Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning
Main Track

The Sticky Path to Expressive Querying: Decidability of Navigational Queries under Existential Rules

Piotr Ostropolski-Nalewaja[1,2] , **Sebastian Rudolph**[1,3]
[1]TU Dresden
[2]University of Wrocław
[3]Center for Scalable Data Analytics and Artificial Intelligence Dresden/Leipzig
postropolski@cs.uni.wroc.pl, sebastian.rudolph@tu-dresden.de

Abstract

Extensive research in the field of ontology-based query answering has led to the identification of numerous fragments of existential rules (also known as tuple-generating dependencies) that exhibit decidable answering of atomic and conjunctive queries. Motivated by the increased theoretical and practical interest in navigational queries, this paper considers the question for which of these fragments decidability of querying extends to regular path queries (RPQs). In fact, decidability of RPQs has recently been shown to generally hold for the comprehensive family of all fragments that come with the guarantee of universal models being reasonably well-shaped (that is, being of finite cliquewidth). Yet, for the second major family of fragments, known as finite unification sets (short: fus), which are based on first-order-rewritability, corresponding results have been largely elusive so far. We complete the picture by showing that RPQ answering over arbitrary fus rulesets is undecidable. On the positive side, we establish that the problem is decidable for the prominent fus subclass of sticky rulesets, with the caveat that a very mild extension of the RPQ formalism turns the problem undecidable again.

1 Introduction

Existential rules, also known under the names *tuple-generating dependencies* (TGD) (Abiteboul, Hull, and Vianu 1995), *Datalog*$^+$ (Gottlob 2009), or $\forall\exists$*-rules* (Baget et al. 2011) have become a very popular formalism in knowledge representation and database theory, with a plethora of applications in ontology-based data access, data exchange, and many more. One of the fundamental tasks in the context of existential rules is *ontological query answering*, where a query, expressing some information need, is executed over some given data(base) taking into account background knowledge (the *ontology*), which is expressed via a set of existential rules.

The standard query language classically considered in this setting are *conjunctive queries* (CQs) (Chandra and Merlin 1977), corresponding to the select-project-join fragment of SQL. As answering[1] of CQs using unconstrained existential rules is undecidable (Chandra, Lewis, and Makowsky 1981), much research has been devoted to identifying syntactic and semantic restrictions that would warrant decidability of that task. Most investigations in that respect have been along two major lines of research, which can roughly be summarized as *forward-chaining-based* (iteratively applying the rules to the data to see if a query instance is eventually produced) on one side, and *backward-chaining-based* (iteratively applying rules "backwards" to the query to see if it is ultimately substantiated by the data) on the other (Rudolph 2014).

The advent of NoSQL and graph databases has led to an renewed interest in query languages that overcome some expressivity limitations of plain CQs by accomodating certain forms of recursion (Rudolph and Krötzsch 2013). Among the mildest such extensions are so-called *navigational queries*, which are able to express the existence of size-unbounded structural patterns in the data. Among the most popular such query languages are *regular path queries* (RPQs) and their conjunctive version (CRPQs) as well as their respective 2-way variants (2RPQs, C2RPQs) (Florescu, Levy, and Suciu 1998; Calvanese et al. 2003). In this context, the natural question arises for which of the known classes of rulesets the decidability of CQ entailment generalizes to navigational queries. As it turns out, the situation differs significantly between the forward- and the backward-chaining approaches.

Forward-chaining approaches are based on well-shaped universal models (Deutsch, Nash, and Remmel 2008), usually obtained via a construction called the *chase* (Beeri and Vardi 1984). If one can guarantee the existence of universal models with certain properties (such as being finite or having finite treewidth or – subsuming the two former cases – being of finite cliquewidth), CQ entailment is known to be decidable. Fortunately, this generic result was shown to generalize to all query languages which are simultaneously expressible in universal second-order logic and in monadic second-order logic (Feller et al. 2023), subsuming the all navigational queries considered in this paper.

Corollary 1 (following from Feller et al., 2023)**.** *Let $\mathcal{R}$ be a ruleset such that for every database $\mathcal{D}$, there exists a universal model of $\mathcal{D},\mathcal{R}$ having finite cliquewidth. Then C2RPQ answering wrt. $\mathcal{R}$ is decidable.*

This very general result establishes in a uniform way decidability of C2RPQ (thus also CRPQ, 2RPQ, and RPQ)

[1]As *query answering* and *query entailment*, seen as decision problems, are logspace-interreducible, we will use the two terms interchangeably in this paper, sometimes also simply referring to it as *querying*.

answering for all *finite cliquewidth sets* (fcs) of existential rules, subsuming various classes based on acyclicity (Grau et al. 2013) as well as the guarded family – including (weakly/jointly/glut-) guarded and (weakly/jointly/glut-) frontier-guarded rulesets (Baget et al. 2011; Calì, Gottlob, and Kifer 2013; Krötzsch and Rudolph 2011). For various of these subfragments, decidability of C2RPQ answering had been shown before, partially with tight complexities (Baget et al. 2017).

The situation is much less clear (and as of yet essentially unexplored) for backward-chaining-based decidability notions. Rulesets falling into that category are also known as *first-order-rewritable* or *finite unification sets* (fus). Some types of fus rulesets simultaneously fall under the forward-chaining case (e.g., linear rules or binary single-head fus, which are subsumed by fcs (Feller et al. 2023)) and admit decidable CRPQ entailment on those grounds. Yet, for fus rulesets and its popular syntactically defined subclasses such as *sticky rulesets* (Calì, Gottlob, and Pieris 2010), decidability of path queries has been wide open until now.

In this paper we establish the following results:

- Entailment of Boolean RPQs over fus rulesets is undecidable (shown by a reduction from the halting problem of Minsky-type two-counter machines).
- However, the same problem over sticky rulesets is decidable, as will be proven by an elaborate reduction to a finitely RPQ-controllable setting.
- Yet, if we slightly extend Boolean RPQs admitting higher-arity predicates in paths, decidability is lost even for sticky rulesets.

2 Preliminaries

Structures and homomorphisms. Let $\mathbb{F}$ be a countably infinite set of *function symbols*, each with an associated arity. We define the set of *terms* $\mathbb{T}$ as a minimal set containing three mutually disjoint, countably infinite sets of *constants* $\mathbb{C}$, *variables* $\mathbb{V}$, and *nulls* $\mathbb{N}$ that satisfies: $f(\bar{t}) \in \mathbb{T}$ for each tuple $\bar{t}$ of its elements and each symbol $f \in \mathbb{F}$ of matching arity. A *signature* $\mathbb{S}$ is a finite set of predicates. We denote the *arity* of a predicate $\mathtt{P}$ with $ar(\mathtt{P})$. An *atom* is an expression of the form $\mathtt{P}(\bar{t})$ where $\mathtt{P}$ is a predicate and $\bar{t}$ is an $ar(\mathtt{P})$-tuple of terms. Atoms of binary arity will also be referred to as *edges*. *Facts* are atoms containing only constants. An *instance* is a countable (possibly infinite) set of atoms. Moreover, we treat conjunctions of atoms as sets. A *database* is a finite set of facts. The *active domain* of an instance $\mathcal{I}$, denoted $adom(\mathcal{I})$, is the set of terms appearing in the atoms of $\mathcal{I}$. We recall that instances naturally represent first-order (FO) interpretations.

A *homomorphism* from instance $\mathcal{I}$ to instance $\mathcal{I}'$ is a function $h : adom(\mathcal{I}) \to adom(\mathcal{I}')$ such that (1) for each atom $\mathtt{P}(\bar{t})$ of $\mathcal{I}$ we have $\mathtt{P}(h(\bar{t})) \in \mathcal{I}'$, and (2) for each constant $c \in \mathbb{C}$ we have $h(c) = c$. Given a finite instance $\mathcal{I}$, a *core* of $\mathcal{I}$ is a minimal subset $\mathcal{I}'$ such that $\mathcal{I}$ homomorphically maps to it. It is well known that all cores of a finite structure are isomorphic, allowing us to speak of "the core" – denoted $core(\mathcal{I})$.

Queries. A *conjunctive query* (CQ) is an FO formula of the form: $\exists \bar{x}.\phi(\bar{x}, \bar{y})$ where ϕ is a conjunction of atoms over disjoint tuples of variables $\bar{x}, \bar{y}$. The tuple $\bar{y}$ denotes the *free variables* of ϕ. A query with no free variables is *Boolean*. A *union of conjunctive queries* (UCQ) is a disjunction of conjunctive queries having the same tuples of free variables. Seeing ϕ as a set of atoms, the definition of homomorphism naturally extends to functions from CQs to instances.

A *regular path query* (RPQ) is an expression $\exists \bar{z}.A(x, y)$ where x and y are distinct variables, $\bar{z} \subseteq \{ x, y \}$, and A is a regular expression over binary predicates from some signature. Given an instance $\mathcal{I}$ and two of its terms s and t, we write $\mathcal{I} \models A(s, t)$ to indicate that there exists a directed path P from s to t in $\mathcal{I}$ whose subsequent edge labels form a word w such that w belongs to the language of A. Given a signature $\mathbb{S}$, we define $\mathbb{S}^-$ as the set $\{\sigma^- \mid \sigma \in \mathbb{S}\}$. Given an instance $\mathcal{I}$ over signature $\mathbb{S}$, we define an instance $ud(\mathcal{I}')$ as $\mathcal{I} \cup \{ \mathtt{P}^-(y, x) \mid \mathtt{P}(x, y) \in \mathcal{I} \}$. A *two-way regular path query* (2RPQ) is defined as $A(x, y)$ where x and y are variables and A is a regular expression over predicates from $\mathbb{S} \cup \mathbb{S}^-$ for signature $\mathbb{S}$. We define C(2)RPQ as a conjunction of (2)RPQs with an existential quantifier prefix.

