Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning
Main Track - Short paper

Sticky Existential Rules and Disjunction are Incompatible*

Michael Morak
University of Klagenfurt, Austria
michael.morak@aau.at

Abstract

Stickiness is one of the well-known properties in the literature that guarantees decidability of query answering under sets of existential rules, that is, Datalog rules extended with existential quantification in rule heads. In this note, we investigate whether this remains true in the case when rule heads are allowed to be disjunctive. We answer this question in the negative, providing a strong undecidability result that shows that the concept of stickiness cannot be extended to disjunctive existential rules, even when considering only fixed atomic queries and a fixed set of rules. This provides evidence that, in order to keep query answering decidable, a stronger property than stickiness is needed in the disjunctive case.

1 Introduction

Rule-based languages form the core of several applications in the areas of databases and knowledge representation. In the setting of ontological query answering, a traditional database D is enriched with a set of rules Σ that represent an ontology (that is, knowledge about the world). Conjunctive queries (CQs) are then answered w.r.t. the combined knowledge represented by the database and the ontology, that is, $D \cup \Sigma$. In this paper, we consider the language of *existential rules*, also known as *Datalog*$^{\pm}$ or *tuple-generating dependencies (TGDs)*, which are Datalog rules extended with existential quantifiers in rule heads.

A TGD is a formula $\forall X\, \varphi(X) \rightarrow \exists Y\, \psi(X, Y)$, where φ and ψ are conjunctions of atoms. In general, CQ answering w.r.t. a database enriched with TGDs is undecidable. A wide range of syntactic restrictions have been introduced in the literature that guarantee decidability. Notably, languages based on the concepts of guardedness (Calì, Gottlob, and Kifer 2013), stickiness (Calì, Gottlob, and Pieris 2012), and weak-acyclicity (Fagin et al. 2005) have proven successful in this regard.

In this paper we focus on the extension of TGDs with disjunction (that is, where the formula ψ above is a disjunction of conjunctions). These rules are then called *disjunctive existential rules* or *disjunctive tuple-generating dependencies (DTGDs)*. A thorough investigation of the complexity of CQ answering has been done for guarded classes of DTGDs (Bourhis et al. 2016) and for acyclicity-based notions (Carral, Dragoste, and Krötzsch 2017). However, for the decidable class of sticky TGDs (and its extensions), such an investigation has not yet been presented. We aim to close this gap in the present work.

In particular, we extend the class of sticky sets of TGDs with disjunction, and give the relevant definitions to establish the corresponding class of sticky sets of DTGDs. We then show that even for schemas with a maximum arity of two and atomic queries, the CQ answering problem becomes undecidable for sticky sets of DTGDs. Undecidability even holds in the data complexity, where the set of DTGDs and the CQ are fixed. Our proof technique reveals that a useful extension of stickiness with disjunction seems impossible.

2 Preliminaries

Technical Definitions. We define the following pairwise disjoint (infinite) sets: a set $\mathbf{C}$ of *constants*, a set $\mathbf{N}$ of *labeled nulls*, and a set $\mathbf{V}$ of regular *variables*. We denote by $\mathbf{X}$ sequences (or sets) of variables $X_1, \ldots, X_k$, $k \geq 0$. A *schema* $\mathcal{R}$ is a set of *relations* (or *predicates*), each of which has an associated *arity*. The arity of a relation r is denoted $arity(r)$, and the maximum arity over all predicates of schema $\mathcal{R}$ is denoted $arity(\mathcal{R})$. A *position* in schema $\mathcal{R}$ is a pair (r, i), where r is a relation in $\mathcal{R}$, and i a number such that $1 \leq i \leq arity(r)$. We will denote positions using the notation $r[i]$. A *term* t is a constant ($t \in \mathbf{C}$), labeled null ($t \in \mathbf{N}$), or variable ($t \in \mathbf{V}$). An *atomic formula* (also simply called an *atom*) has the form $p(t_1, \ldots, t_n)$, where p is an n-ary predicate, and $t_1, \ldots, t_n$ are terms. For an atom $\underline{a}$, we denote as $dom(\underline{a})$ and $var(\underline{a})$ the set of terms and the set of variables occurring in $\underline{a}$, respectively. These notations naturally extend to sets of atoms. For convenience, we will treat conjunctions of atoms as sets of atoms. An *instance* I over a schema $\mathcal{R}$ is a (possibly infinite) set of atoms of the form $p(\mathbf{t})$, where $\mathbf{t}$ is a tuple of constants and labeled nulls, that is, $\mathbf{t} \subseteq (\mathbf{C} \cup \mathbf{N})^{|\mathbf{t}|}$. A *database* D is a finite instance that does not contain null values. When an instance I is treated as a logical formula, it is the conjunction $\exists \mathbf{X} \bigwedge_{\underline{a} \in I} \underline{a}$, where $\mathbf{X}$ contains a variable X_z for each labeled null z in I.

