR (2020)
7. Gottlob, G., Pieris, A., Tendera, L.: Querying the Guarded Fragment with Transitivity. In: ICALP (2013)
8. Grädel, E.: Description Logics and Guarded Fragments of First Order Logic. DL (1998)
9. Grädel, E.: On the restraining power of guards. J. Symb. Log. (1999)
10. Grädel, E., Otto, M.: The Freedoms of (Guarded) Bisimulation (2013)
11. Herzig, A.: A new decidable fragment of first order logic. In: Third Logical Biennial, Summer School and Conference in Honour of S. C. Kleene (1990)
12. Hoogland, E., Marx, M., Otto, M.: Beth Definability for the Guarded Fragment. LPAR (1999)
13. Horrocks, I., Tessaris, S.: Answering Conjunctive Queries over DL ABoxes: A Preliminary Report. DL (2000)
14. Kieronski, E.: On the complexity of the two-variable guarded fragment with transitive guards. Inf. Comput. (2006)
15. Kieronski, E.: One-Dimensional Guarded Fragments. MFCS (2019)
16. Kieronski, E., Malinowski, A.: The triguarded fragment with transitivity. LPAR (2020)
17. Libkin, L.: Elements of finite model theory. In: Libkin, L. (ed.) Texts in Theoretical Computer Science. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-662-07003-1
18. Lutz, C.: Inverse Roles Make Conjunctive Queries Hard. DL (2007)
19. Lutz, C.: Two Upper Bounds for Conjunctive Query Answering in SHIQ. DL (2008)
20. Otto, M.: Elementary Proof of the van Benthem-Rosen Characterisation Theorem. Technical Report (2004)
21. Pratt-Hartmann, I.: Complexity of the guarded two-variable fragment with counting quantifiers. J. Log. Comput. (2007)
22. Pratt-Hartmann, I., Szwast, W., Tendera, L.: The fluted fragment revisited. J. Symb. Log. (2019)
23. Quine, W.: The Ways of Paradox and Other Essays, Revised edn. Harvard University Press, Cambridge (1976)
24. Rosati, R.: On the decidability and finite controllability of query processing in databases with incomplete information. PODS (2006)
25. Stockmeyer, L.: The Complexity of Decision Problems in Automata Theory and Logic (1974)