Deterministic Finite Automaton. A *deterministic finite automaton* (DFA) $\mathcal{A}$ is a tuple $\langle \mathbb{Q}, \Sigma, \delta, \mathtt{q}_0, \mathtt{q}_{\mathrm{fin}} \rangle$ consisting of a set $\mathbb{Q}$ of *states*, a finite set Σ of symbols called an *alphabet*, a *transition function* $\delta : \mathbb{Q} \times \Sigma \to \mathbb{Q}$, a *starting* state $\mathtt{q}_0 \in \mathbb{Q}$, and an *accepting* state $\mathtt{q}_{\mathrm{fin}} \in \mathbb{Q}$. Let $\delta^* : \mathbb{Q} \times \Sigma^* \to \mathbb{Q}$ be defined as follows: $\delta^*(\mathtt{q}, \varepsilon) = \mathtt{q}$ if ε is the empty word, and $\delta^*(\mathtt{q}, aw) = \delta^*(\delta(\mathtt{q}, a), w)$ if $a \in \Sigma$. $\mathcal{A}$ *accepts* a word $w \in \Sigma^*$ *iff* $\delta^*(\mathtt{q}_0, w) = \mathtt{q}_{\mathrm{fin}}$. The *language* of $\mathcal{A}$ is the set of words it accepts. Note that for any DFA, one can construct a regular expression representing the same language, and vice versa.

2.1 The Chase and Existential Rules

An FO formula ρ of the form $\forall \bar{x}\bar{y}.\alpha(\bar{x}, \bar{y}) \to \exists \bar{z}.\beta(\bar{y}, \bar{z})$ is called an *existential rule* (short: rule), where $\bar{x}, \bar{y}$ and $\bar{z}$ are tuples of variables, α is a conjunction of atoms, and β is an atom. We call $\alpha(\bar{x}, \bar{y})$ the *body* of ρ and $\beta(\bar{y}, \bar{z})$ its *head*, while $\bar{y}$ is called the *frontier*. CQs $\exists \bar{x}.\alpha(\bar{x}, \bar{y})$ and $\exists \bar{z}.\beta(\bar{y}, \bar{z})$ will be called the *body query* and the *head query* of ρ, respectively. A rule is *Datalog* if $\bar{z}$ is empty. A finite set of rules is simply called a *ruleset*. We may drop the universal quantifier in rules for visual clarity. Satisfaction of a rule ρ (a ruleset $\mathcal{R}$) by an instance $\mathcal{I}$ is defined as usual and is written $\mathcal{I} \models \rho$ ($\mathcal{I} \models \mathcal{R}$). Given a database $\mathcal{D}$ and a ruleset $\mathcal{R}$, we define an instance $\mathcal{I}$ to be a *model* of $\mathcal{D}$ and $\mathcal{R}$, written $\mathcal{I} \models (\mathcal{D}, \mathcal{R})$, *iff* $\mathcal{D} \subseteq \mathcal{I}$ and $\mathcal{I} \models \mathcal{R}$.

Querying under existential rules. Given a query $Q(\bar{x})$, a set of rules $\mathcal{R}$, and a database $\mathcal{D}$ along with a tuple $\bar{a}$ of constants, we say that $Q(\bar{a})$ is *entailed* by $\mathcal{D}, \mathcal{R}$ *iff* every model of $\mathcal{D}$ and $\mathcal{R}$ satisfies $Q(\bar{a})$. In such a case we write $\mathcal{D}, \mathcal{R} \models Q(\bar{a})$ or $\mathcal{D}, \mathcal{R}, \bar{a} \models Q(\bar{x})$. We then also call the tuple $\bar{a}$ a *certain answer* for the query $Q(\bar{x})$ with respect to $\mathcal{D}, \mathcal{R}$. Due to the computational similarity and easy interreducibility of the two tasks, we will not distinguish *query entailment* from *query answering* and use the two terms synonymously in this paper.

Next, we recap a specialized version of the Skolem chase (Ostropolski-Nalewaja et al. 2022). While more involved than the "mainstream" Skolem chase, we need to employ this variant to establish the required results regarding the various ruleset-transformations presented in the paper. In short, it enforces that the Skolem naming depends only on the shape of the rule head.

Isomorphism types. Two CQs have the same *isomorphism type* if one can be obtained from the other by a bijective renaming of its variables (including existentially quantified ones). We denote the isomorphism type of a CQ Φ as $\tau(\Phi)$. For a CQ $\phi(\bar{y})$ of isomorphism type τ and any existentially quantified variable z of $\phi(\bar{y})$, we introduce a $|\bar{y}|$-ary function symbol f^{τ}_{z}.

Skolemization. Let $\bar{z}$ be a tuple $\langle z_1, \dots z_k \rangle$ of variables. For a CQ $\phi = \exists \bar{z}.\psi(\bar{y}, \bar{z})$ and a mapping h from $\bar{y}$ to a set of terms, we define the *Skolemization* $sh(\phi)$ of ϕ through h as the instance $\psi(h(\bar{y}), \langle z'_1, \dots z'_k \rangle)$ where $z'_i = f^{\tau}_{z_i}(h(\bar{y}))$ and τ is the isomorphism type of $\exists \bar{z}.\psi(h(\bar{y}), \bar{z})$ with $h(\bar{y})$ treated as free variables. We call $h(\bar{y})$ the *frontier terms* of $\psi(h(\bar{y}), \bar{z}')$.

Skolem Chase. Given a ruleset $\mathcal{R}$ and an instance $\mathcal{I}$ we call a pair $\pi = \langle \rho, h \rangle$, where h is a homomorphism from the body of the rule $\rho \in \mathcal{R}$ to $\mathcal{I}$, an *$\mathcal{R}$-trigger in $\mathcal{I}$*. We define the *application* $appl(\pi, \mathcal{I})$ of the trigger π to the instance $\mathcal{I}$ as $\mathcal{I} \cup \gamma$ where γ is the Skolemization of the head query of ρ through h. Let $\Pi(\mathcal{I}, \mathcal{R})$ denote the set of $\mathcal{R}$-triggers in $\mathcal{I}$. Given a database $\mathcal{D}$ and a set of rules $\mathcal{R}$ we define the *Skolem chase* $Ch(\mathcal{D}, \mathcal{R})$ as:

$$Ch_0(\mathcal{D}, \mathcal{R}) = \mathcal{D}$$
$$Ch_{i+1}(\mathcal{D}, \mathcal{R}) = \bigcup_{\pi \in \Pi(Ch_i(\mathcal{D}, \mathcal{R}), \mathcal{R})} appl(\pi, Ch_i(\mathcal{D}, \mathcal{R}))$$
$$Ch(\mathcal{D}, \mathcal{R}) = \bigcup_{i \in \mathbb{N}} Ch_i(\mathcal{D}, \mathcal{R}).$$

$Ch(\mathcal{D}, \mathcal{R})$ is a universal model for $\mathcal{R}$ and $\mathcal{D}$, i.e., a model that homomorphically maps into any model (Ostropolski-Nalewaja et al. 2022). Thus for any database $\mathcal{D}$, a tuple of its constants $\bar{a}$, rule set $\mathcal{R}$, and a homomorphism-closed query[2] $\phi(\bar{x})$ we have: $Ch(\mathcal{D}, \mathcal{R}) \models \phi(\bar{a}) \iff \mathcal{D}, \mathcal{R} \models \phi(\bar{a})$. In words: $\phi(\bar{a})$ is entailed by $\mathcal{D}, \mathcal{R}$ *iff* it is satisfied by the particular instance $Ch(\mathcal{D}, \mathcal{R})$.

For an atom $\alpha \in Ch(\mathcal{D}, \mathcal{R})$, we define its *frontier terms* as the terms of α introduced during the chase earlier than α. The *birth atom* of a term t of $Ch(\mathcal{D}, \mathcal{R})$ is the atom introduced along t during the chase. A *join variable* of a rule is a variable appearing more than once in its body. We will find it useful to extend the above chase definition in the natural way to not just start from databases, but from arbitrary instance resulting from prior chases – including infinite ones.

2.2 Query Rewritability

Definition 2. Given a ruleset $\mathcal{R}$ and a UCQ $Q(\bar{x})$ we say that a UCQ $Q'(\bar{x})$ is a *rewriting of* $Q(\bar{x})$ *(under $\mathcal{R}$)* if and only if, for every database $\mathcal{D}$ and tuple $\bar{a}$ of its constants, we have:

$$Ch(\mathcal{D}, \mathcal{R}) \models Q(\bar{a}) \Leftrightarrow \mathcal{D} \models Q'(\bar{a}).$$

Finite Unification Sets A rule set $\mathcal{R}$ is a *finite unification set* (`fus`) *iff* every UCQ has a UCQ rewriting under $\mathcal{R}$.

Bounded Derivation Depth Property We say a ruleset $\mathcal{R}$ admits the *bounded derivation depth property* (is `bdd`) *iff* for every UCQ $Q(\bar{x})$ there exists a natural number k such that for every instance $\mathcal{I}$ and every tuple of its terms $\bar{a}$ we have:

$$Ch(\mathcal{I}, \mathcal{R}) \models Q(\bar{a}) \iff Ch_k(\mathcal{I}, \mathcal{R}) \models Q(\bar{a}).$$

It turns out that `fus` and `bdd` classes are equivalent (Calì, Gottlob, and Lukasiewicz 2009):

Lemma 3. *A ruleset $\mathcal{R}$ is* `fus` *if and only if it is* `bdd`.

Rewritings of UCQs can be obtained in a number of ways. For the purpose of this paper, we will rely on the algorithm provided by König et al. (2015). We denote the rewriting of a UCQ Q against a `bdd` ruleset $\mathcal{R}$ obtained through the algorithm presented therein by $\texttt{rew}(Q, \mathcal{R})$, or $\texttt{rew}(Q)$ in case $\mathcal{R}$ is known from the context.