Disjunctive Tuple-Generating Dependencies (DTGDs). A DTGD σ is a first-order formula of the form $\forall \mathbf{X} \left(\varphi(\mathbf{X}) \rightarrow \bigvee_{i=1}^{n} \exists \mathbf{Y}_i\, \psi_i(\mathbf{X}, \mathbf{Y}_i)\right)$, where $n \geq 1$, $\mathbf{X} \cup \mathbf{Y}_1 \cup \ldots \cup \mathbf{Y}_n \subset \mathbf{V}$, and $\varphi, \psi_1, \ldots, \psi_n$ are conjunctions

*This note is an excerpt of the author's doctoral thesis (Morak 2014), which has hitherto not been published at a scientific venue.

of atoms (possibly containing constants of $\mathbf{C}$). The formula φ is called the *body* of σ, denoted $body(\sigma)$, while $\bigvee_{i=1}^{n} \psi_i$ is the *head* of σ, denoted $head(\sigma)$. The set of variables $var(body(\sigma)) \cap var(head(\sigma)) \subseteq \mathbf{X}$, that is, the variables of $\mathbf{X}$ which appear both in the body and in the head of σ, is known as the *frontier* of σ, and is denoted as $fr(\sigma)$.

If $n = 1$, then σ is called *tuple-generating dependency (TGD)*. We will sometimes refer to sets of (D)TGDs as a *theory* or as a set of *(disjunctive) existential rules* or simply *rules*. The *schema* of a set Σ of DTGDs, denoted $sch(\Sigma)$, is the set of all predicates occurring in Σ. In the rest of the paper, for brevity, we will omit the universal quantifiers in front of DTGDs, and implicitly assume such a quantification. We will also use the comma (instead of $\wedge$) for conjoining atoms in the body and in the head of a DTGD. An instance I satisfies a DTGD σ, written $I \models \sigma$, if the following holds: whenever there exists a homomorphism h such that $h(\varphi(\mathbf{X})) \subseteq I$, then there exists $i \in [n]$ and $h' \supseteq h$ such that $h'(\psi_i(\mathbf{X}, \mathbf{Y}_i)) \subseteq I$; I satisfies a set Σ of DTGDs, denoted $I \models \Sigma$, if $I \models \sigma$, for each $\sigma \in \Sigma$. Sometimes, we treat a set Σ of DTGDs as the logical formula $(\bigwedge_{\sigma \in \Sigma} \sigma)$.

Query Answering. A *conjunctive query (CQ)* q is a positive existential first-order formula $\exists \mathbf{Y}\, \varphi(\mathbf{X}, \mathbf{Y})$, where φ is a conjunction of atoms with variables from $\mathbf{X} \cup \mathbf{Y} \subset \mathbf{V}$, and possibly constants of $\mathbf{C}$. A 0-ary CQ is called *Boolean CQ (BCQ)*. *Atomic queries*, denoted CQ_1, are CQs with a single atom. Unions of CQs (UCQs) are disjunctions of CQs.

The *models* of a database D and a theory Σ, denoted $models(D, \Sigma)$, is the set of instances $\{I \mid I \supseteq D \text{ and } I \models \Sigma\}$. The *answer* to a CQ q w.r.t. D and Σ, denoted $ans(q, D, \Sigma)$, is the set of tuples of constants $\bigcap_{I \in models(D,\Sigma)} \{\mathbf{t} \mid \mathbf{t} \in q(I)\}$. The answer to a BCQ q w.r.t. D and Σ is *positive*, denoted $D \cup \Sigma \models q$, if $ans(q, D, \Sigma) \neq \emptyset$. Our central problem, CQ-ANSWERING, is defined as follows: given a BCQ[1] q, a database D, and a set Σ of DTGDs, decide whether it has a positive answer. Following the taxonomy used by (Vardi 1982), the *data complexity* of the above problems is calculated taking only the database as input. For the *combined complexity*, the query and set of DTGDs count as part of the input as well.