2.3 Sticky Rulesets

Definition 4 (Sticky)**.** Following Calì, Gottlob, and Pieris (2010), a ruleset $\mathcal{R}$ over signature $\mathbb{S}$ is *sticky iff* there exists a marking $\gg$ of $\mathbb{S}$ assigning to each $\mathtt{P} \in \mathbb{S}$ a subset of $[1, ar(\mathtt{P})]$ called *marked positions* such that, for every $\rho \in \mathcal{R}$,

- if x is a join variable in ρ then x appears at a marked position in the head-atom of ρ, and
- if x appears in a body-atom of ρ at a marked position then x appears at a marked position in the head-atom of ρ.

Observation 5. *Let $\mathcal{I}$ be an instance, $\mathcal{R}$ a sticky ruleset, and t a term from $Ch(\mathcal{I}, \mathcal{R})$ with birth atom α. If some t' is on a marked position in α, then each atom β containing t must also contain t' on a marked position.*

Proof. We prove this by contradiction. Let i be the smallest natural number such that there exists an atom $\beta \in Ch_i(\mathcal{I}, \mathcal{R})$ containing t but not t'. Take the rule ρ that created β and pick some atom $\gamma \in Ch_{i-1}(\mathcal{I}, \mathcal{R})$ containing t to which one of ρ's body atoms was mapped. Then by assumption γ contains t' in a marked position. However, since $\mathcal{R}$ is sticky, t' must appear in a marked position in β, leading to a contradiction. □

Definition 6. Following Gogacz and Marcinkowski (2017): A ruleset is *joinless iff* none of its rule bodies contains repeated variables. Clearly, every joinless ruleset is sticky and therefore is `fus`.

2.4 On the Single-Head Assumption

Note that, throughout this paper, we assume that all rulesets are "single-head", meaning that we disallow conjunctions of multiple atoms in the heads of rules. This is without loss of generality, due to the following transformation presented by Calì, Gottlob, and Kifer (2013): Given a multi-head ruleset

[2] That is, a query whose answers are preserved under homomorphisms between instances. Both UCQs and (2)RPQs are known to be homomorphism-closed. We refer to Deutsch, Nash, and Remmel (2008) for a brief discussion.

$\mathcal{R}_{\text{multi}}$ over a signature $\mathbb{S}$, one can define the ruleset $\mathcal{R}_{\text{single}}$ as follows: take a rule ρ of $\mathcal{R}_{\text{multi}}$

$$\beta(\bar{x}, \bar{y}) \;\; \rightarrow \;\; \exists \bar{z}.\, \alpha_1(\bar{y}, \bar{z}), \ldots \alpha_n(\bar{y}, \bar{z})$$

and replace it by the following collection of rules:

$$\{\, \beta(\bar{x}, \bar{y}) \;\; \rightarrow \;\; \exists \bar{z}.\, \mathtt{P}_\rho(\bar{y}, \bar{z}) \,\}$$
$$\cup \; \{\, \mathtt{P}_\rho(\bar{y}, \bar{z}) \;\; \rightarrow \;\; \alpha_i(\bar{y}, \bar{z}) \;\mid\; i \in [n] \,\}$$

where $\mathtt{P}_\rho$ is a fresh symbol, unique for each rule ρ.

One can note that the chases of both rulesets (over any instance) coincide when restricted to $\mathbb{S}$, which means the transformation preserves fusness when restricted to queries over $\mathbb{S}$. It is also straightforward to note that stickiness is preserved as well – the marking of positions of fresh $\mathtt{P}_\rho$ symbols can be trivially reconstructed from the markings of the position of the α_i.

Therefore, all results presented in the remainder of the paper hold for multi-head rules as well.

3 Undecidability of RPQs over FUS

Theorem 7. *Boolean RPQ entailment under* fus *rulesets is undecidable.*

We shall prove the above theorem by a reduction from the halting problem for the following kinds of two-counter automata.

Two-counter automaton. A *two-counter automaton (TCA)* consists of a finite set of states $\mathbb{Q}$, two positive integer counters called $\mathtt{C_x}$ and $\mathtt{C_y}$, and an instruction $\mathit{instr}(\mathtt{q}) = \langle\, \mathtt{C}, \mathtt{d}, \mathtt{q_t}, \mathtt{q_f} \,\rangle$ for each state $\mathtt{q} \in \mathbb{Q}$ which is executed as follows:

1: **if** $\mathtt{C} \;==\; 0$ **then**
2: $\quad \mathtt{C} := \mathtt{C} + 1,$ move from q to $\mathtt{q_t}$
3: **else**
4: $\quad \mathtt{C} := \mathtt{C} + \mathtt{d},$ move from q to $\mathtt{q_f}$

where $\mathtt{C} \in \{\, \mathtt{C_x}, \mathtt{C_y} \,\}$ and $\mathtt{d} \in \{\, -1, 1 \,\}$. In each step, the automaton executes the instruction assigned to its current state and moves to the next. We also distinguish a starting state $\mathtt{q}_0 \in \mathbb{Q}$ and a halting state $\mathtt{q}_{halt} \in \mathbb{Q}$. The *TCA halting problem* asks whether a given TCA $\mathcal{M}$ can reach the halting state starting from $\mathtt{q}_0$ with both counters set to zero. For a state $\mathtt{q} \in \mathbb{Q}$ and numbers x and y representing the states of counters $\mathtt{C_x}$ and $\mathtt{C_y}$ respectively, we call the tuple $\langle\, \mathtt{q}, x, y \,\rangle$ a *configuration* of $\mathcal{M}$. Note that our TCAs are deterministic.

The above TCA is a standard variation of one presented by Minsky (1967, Chapter 14). The original automaton exhibits two types of instructions: 1) *Add unity to counter* C *and go to the next instruction*, and 2) *If counter* C *is not zero, then subtract one and jump to the n-th instruction, otherwise go to the next instruction*. It is straightforward to implement Minsky's automata by means of our TCAs. Thus, we inherit undecidability of the corresponding Halting problem.

In order to prove Theorem 7, we use the following reduction. Given a TCA $\mathcal{M}$ we construct a database $\mathcal{D}_{\text{grid}}$, ruleset $\mathcal{R}_{\text{grid}}$ and a RPQ $\mathcal{Q}_\mathcal{M}$ such that $\mathcal{M}$ halts starting from configuration $\langle\, \mathtt{q}_0, 0, 0 \,\rangle$ if and only if $\mathcal{D}_{\text{grid}}, \mathcal{R}_{\text{grid}} \models \mathcal{Q}_\mathcal{M}$.

3.1 Database $\mathcal{D}_{\text{grid}}$ and Ruleset $\mathcal{R}_{\text{grid}}$

Both the database and the ruleset are independent of the specific TCA $\mathcal{M}$ considered – thus they are fixed. The idea behind the construction is that $Ch(\mathcal{D}_{\text{grid}}, \mathcal{R}_{\text{grid}})$ represents an infinite grid. The database $\mathcal{D}_{\text{grid}}$ consists of just the two atoms $\mathtt{Succ}(a, b)$ and $\mathtt{Zero}(a)$.

Before stating $\mathcal{R}_{\text{grid}}$, it is convenient to define two abbreviations. Let $\varphi_{\text{right}}(x, x', y, z, z')$ denote the CQ

$$\mathtt{XCoord}(z,\; x) \wedge \mathtt{YCoord}(z,\; y) \wedge$$
$$\mathtt{XCoord}(z', x') \wedge \mathtt{YCoord}(z', y) \wedge \mathtt{Succ}(x, x')$$

with the intuitive meaning that *node z' is one to the right from node z on the grid*. In a similar vein, we let $\varphi_{\text{up}}(x, y, y', z, z')$ denote the CQ

$$\mathtt{XCoord}(z,\; x) \wedge \mathtt{YCoord}(z,\; y) \wedge$$
$$\mathtt{XCoord}(z', x) \wedge \mathtt{YCoord}(z', y') \wedge \mathtt{Succ}(y, y')$$

meaning that *node z' is one above node z on the grid*.

Definition 8. We define the grid-building ruleset $\mathcal{R}_{\text{grid}}$ as the following collection of rules:

$$\begin{aligned}
\mathtt{Succ}(x, x') &\;\rightarrow\; \exists x''.\, \mathtt{Succ}(x', x'') \\
\mathtt{Succ}(x, x') \wedge \mathtt{Succ}(y, y') &\;\rightarrow\; \exists z.\, \mathtt{GridPoint}(x, y, z) \\
\mathtt{GridPoint}(x, y, z) &\;\rightarrow\; \mathtt{XCoord}(z, x) \\
\mathtt{GridPoint}(x, y, z) &\;\rightarrow\; \mathtt{YCoord}(z, y) \\
\varphi_{\text{right}}(x, x', y, z, z') &\;\rightarrow\; \mathtt{IncX}(z, z') \\
\varphi_{\text{up}}(x, y, y', z, z') &\;\rightarrow\; \mathtt{IncY}(z, z') \\
\mathtt{IncX}(z, z') &\;\rightarrow\; \mathtt{DecX}(z', z) \\
\mathtt{IncY}(z, z') &\;\rightarrow\; \mathtt{DecY}(z', z) \\
\mathtt{XCoord}(z, x) \wedge \mathtt{Zero}(x) &\;\rightarrow\; \mathtt{XZero}(z, z) \\
\mathtt{YCoord}(z, y) \wedge \mathtt{Zero}(y) &\;\rightarrow\; \mathtt{YZero}(z, z)
\end{aligned}$$

Observation 9. *Ruleset $\mathcal{R}$ is* fus

Proof. Let $\mathcal{R}_1$ denote the set of the first four rules and let $\mathcal{R}_2$ denote the rest. Note that: 1) $\mathcal{R}_1$ is sticky (see Definition 4), and thus fus; 2) $\mathcal{R}_2$ terminates after a fixed number of chase steps thus is trivially bdd (and from Lemma 3 it is fus); and 3) $Ch(\mathcal{D}, \mathcal{R}) = Ch(Ch(\mathcal{D}, \mathcal{R}_1), \mathcal{R}_2)$. Then, from the definition of fus, if $\mathcal{R}_1$ is fus and $\mathcal{R}_2$ is fus then $\mathcal{R}$ is fus – thanks to the "stratification" of the two parts of $\mathcal{R}$, one can obtain the required rewritings by rewriting first against $\mathcal{R}_2$ and then rewriting the resulting UCQ against $\mathcal{R}_1$. □

Without going into detail, we note that the observation of $\mathcal{R}$ being fus, can also be readily argued using the generic framework for static analysis of rulesets by Baget et al. (2011, Section 7), according to which $\mathcal{R}$ falls into the class "sticky ▷ aGRD" where a ruleset with an acyclic graph of rule depenndencies (in our case: $\mathcal{R}_2$) is layered "on top of" a sticky ruleset (in our case: $\mathcal{R}_1$). Since both aGRD and sticky are fus, so is their stratified combination.

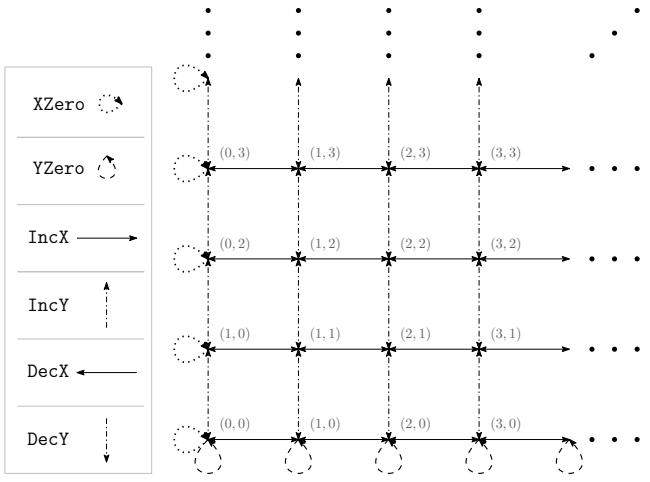


Figure 1: Depiction of $Ch(\mathcal{D}_{\text{grid}}, \mathcal{R}_{\text{grid}})$ restricted to predicates IncX, DecX, IncY, DecY, XZero, YZero, henceforth denoted $\mathcal{G}$.

3.2 Grid $\mathcal{G}$

Figure 1 visualizes $Ch(\mathcal{D}_{\text{grid}}, \mathcal{R}_{\text{grid}})$ restricted to the atoms using IncX, DecX, IncY, DecY, XZero, and YZero. As the soon to be defined RPQ $\mathcal{Q}_\mathcal{M}$ only uses these binary predicates, it is sufficient to consider this part of $Ch(\mathcal{D}_{\text{grid}}, \mathcal{R}_{\text{grid}})$, which we denote by $\mathcal{G}$.

As $\mathcal{G}$ resembles a grid, we find it convenient to introduce a few notions. First, we identify any term of $\mathcal{G}$ with a pair of natural numbers called *coordinates*. As usual, the first element of a coordinate pair is called *X-coordinate*, while the second is called *Y-coordinate*.

3.3 RPQ $\mathcal{Q}_\mathcal{M}$

We shall define the Boolean regular path query $\mathcal{Q}_\mathcal{M}$ by means of a deterministic finite automaton $\mathcal{A}_\mathcal{M}$. To simplify its definition, we will allow ourselves to write some expressions which evaluate to binary symbols from $\mathbb{S}$. We let the function $\Sigma(\cdot)$ assign predicates from Σ to operations and tests on the two counters:

$$\Sigma(\mathtt{C_x := C_x + 1}) = \mathtt{IncX}, \qquad \Sigma(\mathtt{C_y := C_y + 1}) = \mathtt{IncY},$$
$$\Sigma(\mathtt{C_x := C_x - 1}) = \mathtt{DecX}, \qquad \Sigma(\mathtt{C_y := C_y - 1}) = \mathtt{DecY},$$
$$\Sigma(\mathtt{C_x == 0}) = \mathtt{XZero}, \qquad \Sigma(\mathtt{C_y == 0}) = \mathtt{YZero}.$$

Definition of the automaton $\mathcal{A}_\mathcal{M}$. The DFA $\mathcal{A}_\mathcal{M}$ is a tuple $\langle\, \mathbb{Q}_\mathcal{A}, \Sigma_\mathcal{A}, \delta_\mathcal{A}, \mathtt{q}_0, \mathtt{q}_{\text{fin}} \,\rangle$ consisting of the set of *states* $\mathbb{Q}_\mathcal{A}$, the *alphabet* $\Sigma_\mathcal{A}$, the *transition function* $\delta_\mathcal{A}$, the *starting state* $\mathtt{q}_0$, and the *accepting state* $\mathtt{q}_{\text{fin}}$.

States. The set $\mathbb{Q}_\mathcal{A}$ of states of $\mathcal{A}_\mathcal{M}$ is $\mathbb{Q} \cup \mathbb{Q}_{aux}$ where $\mathbb{Q}_{aux}$ is a set of auxiliary states consisting of four states $\mathtt{q}^{\text{then}_1}$, $\mathtt{q}^{\text{then}_2}$, $\mathtt{q}^{\text{else}_1}$, and $\mathtt{q}^{\text{else}_2}$ for every state $\mathtt{q} \in \mathbb{Q}$. The starting (accepting) state of $\mathcal{A}_\mathcal{M}$ is the starting (halting) state of $\mathcal{M}$.

Alphabet. The alphabet $\Sigma_\mathcal{A}$ of $\mathcal{A}_\mathcal{M}$ consists of IncX, IncY, DecX, DecY, XZero, and YZero.

Transition function. The transition function $\delta_\mathcal{A}$ of $\mathcal{A}_\mathcal{M}$ is a subset of $\mathbb{Q}_\mathcal{A} \times \Sigma_\mathcal{A} \times \mathbb{Q}_\mathcal{A}$. First, we shall explain the transitions of $\mathcal{A}_\mathcal{M}$ in a graphical manner. Given a state $\mathtt{q} \in \mathbb{Q}$ recall the instruction of $\mathcal{M}$ assigned to that state.

1: **if** C == 0 **then**
2: C := C + 1, move from q to $\mathtt{q_t}$
3: **else**
4: C := C + d, move from q to $\mathtt{q_f}$

Now, we take that instruction and with it we define the following transitions as depicted in the figure below:

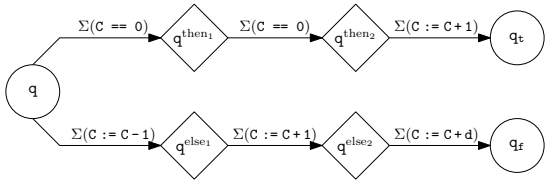


Figure 2: States other than q, $\mathtt{q_f}$, and $\mathtt{q_t}$ are part of $\mathbb{Q}_{aux}$ and are marked with diamonds in the figure. Label Σ(C==0) evaluates to XZero or YZero depending whether C is $\mathtt{C_x}$ or $\mathtt{C_y}$; other labels evaluate to IncX, IncY, DecX, or DecY. The top branch of the figure corresponds to the "**then**" branch of the instruction assigned to the state q of $\mathcal{M}$. Note that Σ(C==0) appears twice on the top branch. This redundancy is added to ensure an equal number of states on each branch toward establishing Observation 11; it is by no means critical to the proof.

The above figure encodes six triples from the set $\mathbb{Q}_\mathcal{A} \times \Sigma_\mathcal{A} \times \mathbb{Q}_\mathcal{A}$ (for example, $\langle\, \mathtt{q},\ \Sigma(\mathtt{C == 0}),\ \mathtt{q}^{\text{then}_1} \,\rangle$ is one of them). To obtain the transition function $\delta_\mathcal{A}$ of $\mathcal{A}_\mathcal{M}$, we take the union of the above-defined triples over the instruction set of $\mathcal{M}$.

Definition of the query $\mathcal{Q}_\mathcal{M}$. Let A be any regular expression that defines the same regular language as $\mathcal{A}_\mathcal{M}$, and let B be the regular expression resulting from concatenating XZero, YZero, and A. Then the Boolean RPQ $\mathcal{Q}_\mathcal{M}$ is expressed as $\exists x, y.B(x, y)$.