Disjunctive Chase. We employ the *disjunctive chase* introduced in (Deutsch and Tannen 2005), i.e. an extension of the well-known chase procedure where each step "branches" out several instances, each satisfying one of the disjuncts in the DTGD that is applied. Therefore, the result of the disjunctive chase is, in general, a set of instances.

Definition 2.1 (DTGD Chase Rule)**.** *Consider an instance I, and a DTGD σ of the form $\varphi(\mathbf{X}) \rightarrow \bigvee_{i=1}^{n} \exists \mathbf{Y}\, \psi_i(\mathbf{X}, \mathbf{Y})$. σ is* applicable *to I if there exists a homomorphism h such that $h(\varphi(\mathbf{X})) \subseteq I$, and the result of applying σ to I with h is the set $\{I_1, \ldots, I_n\}$, where $I_i = I \cup h'(\psi_i(\mathbf{X}, \mathbf{Y}))$, for each $i \in [n]$, and $h' \supseteq h$ is such that $h'(Y)$ is a "fresh" null not occurring in I, and following lexicographically all those in I, for each $Y \in \mathbf{Y}$. For such an application, which defines a single* chase step*, we write $I\langle \sigma, h \rangle \{I_1, \ldots, I_n\}$.*

[1]We focus on BCQs because it is well-known they are LOGSPACE-equivalent to CQs for CQ-ANSWERING.

A *disjunctive chase tree* of a database D and a set Σ of DTGDs is a (possibly infinite) tree such that the root is D, and for every node I, assuming that $\{I_1, \ldots, I_n\}$ are the children of I, there exists $\sigma \in \Sigma$ and a homomorphism h such that $I\langle \sigma, h \rangle \{I_1, \ldots, I_n\}$. The disjunctive chase algorithm for D and Σ consists of an exhaustive application of DTGD chase steps in a *fair fashion*, that is, every DTGD that can be applied is applied via at least one homomorphism in each path of the chase tree. This leads to a disjunctive chase tree T of D and Σ. Let $chase(D, \Sigma)$ be the set $\{I \mid I$ is the set of all atoms occurring in a branch of $T\}$. Notice that each such branch of T (i.e. a path from the root downwards) is well-defined as the least fixpoint of a monotonic operator, even though such paths can of course be infinite. By construction, each instance in $chase(D, \Sigma)$ is a model of D and Σ. Interestingly, $chase(D, \Sigma)$ is a *universal model set* of D and Σ, that is, for each $I \in models(D, \Sigma)$, there exists $J \in chase(D, \Sigma)$ and a homomorphism h_J such that $h_J(J) \subseteq I$ (Deutsch, Nash, and Remmel 2008). This implies that the chase is the right tool for query answering.

3 Sticky Sets of DTGDs

In this section, we extend the syntactic restriction of sticky sets of TGDs to DTGDs. Our extension also works for the other sticky-based restrictions. However, as we will see, the combination of the limited joins allowed by sticky theories and disjunction lead to undecidability of CQ-ANSWERING.

3.1 Extending Sticky Sets of TGDs to DTGDs

Stickiness is determined by employing a marking algorithm called SMarking (Calì, Gottlob, and Pieris 2012). The formulation of the SMarking procedure given there is easy to extend to DTGDs, as done below. For a DTGD σ, let $head(\sigma)$ denote all the atoms occurring in the head of σ, independent of any disjunctions σ may contain.

Definition 3.1. SMarking *takes a set Σ of DTGDs as input and returns a set of marked DTGDs where for each marked DTGD, every body variable is either marked, or not. The procedure works in two steps, where the second step is exhaustively applied until a fixpoint is reached.*

Initial Marking Step *For each DTGD $\sigma \in \Sigma$ and each variable V in $body(\sigma)$, if there exists an atom $\underline{a}$ in $head(\sigma)$ that does not contain V, mark V in σ.*

Propagation Step *For each TGD $\sigma \in \Sigma$ and variable V in $body(\sigma)$, if there is a $\sigma' \in \Sigma$ such that $head(\sigma)$ and $body(\sigma')$ contain a relation r and the positions where V appears in r in $head(\sigma)$ coincide exclusively with positions of marked variables in $body(\sigma')$, then mark V in σ.*

Analogously to the definition of sticky sets of TGDs, we can now give the same definition of DTGDs.