3.4 Correspondence between $\mathcal{Q}_\mathcal{M}$ and $\mathcal{M}$

In this section we shall show the following lemma:

Lemma 10. *$Ch(\mathcal{D}_{\text{grid}}, \mathcal{R}_{\text{grid}}) \models \mathcal{Q}_\mathcal{M}$ if and only if $\mathcal{M}$ halts starting from $\langle\, 0, 0, \mathtt{q}_0 \,\rangle$.*

To this end, we shall imagine a B-path corresponding to a match of $\mathcal{Q}_\mathcal{M}$ to $\mathcal{G}$. Its two first steps have to be over symbols XZero and YZero which is possible only at the $\langle\, 0, 0 \,\rangle$ coordinates of the grid $\mathcal{G}$. After this point, the query uses the automaton $\mathcal{A}_\mathcal{M}$ to define its behavior. Therefore, we will be discussing states and transitions of $\mathcal{A}_\mathcal{M}$ in the context of $\mathcal{M}$. Imagine the automaton $\mathcal{A}_\mathcal{M}$ starting at coordinates $\langle\, 0, 0 \,\rangle$ and "walking" over $\mathcal{G}$ and "reading" its binary predicates. We say that the automaton $\mathcal{A}_\mathcal{M}$ at coordinates $\langle\, x, y \,\rangle$ and in state q is in *configuration* $\langle\, x, y, \mathtt{q} \,\rangle$. To prove Lemma 10, and thus Theorem 7, it is enough to inductively use the following observation:

Observation 11. *For all natural numbers x, x', y, and y' and every pair of states $\mathtt{q}, \mathtt{q}' \in \mathbb{Q}$ the following two are equivalent:*

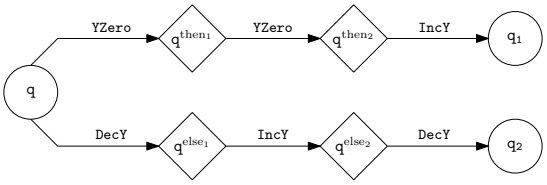
- *Automaton $\mathcal{A}_\mathcal{M}$ transitions from configuration $\langle\, x, y, \mathtt{q} \,\rangle$ to $\langle\, x', y', \mathtt{q}' \,\rangle$ in three steps.*
- *TCA $\mathcal{M}$ transitions from $\langle\, x, y, \mathtt{q} \,\rangle$ to $\langle\, x', y', \mathtt{q}' \,\rangle$ in one step.*

Proof. We shall inspect the case when the instruction from Fig. 3 is assigned in $\mathcal{M}$ to the state $\mathtt{q}$.

1: **if** $\mathtt{C_y}$ == $\mathtt{0}$ **then**
2: $\quad \mathtt{C_y} := \mathtt{C_y} + \mathtt{1}$, move from $\mathtt{q}$ to $\mathtt{q_1}$
3: **else**
4: $\quad \mathtt{C_y} := \mathtt{C_y} - \mathtt{1}$, move from $\mathtt{q}$ to $\mathtt{q_2}$

Figure 3: Instruction assigned to state $\mathtt{q}$ of TCA $\mathcal{M}$.

Then the part of $\mathcal{A}_\mathcal{M}$ responsible for transitioning out of $\mathtt{q}$ has the following form:



Assume $y = 0$. Then $\mathcal{M}$ transitions to $\langle\, \mathtt{q_1}, x, 1 \,\rangle$. We argue that the automaton $\mathcal{A}_\mathcal{M}$ follows in three steps to the same configuration. First, note that the automaton cannot transition through the bottom branch of Fig. 3 as it cannot follow a DecY-edge in $\mathcal{G}$ while being at zero Y-coordinate. However, $\mathcal{A}_\mathcal{M}$ can move through the top branch, transitioning to $\langle\, \mathtt{q_1}, x, 1 \,\rangle$.

Assume $y \neq 0$. Then $\mathcal{M}$ transitions to $\langle\, \mathtt{q_2}, x, y+1 \,\rangle$. Note that $\mathcal{A}_\mathcal{M}$ cannot follow the top branch (there is no YZero outgoing edge at $\langle\, x, y \,\rangle$ in $\mathcal{G}$, as $y > 0$). It can, however, follow the bottom one transitioning to $\langle\, \mathtt{q_2}, x, y-1 \,\rangle$. The other cases follow analogously. □

4 Deciding RPQs over Sticky Rulesets

This section is dedicated to the following theorem:

Theorem 12. *RPQ entailment from sticky rulesets is decidable.*

To simplify the intricate proof of the above, we restrict the query language to plain RPQs. However, it is straightforward to obtain the following as a consequence:

Corollary 13. *2RPQ entailment from sticky rulesets is decidable.*

One can obtain Corollary 13 by a simple reduction. Take a sticky ruleset $\mathcal{R}$ over signature $\mathbb{S}$, a database $\mathcal{D}$ and a 2RPQ $\mathcal{Q}$. Keep $\mathcal{D}$ intact, add a single rule $\mathtt{E}(x, y) \to \mathtt{E}'(y, x)$ to $\mathcal{R}$ for every binary predicate $\mathtt{E} \in \mathbb{S}$, and replace every occurence of the inverted symbol $\mathtt{E}^-$ by $\mathtt{E}'$ in the regular expression of $\mathcal{Q}$. It should be clear that entailment for the original problem coincides with that of the transformed problem. Finally, note that the transformation preserves stickiness of the ruleset.

The rest of the section is dedicated to the proof of Theorem 12. ***Until the end of this section, we fix a database $\mathcal{D}$, and a sticky ruleset $\mathcal{R}$.*** We show decidability through two semi-decision procedures.

Caveat: no repeated variables in RPQs. We emphasize that under our definition, RPQs of the form $A(x, x)$ and $\exists x.A(x, x)$ are not allowed, which is crucial for our proof. Queries with "variable joining" would require distinct techniques and would, arguably, better fit into the category of *conjunctive* (2)RPQs.

4.1 Recursive Enumerability

This part exploits an easy reduction of RPQ entailment to a first-order entailment problem.

Observation 14. *Given a Boolean RPQ $Q = \exists x, y.A(x, y)$ one can write a Datalog ruleset $\mathcal{R}_Q$ with one distinguished nullary predicate* GOAL*, such that:*

$$\mathcal{D},\ \mathcal{R} \models Q \quad \iff \quad \mathcal{D},\ \mathcal{R} \cup \mathcal{R}_Q \models \textsc{Goal}.$$

Proof. See supplementary material, **??**, in the full version (Ostropolski-Nalewaja and Rudolph 2024). □

As $\mathcal{D} \wedge \mathcal{R} \wedge \mathcal{R}_\mathcal{Q}$ is an FO theory, we can invoke completeness of FOL to recursively enumerate consequences of $\mathcal{D}, \mathcal{R} \cup \mathcal{R}_\mathcal{Q}$ and therefore semi-decide $\mathcal{D}, \mathcal{R} \cup \mathcal{R}_\mathcal{Q} \models$ GOAL. Thus, we can semi-decide $\mathcal{D}, \mathcal{R} \models \mathcal{Q}$ as well.

4.2 Co-Recursive Enumerability

This part presents a greater challenge and necessitates a novel approach. We leverage the fact that, following a specific pre-processing of $\mathcal{D}$ and $\mathcal{R}$, the non-entailment of RPQs is witnessed by a finite instance

Definition 15. Given a query language $\mathbb{L}$ and a ruleset $\mathcal{R}$ we say that $\mathcal{R}$ is *finitely $\mathbb{L}$-controllable iff* for every $Q \in \mathbb{L}$ and for every database $\mathcal{D}$ such that $\mathcal{D}, \mathcal{R} \not\models Q$ there exists a finite model $\mathcal{M}$ of $\mathcal{D}$ and $\mathcal{R}$ s.t. $\mathcal{M} \not\models Q$.

Lemma 16. *One can compute a database $\mathcal{D}^+$ and a ruleset $\mathcal{R}^+$ such that $\mathcal{R}^+$ is finitely RPQ-controllable and such that for every Boolean RPQ $\mathcal{Q}$ we have $\mathcal{D}, \mathcal{R} \models \mathcal{Q} \iff \mathcal{D}^+, \mathcal{R}^+ \models \mathcal{Q}$.*

In view of this lemma, we can "co-semidecide" $\mathcal{D}, \mathcal{R} \models \mathcal{Q}$ (that is, semi-decide $\mathcal{D}, \mathcal{R} \not\models \mathcal{Q}$) by recursively enumerating all finite instances and terminating whenever a $\mathcal{Q}$-countermodel of $\mathcal{D}^+, \mathcal{R}^+$ is found.

4.3 Overview of the Proof of Lemma 16

Before we begin discussing the proof, let us introduce a specific class of rulesets:

Stellar rules. A *stellar rule* is a rule with at most one join variable, which also must be a frontier variable. A ruleset comprising only stellar rules is *stellar*.

The ultimate goal is to construct $\mathcal{R}^+$ as a stellar ruleset. The reason behind this is as follows:

Lemma 17. *Any ruleset consisting of stellar rules is finitely RPQ-controllable.*

We will first prove the above lemma, then we will show the construction of $\mathcal{R}^+$ and $\mathcal{D}^+$.

4.4 Finite RPQ-Controllability

Assume $\mathcal{S}$ is a stellar ruleset. To prove its finite RPQ-controllability, we present, given some database and a non-entailed RPQ, a construction of a finite countermodel. This involves two key parts: the "finite model" and the "counter" parts. We start with the first.

For any instance $\mathcal{I}$ and two of its terms t, t' we let $\mathcal{I}[t, t' \mapsto t'']$ denote the structure obtained from $\mathcal{I}$ by replacing each occurrence of t or t' by the fresh null t''.

Preservation of modelhood under this kind of "term merging" is trivial for rules without join variables:

Observation 18. *Let $\mathcal{I}$ be an instance with terms s, t and ρ a joinless rule.Then $\mathcal{I} \models \rho$ implies $\mathcal{I}[s, t \mapsto u] \models \rho$.*

Proof. Note, as ρ is joinless, identification of terms cannot create new active triggers. □

For stellar rules with a join variable, we define a kind of 1-type that indicates which terms are similar enough so they can be "merged" while preserving modelhood.

Stellar types. A *stellar query* (SQ) is a CQ $S(y)$ such that y is the only join variable of S and each atom of S has at most one occurrence of y. Given a term t of some instance $\mathcal{I}$ the *stellar type* $\bigstar(t)$ of t is the (up to equivalence) most specific[3] SQ $S(y)$ such that $\mathcal{I} \models S(t)$.

Observation 19. *Let $\mathcal{I}$ be an instance with terms s, t satisfying $\bigstar(s) = \bigstar(t)$ and ρ a stellar rule with a single join variable. Then $\mathcal{I} \models \rho$ implies $\mathcal{I}[s, t \mapsto u] \models \rho$.*

Proof. Observe $\bigstar(s) = \bigstar(t) = \bigstar(u)$ and that the stellar types of other terms remain unchanged after identification ($\heartsuit$). Let $\alpha(\bar{x})$ be a head of ρ, with $\bar{x}$ a tuple of its frontier variables. Note that as ρ is stellar, $\bar{x}$ contains its join-variable denoted by x. Let H_v be the set of homomorphisms from a body of ρ to $\mathcal{I}$ such that x is always mapped to v. Note, it can be determined whether every trigger in the set $\langle\, \rho, h \,\rangle \mid h \in H_v$ is satisfied based only on the stellar type of v. Therefore, from ($\heartsuit$), the satisfaction of $\alpha(\bar{x})$ is preserved. □

Corollary 20. *Let $\mathcal{I}$ be an instance with terms s, t satisfying $\bigstar(s) = \bigstar(t)$, and let $\mathcal{S}$ be a stellar ruleset. Then $\mathcal{I} \models \mathcal{S}$ implies $\mathcal{I}[s, t \mapsto u] \models \mathcal{S}$.*

Proof. From Observation 18 and Observation 19. □

We now begin the "counter" part of the finite countermodel construction. Note its independence from $\mathcal{R}$.

Regular types. Given a DFA $\mathcal{A} = \langle\, \mathbb{Q}, \Sigma, \delta, \mathtt{q}_0, \mathtt{q}_{\mathrm{fin}} \,\rangle$, an instance $\mathcal{I}$, and one of its terms t, we define the *regular type* of t, denoted $\uparrow_{\mathcal{A}}(t)$ as the set containing all expressions $\mathtt{q} \rightsquigarrow \mathtt{q}'$ for which t has an outgoing w-path for some w with $\delta^*(\mathtt{q}, w) = \mathtt{q}'$.

Lemma 21. *Let $\mathcal{Q}$ be a Boolean RPQ and $\mathcal{A}$ the DFA $\mathcal{A}$ defining it. Let $\mathcal{I}$ be an instance and t, t' two of its terms satisfying $\uparrow_{\mathcal{A}}(t) = \uparrow_{\mathcal{A}}(t)$. Then:*

$$\mathcal{I} \models \mathcal{Q} \quad \Longleftrightarrow \quad \mathcal{I}[t, t' \mapsto t''] \models \mathcal{Q}.$$

[3] that is: minimal under CQ containment

Proof. We note that the regular types present in both $\mathcal{I}$ and $\mathcal{I}[t, t' \mapsto t'']$ are the same. On the other hand, for any instance $\mathcal{J}$ we have $\mathcal{J} \models \mathcal{Q}$ *iff* there exists a term s of $\mathcal{J}$ satisfying $\mathtt{q}_0 \rightsquigarrow \mathtt{q}_{\mathrm{fin}} \in \uparrow_{\mathcal{A}}(s)$. □

We are now ready to show that $\mathcal{S}$ is finitely RPQ-controllable. Let $\mathcal{D}'$ be a database and Q be a Boolean RPQ. Suppose $Ch(\mathcal{D}', \mathcal{S}) \not\models Q$. Let $\mathcal{I}$ be an instance obtained from $Ch(\mathcal{D}', \mathcal{S})$ by identifying all nulls that share the same regular and stellar types. With the support of Corollary 20 and Lemma 21, given that there are only finitely many stellar and regular types for Q, we conclude that $\mathcal{I}$ is finite, acts as a model of $\mathcal{D}'$ and $\mathcal{S}$, and does not entail Q. Thus, Lemma 17 is established.

4.5 Construction of $\mathcal{D}^+$ and $\mathcal{R}^+$

In this section, we construct $\mathcal{D}^+$ and $\mathcal{R}^+$ satisfying the two following lemmas. These, together with Lemma 17, establish Lemma 16.

Lemma 22. *The ruleset $\mathcal{R}^+$ is stellar.*

Lemma 23. *For every Boolean query Q it holds:*

$$Ch(\mathcal{D}, \mathcal{R}) \models Q \iff Ch(\mathcal{D}^+, \mathcal{R}^+) \models Q.$$

In order to ensure the first of the two lemmas, we perform a couple of transformations on the initially fixed sticky ruleset $\mathcal{R}$. As we progress, we maintain intermediate variants of the second lemma.

Rewriting-away non-stellar rules For this step, we heavily rely on the stickiness (and thus `fus`ness) of $\mathcal{R}$. The below relies on the idea that for `fus` rulesets, rewriting can be also applied to bodies of rules – yielding a new equivalent ruleset. This transformation preserves stickiness of the input ruleset and, importantly, it preserves Boolean RPQ entailment.

Definition 24. Given an existential rule $\rho \in \mathcal{R}$ of the form: $\alpha(\bar{x}, \bar{y}) \to \exists \bar{z}.\ \beta(\bar{y}, \bar{z})$ let $\mathtt{rew}(\rho, \mathcal{R})$ be the ruleset:

$$\Big\{\ \gamma(\bar{x}', \bar{y}) \ \to\ \exists \bar{z}.\ \beta(\bar{y}, \bar{z}) \ \Big|\ \exists \bar{x}'.\gamma(\bar{x}', \bar{y}) \ \in\ \mathtt{rew}(\exists \bar{x}.\alpha(\bar{x}, \bar{y}), \mathcal{R}) \ \Big\}.$$

Finally, let $\mathtt{rew}(\mathcal{R}) = \mathcal{R} \cup \bigcup_{\rho \in \mathcal{R}} \mathtt{rew}(\rho, \mathcal{R})$.

Lemma 25. *The ruleset $\mathtt{rew}(\mathcal{R})$ is sticky.*

Proof. See supplementary material, **??**, in the full version (Ostropolski-Nalewaja and Rudolph 2024). □

Lemma 26. *For every Boolean RPQ $\mathcal{Q}$ we have:*

$$Ch(\mathcal{D}, \mathcal{R}) \models \mathcal{Q} \iff Ch(\mathcal{D}, \mathtt{rew}(\mathcal{R})) \models \mathcal{Q}.$$

Proof. See supplementary material, **??**, in the full version (Ostropolski-Nalewaja and Rudolph 2024). □

We are now prepared to introduce the primary tool of the construction. The conceptual basis of this tool can be traced back to Ostropolski-Nalewaja et al. (2022).

Definition 27. A ruleset $\mathcal{R}'$ is *quick iff* for every instance $\mathcal{I}$ and every atom β of $Ch(\mathcal{I}, \mathcal{R}')$ if all frontier terms of β appear in $adom(\mathcal{I})$ then $\beta \in Ch_1(\mathcal{I}, \mathcal{R}')$.

The technique of "quickening" fus rulesets is quite versatile, allowing for the simplification of rulesets and the general streamlining of proof-related reasoning.

Lemma 28. *For any* fus *ruleset* $\mathcal{R}$, $\mathtt{rew}(\mathcal{R})$ *is quick.*

Proof. Suppose $\beta(\bar{t}, \bar{s}) \in Ch(\mathcal{I}, \mathtt{rew}(\mathcal{R}))$, where $\bar{t}$ represents the frontier terms of β. Assume that $\bar{t}$ appears in $\mathcal{I}$, but β does not. Let ρ be the rule that created β during the chase and let it be of the following, general form: $\alpha(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}.\ \beta(\bar{y}, \bar{z})$. From the above, note that $\mathcal{I}, \mathtt{rew}(\mathcal{R}) \models \exists \bar{x}.\alpha(\bar{x}, \bar{t})$.

If $\rho \in \mathcal{R}$ then there exists $\gamma(\bar{x}', \bar{y})$ such that: $(\exists \bar{x}'.\gamma(\bar{x}', \bar{y})) \in \mathtt{rew}(\exists \bar{x}.\alpha(\bar{x}, \bar{y}), \mathcal{R})$ and $\mathcal{I} \models \exists \bar{x}'.\gamma(\bar{x}', \bar{t})$. Therefore, from definition of $\mathtt{rew}(\mathcal{R})$, there exists a rule $\rho' \in \mathtt{rew}(\mathcal{R})$ (with γ begin its body) such that ρ' derives β in $Ch_1(\mathcal{I}, \mathcal{R}')$.

If $\rho \notin \mathcal{R}$ then let ρ' be such that $\rho \in \mathtt{rew}(\rho', \mathcal{R})$ and $\rho' \in \mathcal{R}$. As $\rho \in \mathtt{rew}(\rho', \mathcal{R})$ we know that ρ' can be used as well during the chase to obtain β. Thus, we can use the above reasoning for ρ' instead of ρ to complete the proof of the lemma. □

Taking cores of rules. The ruleset we have ($\mathtt{rew}(\mathcal{R})$) is almost sufficient for constructing $\mathcal{R}^+$. Given our focus on sticky rulesets, we are particularly interested in variable joins appearing in rules. The rationale behind the following transformation step can be explained using the simple conjunctive query $\Phi(x) = \exists y, y'.\mathtt{E}(x, y), \mathtt{E}(x, y')$. The variable join on x is unnecessary to express Φ as it is equivalent to its core $\Phi'(x) = \exists y.\mathtt{E}(x, y)$. This motivates the following:

Definition 29. Given an existential rule ρ of the form $\alpha(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}.\beta(\bar{y}, \bar{z})$, we define $core(\rho)$ as the existential rule $\alpha'(\bar{x}', \bar{y}) \rightarrow \exists \bar{z}.\beta(\bar{y}, \bar{z})$ where α' is a core of α with variables $\bar{y}$ treated as constants.

Definition 30. Let $\mathtt{cr}(\mathcal{R})$ be $\{\ core(\rho) \mid \rho \in \mathtt{rew}(\mathcal{R})\ \}$.