Definition 3.2. *A set Σ of DTGDs is* sticky *if there is no DTGD $\sigma \in \text{SMarking}(\Sigma)$ such that a marked variable occurs in $body(\sigma)$ more than once.*

The extension of stickiness to DTGDs is purposefully narrow: it does not distinguish between disjunction and conjunction in rule heads. As we will later see, this narrow ex-

tension already leads to undecidability, even though the extended SMarking procedure guarantees that the sticky property (Calì, Gottlob, and Pieris 2011) is valid in each model of the disjunctive chase w.r.t. sticky sets of DTGDs.

3.2 CQ-ANSWERING under Sticky Sets of DTGDs

We investigate the problems of CQ-ANSWERING, as well as CQ_1-ANSWERING. However, even in the case of sticky sets of (non-disjunctive) TGDs, we can see that in fact both of these problems are equivalent. We will later see that results also hold for UCQs.

To make our investigation easier, it is not difficult to see from the definition of sticky sets of DTGDs that CQs can be encoded as rules, where the CQ becomes the rule body, and a single, fresh propositional atom becomes the head. We can then simply ask an atomic query consisting of just this propositional atom. This observation immediately leads to the following result:

Proposition 3.3. *Under sticky sets of DTGDs, the* CQ-ANSWERING *and* CQ_1-ANSWERING *problems are* LOGSPACE*-equivalent in combined and data complexity.*

We can thus restrict our attention to arbitrary CQs, and all the undecidability results obtained for the CQ-ANSWERING problem also apply to CQ_1-ANSWERING.

3.3 Undecidability of CQ-ANSWERING

We will now establish that the combination of stickiness and non-determinism leads to undecidability. In fact the problem lies in the combination of simple joins like cross products, and disjunction. Allowed to interact, as is the case in sticky sets of DTGDs, these two features lead to undecidability of query answering. We thus do not even need the full expressive power that stickiness provides. In fact it is sufficient to take a set of *disjunctive inclusion dependencies (DIDs)*—that is, DTGDs that only allow one body atom with non-repeated variables, and hence are clearly sticky—and allow a single, binary Cartesian product to appear. We thus define the class of $\times$-DIDs as follows:

Definition 3.4. *A set Σ of DTGDs is a set of* $\times$-DIDs*, if Σ can be partitioned into two sets Σ_{DID} and $\Sigma_\times$, such that Σ_{DID} is a set of DIDs, and $\Sigma_\times$ is a set of cartesian products, that is, TGDs of the form $r(\mathbf{X}), s(\mathbf{Y}) \rightarrow rs(\mathbf{X}', \mathbf{Y}')$, where r, s and rs are relations and $\mathbf{X}$ and $\mathbf{Y}$ are disjoint sets of variables, with $\mathbf{X}' \subseteq \mathbf{X}$ and $\mathbf{Y}' \subseteq \mathbf{Y}$.*

With $\times$-DIDs it is possible to encode UCQs as CQs:

Proposition 3.5. *Consider a database D, a set Σ of $\times$-DIDs, and a UCQ Q over schema $\mathcal{R}$. A database D', a set Σ' of $\times$-DIDs, and a CQ q over a schema $\mathcal{R}'$, with $arity(\mathcal{R}) = arity(\mathcal{R}')$ can be constructed in polynomial time such that $D \cup \Sigma \models Q$ if and only if $D' \cup \Sigma' \models q$.*

Proof (Idea). This can be achieved by making use of a hard-coded ternary OR relation in the database, keeping track of the atoms derived by the chase (by marking them with a special constant via a cross-product rule), and having dummy atoms to satisfy all UCQ disjuncts. The final CQ is constructed so that it ensures, using the OR relation, that at least one UCQ disjunct is satisfied by non-dummy atoms. □

Since $\times$-DIDs are sticky, the above proposition also holds for sticky sets of DTGDs:

Corollary 3.6. *Under sticky sets of DTGDs,* CQ-ANSWERING *and* UCQ-ANSWERING *are* PTIME*-equivalent in the combined and data complexity.*

Hence, sticky sets of DTGDs are expressive enough that the query language no longer plays a role, and any undecidability result shown for answering UCQs under sticky sets of DTGDs also holds for CQs and atomic queries.