As taking cores of bodies of rules produces an equivalent ruleset we have:

Observation 31. $Ch(\mathcal{D}, \mathtt{cr}(\mathcal{R})) = Ch(\mathcal{D}, \mathtt{rew}(\mathcal{R}))$*, and* $\mathtt{cr}(\mathcal{R})$ *is both sticky and quick.*

Introducing redundant stellar rules. We are one step away from defining $\mathcal{R}^+$. Let $\mathtt{cr}^+(\mathcal{R})$ be the ruleset obtained from $\mathtt{cr}(\mathcal{R})$ by augmenting it with a set of additional rules as follows: For every non-stellar rule ρ in $\mathtt{cr}(\mathcal{R})$, let $\mathtt{cr}^+(\mathcal{R})$ also contain the rule derived from ρ by substituting all its join variables by one and the same fresh variable. The following is straightforward:

Observation 32. $Ch(\mathcal{D}, \mathtt{cr}^+(\mathcal{R})) = Ch(\mathcal{D}, \mathtt{rew}(\mathcal{R}))$*, and* $\mathtt{cr}^+(\mathcal{R})$ *is sticky and quick.*

Proof. The first holds, as anytime a rule of $\mathtt{cr}^+(\mathcal{R}) \setminus \mathtt{cr}(\mathcal{R})$ is used to derive an atom, the original rule of $\mathtt{cr}(\mathcal{R})$ can be used instead. This argument also ensures the quickness of $\mathtt{cr}^+(\mathcal{R})$. Finally, $\mathtt{cr}^+(\mathcal{R})$ is sticky because the marking of variables for new rules of $\mathtt{cr}^+(\mathcal{R})$ can be directly inherited from $\mathtt{cr}(\mathcal{R})$. □

Now we are ready to introduce $\mathcal{R}^+$.

Definition 33. Let $\mathcal{R}^+$ be obtained from $\mathtt{cr}^+(\mathcal{R})$ by removing all rules with two or more join variables.

The following lemma shows, that, as far as the derivation of binary atoms is concerned, $\mathcal{R}^+$ almost perfectly mimics $\mathtt{cr}^+(\mathcal{R})$. The only limitation is that $\mathcal{R}^+$ might not be able to derive all binary atoms over database constants.

Lemma 34. $Ch(\mathcal{D}, \mathcal{R}^+)$ *and* $Ch(\mathcal{D}, \mathtt{cr}^+(\mathcal{R}))$ *when restricted to the binary atoms containing at least one non-constant term are equal.*

Proof. Consider $Ch(\mathcal{D}, \mathtt{cr}^+(\mathcal{R}))$. From the definition of $\mathtt{cr}^+(\mathcal{R})$ we know that, whenever a rule containing more than one join variable produces an atom having exactly one and the same term on all its marked positions, then a rule from $\mathtt{cr}^+(\mathcal{R}) \setminus \mathtt{cr}(\mathcal{R})$ can be used to produce exactly the same atom. Note this is also thanks to the specific version of the Skolem chase we use. Therefore, we will assume that rules of $\mathtt{cr}^+(\mathcal{R}) \setminus \mathtt{cr}(\mathcal{R})$ are used whenever possible.

Consider all atoms in the chase $Ch(\mathcal{D}, \mathtt{cr}^+(\mathcal{R}))$ that were necessarily generated by a rule having at least two distinct join variables – that is, they couldn't have been created by any rule from $\mathtt{cr}^+(\mathcal{R}) \setminus \mathtt{cr}(\mathcal{R})$. Denote this set with S. Note that due to stickiness of $\mathtt{cr}^+(\mathcal{R})$ (Observation 32) atoms in S have two distinct frontier terms on $\mathtt{cr}^+(\mathcal{R})$-marked positions. Therefore, every atom that requires for its derivation an atom β from S has to contain both such terms from β (♣).

Let us categorize all binary atoms of $Ch(\mathcal{D}, \mathtt{cr}^+(\mathcal{R}))$ into three distinct groups:

1. Atoms created by non-Datalog rules.
2. Atoms created by Datalog rules and containing at least one term that is not a constant.
3. Atoms containing two constants.

To prove the lemma, it suffices to show that atoms of the first and second kind are neither contained in S (♡) nor does their derivation depend on atoms from S (♢).

Atoms of the first kind trivially satisfy (♡): as $\mathtt{cr}^+(\mathcal{R})$ is sticky, its non-Datalog rules can have at most one join variable. They also satisfy (♢) since binary atoms created by non-Datalog rules cannot contain their birth term and two distinct frontier terms.

For the second category of atoms, the argument is more involved. Let $\alpha(s, t)$ be an atom of the second kind and let ρ be the rule that created $\alpha(s, t)$ in the chase. Let h be the homomorphism witnessing this. Assume w.l.o.g. that s is created in the chase no later than t and that t is not a constant.

We shall argue (♢) for α as follows. Assume that t was created during the $(i-1)$th step of the chase, and let $\gamma(\bar{u}, t)$ be the birth atom of t. Note that $\gamma(\bar{u}, t)$ is the only atom containing t in $Ch_{i-1}(\mathcal{D}, \mathtt{cr}^+(\mathcal{R}))$, and as $\mathtt{cr}^+(\mathcal{R})$ is quick, $\alpha(s, t)$ appears in $Ch_i(\mathcal{D}, \mathtt{cr}^+(\mathcal{R}))$. Assume, towards a contradiction, that (♢) does not hold for $\alpha(s, t)$. Therefore, there exist two distinct terms u, v in $Ch_{i-1}(\mathcal{D}, \mathtt{cr}^+(\mathcal{R}))$ that are contained in $\alpha(s, t)$, and these two terms are frontier terms of some atoms in $Ch_{i-1}(\mathcal{D}, \mathtt{cr}^+(\mathcal{R}))$ (from ♣),

therefore both could not be t. As $\alpha(s,t)$ is a binary atom, we have a contradiction.

Let us argue that ($\heartsuit$) holds for $\alpha(s,t)$. From above, we know that ρ contains a $\gamma(\bar{x},y)$ atom in its body with $h(y)=t$, where y is a frontier variable. Assume towards contradiction that ρ has two distinct join variables. Denote them with z and w. Note, as $\alpha(s,t)$ is binary, z and w map to s and t through h. Without loss of generality assume $h(z)=s$ and $h(w)=t$. However, ρ cannot have three distinct frontier variables – each join variable should be a frontier variable from stickiness of $\mathtt{cr}^+(\mathcal{R})$. Therefore $w=y$. We shall argue that y cannot be a join variable. As ρ is not stellar – we assumed it has two distinct join variables y and z – it is in $\mathtt{cr}(\mathcal{R})$. We shall show a contradiction with the fact that the body of ρ is a core. Assume that $\gamma(\bar{u},t)$ is as in the argument above – specifically it is the only atom containing t just before creation of $\alpha(s,t)$. From this and the fact that y is a join variable we note that $\gamma(\bar{x}',y)$ is an atom of the body of ρ and that $\bar{x}\neq\bar{x}'$. Note that the following cannot occur at the same time: 1) the body of ρ contains both $\gamma(\bar{x},y)$ and $\gamma(\bar{x}',y)$; 2) the body of ρ has exactly two join variables; 3) the body of ρ is a core. From this contradiction we get ($\heartsuit$) for α. □

Note that in the above, the only binary atoms that are missing are those which $\mathtt{cr}^+(\mathcal{R})$ derives over database constants. This can be easily rectified:

Definition 35. Let $\mathcal{D}^+ = Ch(\mathcal{D},\mathtt{cr}^+(\mathcal{R}))|_{adom(\mathcal{D})}$

Lemma 16 requires that $\mathcal{D}^+$ is computable. Yet, as $\mathtt{cr}^+(\mathcal{R})$ is sticky and therefore $\mathtt{fus}$, this is the case.

Observation 36. *$\mathcal{D}^+$ is computable.*

Proof. As $\mathtt{cr}^+(\mathcal{R})$ is $\mathtt{fus}$, we rewrite every atomic query and ask if it holds for any tuple of constants of $\mathcal{D}$. □

Corollary 37. *For every Boolean RPQ Q $Ch(\mathcal{D},\mathtt{cr}^+(\mathcal{R}))\models Q \iff Ch(\mathcal{D}^+,\mathcal{R}^+)\models Q$.*

Proof. Directly from Lemma 34, the fact that $\mathcal{R}^+ \subseteq \mathtt{cr}^+(\mathcal{R})$, and $Ch(\mathcal{D}^+,\mathtt{cr}^+(\mathcal{R})) = Ch(\mathcal{D},\mathtt{cr}^+(\mathcal{R}))$. □

From the above, Lemma 26 and Observation 32 we get:

$$Ch(\mathcal{D},\mathcal{R})\models Q \iff Ch(\mathcal{D}^+,\mathcal{R}^+)\models Q$$

and therefore Lemma 23. Moreover, by the definition of $\mathcal{R}^+$ we get that $\mathcal{R}^+$ is stellar and thus we ensure Lemma 22, concluding our overall argument.

5 Undecidability and Stickiness

In this section we show two seemingly harmless generalizations of the (decidable) case studied in the previous section. Importantly, both lead to undecidability and, given that both generalizations are rather slight, they highlight that the identified decidability result is not very robust.

5.1 Generalizing RPQs

Rather than having RPQs restricted to only use binary predicates, we may include higher-arity predicates, with the assumption that only the first two positions matter for forming paths. We refer to RPQs that permit such slightly extended regular expressions as *higher-arity regular path queries* (HRPQs). We obtain the following:

Theorem 38. *Boolean HRPQ entailment under sticky rulesets is undecidable.*

In order to prove the theorem, one can use the proof of Theorem 7 with a small tweak. Consider the ruleset from Definition 8 with the last six rules tweaked:

$$\begin{aligned}
\mathtt{Succ}(x,x') &\rightarrow \exists x''.\,\mathtt{Succ}(x',x'')\\
\mathtt{Succ}(x,x')\wedge\mathtt{Succ}(y,y') &\rightarrow \exists z.\,\mathtt{GridPoint}(x,y,z)\\
\mathtt{GridPoint}(x,y,z) &\rightarrow \mathtt{XCoord}(z,x)\\
\mathtt{GridPoint}(x,y,z) &\rightarrow \mathtt{YCoord}(z,y)\\
\varphi_{\mathrm{right}}(x,x',y,z,z') &\rightarrow \mathtt{IncX}(z,z',x,x',y)\\
\varphi_{\mathrm{up}}(x,y,y',z,z') &\rightarrow \mathtt{IncY}(z,z',x,y,y')\\
\mathtt{IncX}(z,z',u,v,t) &\rightarrow \mathtt{DecX}(z',z,u,v,t)\\
\mathtt{IncY}(z,z',u,v,t) &\rightarrow \mathtt{DecY}(z',z,u,v,t)\\
\mathtt{XCoord}(z,x)\wedge\mathtt{Zero}(x) &\rightarrow \mathtt{XZero}(z,z,x)\\
\mathtt{YCoord}(z,y)\wedge\mathtt{Zero}(y) &\rightarrow \mathtt{YZero}(z,z,y)
\end{aligned}$$

and note that it is sticky – there exists a trivial marking of positions for it. The rest of the proof of Theorem 7 remains unchanged, observing that when we project $\mathtt{IncX}$, $\mathtt{IncY}$, $\mathtt{DecX}$, $\mathtt{DecY}$, $\mathtt{XZero}$, and $\mathtt{YZero}$ to the first two positions, we obtain the previous ruleset.

5.2 Generalizing stickiness

Alternatively, instead of generalizing RPQs, one can consider "slightly non-sticky" rulesets. Consider an extension of the above ruleset with the following projections.

$$\begin{aligned}
\mathtt{IncX}(z,z',u,v,t) &\rightarrow \mathtt{IncXBin}(z,z')\\
\mathtt{DecX}(z,z',u,v,t) &\rightarrow \mathtt{DecXBin}(z,z')\\
\mathtt{IncY}(z,z',u,v,t) &\rightarrow \mathtt{IncYBin}(z,z')\\
\mathtt{DecY}(z,z',u,v,t) &\rightarrow \mathtt{DecYBin}(z,z')\\
\mathtt{XZero}(z,z,x) &\rightarrow \mathtt{XZeroBin}(z,z)\\
\mathtt{YZero}(z,z,y) &\rightarrow \mathtt{YZeroBin}(z,z)
\end{aligned}$$

Note that such expanded ruleset would as well admit undecidable RPQ entailment, and that the projections are not allowing for further recursion. Therefore the above ruleset can be viewed as essentially sticky, just followed by one single projection step.

6 Conclusion

In this paper, we reviewed established existential rules fragments with decidable CQ answering, asking if the decidability carries over to RPQs. We recalled that for the very comprehensive $\mathtt{fcs}$ class of rulesets, the decidability for RPQs and even much more expressive query languages follows from recent results. Thus focusing on the $\mathtt{fus}$ class of

rulesets, we established that they do not allow for decidable RPQ answering in general, due to the insight that – unlike `fcs` rulesets – `fus` rulesets allow for the creation of grid-like universal models, which then can be used as a "two-counter state space", in which accepting runs of two-counter machines take the shape of regular paths. On the other hand, we showed that 2RPQ answering over sticky rulesets is decidable thanks to the reducibility to a finitely RPQ-controllable querying problem. This decidability result is rather brittle and crucially depends on (1) the restriction of path expressions to binary predicates and (2) the inability to freely project away variables in sticky rulesets. For this reason, a setting where RPQs are slightly liberalized leads to undecidability again.

There are several obvious questions left for future work:

- What is the precise complexity of 2RPQ answering over sticky rulesets? Recall that, as our decidability argument is based on finite controllability, the generic decision algorithm ensuing from that does not come with any immediate upper complexity bound.
- Does the problem remain decidable for sticky rulesets when progressing to CRPQs or C2RPQs? We do not believe that a minor extension of our current proof would be sufficient to positively settle this question, as we are currently lacking methods of coping with variable joins in C(2)RPQs (under such circumstances, establishing finite controllability via stellar types fails). However, we think that some of the tools we used in this paper might come in handy when tackling that extended case.
- What is the decidability status for other (non-`fcs`) syntactically defined `fus` fragments, such as *sticky-join rulesets* (Calì, Gottlob, and Pieris 2012)?
- Is it possible and/or reasonable to establish a new class of rulesets, based on the rewritability of (C2)RPQs rather than CQs?

Acknowledgements

Work supported by the European Research Council (ERC) Consolidator Grant 771779 (DeciGUT).

References

Abiteboul, S.; Hull, R.; and Vianu, V. 1995. *Foundations of Databases*. Addison-Wesley.

Baget, J.-F.; Leclère, M.; Mugnier, M.-L.; and Salvat, E. 2011. On rules with existential variables: Walking the decidability line. *Artificial Intelligence* 175(9):1620–1654.

Baget, J.-F.; Bienvenu, M.; Mugnier, M.-L.; and Thomazo, M. 2017. Answering conjunctive regular path queries over guarded existential rules. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017)*, 793–799. ijcai.org.

Beeri, C., and Vardi, M. Y. 1984. A proof procedure for data dependencies. *Journal of the ACM* 31(4):718–741.

Calì, A.; Gottlob, G.; and Kifer, M. 2013. Taming the infinite chase: Query answering under expressive relational constraints. *Journal of Artificial Intelligence Research* 48:115–174.

Calì, A.; Gottlob, G.; and Lukasiewicz, T. 2009. Datalog$^{\pm}$: a unified approach to ontologies and integrity constraints. In *Proceedings of the 12th International Conference on Database Theory (ICDT 2009)*, volume 361, 14–30. ACM.

Calì, A.; Gottlob, G.; and Pieris, A. 2010. Advanced processing for ontological queries. *Proceedings of the VLDB Endowment* 3(1–2):554–565.

Calì, A.; Gottlob, G.; and Pieris, A. 2012. Towards more expressive ontology languages: The query answering problem. *Artificial Intelligence* 193:87–128.

Calvanese, D.; Giacomo, G. D.; Lenzerini, M.; and Vardi, M. Y. 2003. Reasoning on regular path queries. *SIGMOD Record* 32(4):83–92.

Chandra, A. K., and Merlin, P. M. 1977. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing (STOC 1977)*, 77–90. ACM.

Chandra, A. K.; Lewis, H. R.; and Makowsky, J. A. 1981. Embedded implicational dependencies and their inference problem. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing (STOC 1981)*, 342–354. ACM.

Deutsch, A.; Nash, A.; and Remmel, J. B. 2008. The chase revisited. In *Proceedings of the 27th Symposium on Principles of Database Systems (PODS 2008)*, 149–158. ACM.

Feller, T.; Lyon, T. S.; Ostropolski-Nalewaja, P.; and Rudolph, S. 2023. Finite-cliquewidth sets of existential rules: Toward a general criterion for decidable yet highly expressive querying. In *Proceedings of the 26th International Conference on Database Theory (ICDT 2023)*, volume 255 of *LIPIcs*, 18:1–18:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Florescu, D.; Levy, A. Y.; and Suciu, D. 1998. Query containment for conjunctive queries with regular expressions. In *Proceedings of the 17th Symposium on Principles of Database Systems (PODS 1998)*, 139–148. ACM.

Gogacz, T., and Marcinkowski, J. 2017. Converging to the chase – a tool for finite controllability. *Journal of Computer and System Sciences* 83(1):180–206.

Gottlob, G. 2009. Datalog+/-: A unified approach to ontologies and integrity constraints. In *Proceedings of the 17th Italian Symposium on Advanced Database Systems (SEBD 2009)*, 5–6. Edizioni Seneca.

Grau, B. C.; Horrocks, I.; Krötzsch, M.; Kupke, C.; Magka, D.; Motik, B.; and Wang, Z. 2013. Acyclicity notions for existential rules and their application to query answering in ontologies. *Journal of Artificial Intelligence Research* 47:741–808.

Krötzsch, M., and Rudolph, S. 2011. Extending decidable existential rules by joining acyclicity and guardedness. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, 963–968. IJCAI/AAAI.

König, M.; Leclère, M.; Mugnier, M. L.; and Thomazo, M. 2015. Sound, complete and minimal UCQ-rewriting for existential rules. *Semantic Web* 6(5):451–475.

Minsky, M. L. 1967. *Computation: Finite and Infinite Machines*. USA: Prentice-Hall, Inc.

Ostropolski-Nalewaja, P., and Rudolph, S. 2024. The sticky path to expressive querying: Decidability of navigational queries under existential rules. *CoRR* abs/2407.14384.

Ostropolski-Nalewaja, P.; Marcinkowski, J.; Carral, D.; and Rudolph, S. 2022. A journey to the frontiers of query rewritability. In *Proceedings of the 41st Symposium on Principles of Database Systems (PODS 2022)*, 359–367. ACM.

Rudolph, S., and Krötzsch, M. 2013. Flag & check: data access with monadically defined queries. In *Proceedings of the 32nd Symposium on Principles of Database Systems (PODS 2013)*, 151–162. ACM.

Rudolph, S. 2014. The two views on ontological query answering. In *Proceedings of the 8th Alberto Mendelzon Workshop on Foundations of Data Management*, volume 1189 of *CEUR Workshop Proceedings*. CEUR-WS.org.