With the above definitions and observations in place, we can now proceed to show our first undecidability result. In order to show undecidability of CQ-ANSWERING under $\times$-DIDs, we can simulate a general, deterministic Turing machine and thus provide a reduction from the halting problem.

Theorem 3.7. CQ-ANSWERING *under $\times$-DIDs over a schema $\mathcal{R}$ is undecidable, even when the arity of $\mathcal{R}$ is bounded by the constant* 2*.*

Proof. We will show this by reduction from the co-halting problem. Let $M = \langle S, \Lambda, \delta, s_0 \rangle$ be a deterministic Turing machine, where S is a set of states with $s_0, s_h \in S$ being the starting and halting state, $\Lambda = \{1, 0, \sqcup\}$ it's alphabet with $\sqcup$ the blank symbol and $\delta : S \times \Lambda \rightarrow S \times \Lambda \times \{-1, +1, 0\}$ the transition function. We assume that there is only one halting state s_h and that when the machine halts, it accepts, and has a do-nothing self-loop on s_h. We also assume that M takes no input, that is, the work tape contains only blanks, and that the head is, for technical reasons, at the second position of the tape and never moves further left than that. We will construct a database D, a set of $\times$-DIDs Σ and a CQ q over a schema $\mathcal{R}$, such that $D \cup \Sigma \models q$ iff M does not halt.

The idea of the construction is as follows: we will construct an infinite grid, where each row represents a configuration of the Turing machine and each column a position on the tape. Using a disjunctive rule, we will guess, for each position and configuration, which symbol there is on the tape. With a CQ q, we will then check that at least one model of the chase represents a valid computation of M and that this computation halts. To this end, let $S' = (S \times \Lambda) \cup \Lambda$.

The Schema $\mathcal{R}$. $\mathcal{R}$ contains the following predicates:

- $null(\cdot)$: collects all null values.
- $init(\cdot, \cdot)$: the initial two configurations and cells.
- $conf(\cdot, \cdot)$: two subsequent configurations.
- $cell(\cdot, \cdot)$: two subsequent tape cells.
- $confcell(\cdot, \cdot)$: a configuration and a tape cell together.
- $s(\cdot, \cdot)$ for each $s \in S'$: a combination of a configuration and a tape cell is labelled with s.

The Database D. In our database D we will simply store two constants a, b in a relation $init(a, b)$.

The Set of $\times$-DIDs Σ. First, we construct our grid:

- $init(X, Y) \rightarrow conf(X, Y)$
- $init(X, Y) \rightarrow cell(X, Y)$
- $conf(X, Y) \rightarrow \exists Z\ conf(Y, Z), null(Z)$
- $cell(X, Y) \rightarrow \exists Z\ cell(Y, Z), null(Z)$

- $conf(X_1, Y_1), cell(X_2, Y_2) \rightarrow confcell(X_1, X_2)$

We guess which element from S' is true at each $confcell$:

- $confcell(X_1, X_2) \rightarrow \bigvee_{s \in S'} s(X_1, X_2)$

The CQ q. In order to construct q, we will construct a union (i.e. disjunction) of CQs (UCQ) Q, which makes it easier to grasp what conditions the query checks. In order to check the validity of the transitions, and model the inertia rules, notice that we can represent transitions in δ as a six-tuple $abc \rightarrow a'b'c'$ mapping a 3-tuple of symbols from S' (i.e. including the state) in one configuration to a 3-tuple of symbols in the next configuration, where thus $a, c \in \Lambda$, $b \in S \times \Lambda$ and $a', b', c' \in S'$, obviously with only one of a', b', c' from $S \times \Lambda$. With this observation in place, we now give the formal definition of Q, where each of the following four parts represents a set of disjuncts that check a certain condition. For brevity, we may use disjunction inside a conjunction. This can be converted to a UCQ by unfolding.

1. $Q_{initial}$ is made up of three parts, checking the first, second, and subsequent positions of the tape respectively:
   - $\exists X \exists Y\ init(X, Y) \wedge (0(X, Y) \vee 1(X, Y))$
   - $\bigvee_{[s,\lambda] \in (S' \setminus \{[s_0, \sqcup]\})} \exists X \exists Y\ init(X, Y) \wedge [s, \lambda](X, Y)$
   - $\exists X \exists Y \exists Z\ init(X, Y) \wedge null(Z) \wedge (0(Y, Z) \vee 1(Y, Z))$
2. Q_{trans} is made up of multiple disjuncts as follows. For each of the possible rewriting rules $abc \rightarrow a'b'c'$ that do not occur in δ, we add the following disjunct, that will thus make the query true if the transition function has been violated.

$$\exists X \exists Y \exists X_1 \exists X_2 \exists X_3\ conf(X, Y) \wedge cell(X_1, X_2) \wedge cell(X_2, X_3) \wedge a(X, X_1) \wedge b(X, X_2) \wedge c(X, X_3) \wedge a'(Y, X_1) \wedge b'(Y, X_2) \wedge c'(Y, X_3)$$

3. Q_{inert} checks for a triple of symbols from Λ on the tape and checks that the next configuration contains the same ones. It is thus similar in structure to Q_{trans}. For each triple abc of constants from $\Lambda \times \Lambda \times \Lambda$, and constant $d \in \Lambda \setminus \{b\}$, we add the following disjunct:

$$\exists X \exists Y \exists X_1 \exists X_2 \exists X_3\ conf(X, Y) \wedge cell(X_1, X_2) \wedge cell(X_2, X_3) \wedge a(X, X_1) \wedge b(X, X_2) \wedge c(X, X_3) \wedge d(Y, X_2)$$

4. $Q_{halt} \equiv$

$$\exists X \exists Y [s_h, 0](X, Y) \vee [s_h, 1](X, Y) \vee [s_h, \sqcup](X, Y)$$

Correctness. The above construction is correct. Assume that $D \cup \Sigma$ does not entail Q. Then this means that there is an instance $I \in chase(D, \Sigma)$, such that none of the disjuncts of Q are true in I. But because I, by construction and non-entailment of $Q_{initial}$, Q_{trans} and Q_{inert}, represents a valid computation of M, and I does not entail Q_{halt}, this means that there must be an infinite computation of M that does not reach the halting state. Note that as M is deterministic, there is exactly one such I. Conversely, assume that the query Q holds in every model of $D \cup \Sigma$. This means that every model either encodes an invalid computation, or a valid computation that reaches the halting state, by construction of Q_{halt}. We thus have as desired that $D \cup \Sigma \models Q$ if and only if M does not halt. Also note that the above construction uses only $\times$-DIDs, and the schema has at most arity 2.

However, the proof yields a union of CQs instead of our desired CQ. It remains to show that the result also holds for CQs, but this follows immediately from Proposition 3.5. □

In the proof, we show that the unconstrained combination of disjunction and cross products is a definite cause for undecidability. From the construction, it seems impossible to meaningfully combine stickiness with disjunction, since the simplest combination (DIDs plus a single cross-product, maximum arity 2) already induces undecidability. The next result establishes undecidability also in the data complexity, at the cost of increasing the maximum arity to at most 3.

Theorem 3.8. CQ-ANSWERING *of fixed CQs under fixed sets of $\times$-DIDs over a schema $\mathcal{R}$ is undecidable, even when the arity of $\mathcal{R}$ is bounded by the constant* 3*.*

Proof (Idea). The proof is an adaptation of the previous one, where data about the Turing machine is now stored in relations in the database (i.e. possible symbols, illegal transitions according to the transition function δ). For each $confcell$, we can then guess a symbol and store it in a three-ary relation. Finally, in the query, we can check for guesses that represent illegal transitions. □

We thus have shown, as desired, that even for fixed sets of $\times$-DIDs and fixed queries, that is, in the data complexity, the CQ-ANSWERING problem is still undecidable. The following corollary follows immediately from the above results, and the fact that sets of $\times$-DIDs are sticky sets of DTGDs:

Corollary 3.9. CQ-ANSWERING *and* CQ_1-ANSWERING *under sticky sets of DTGDs are undecidable, even if we consider only the data complexity.*

These results provides further evidence that decidability of CQ-ANSWERING under disjunctive extensions of sticky TGDs seemingly cannot be meaningfully achieved: since any set of inclusion dependencies is sticky, and sticky TGDs allow for the expression of cross products, any extension of sticky TGDs with disjunction should arguably at least contain the class of $\times$-DIDs, and would thus inherit the undecidablility of CQ-ANSWERING.

4 Conclusions

In this paper, we have investigated the impact of allowing disjunctions in rule heads of sticky existential rules, or TGDs, when considering the query answering problem. We established an undecidability result for the query answering problem, even when restricted to atomic queries. In fact, for general queries, this result holds even when only unary and binary predicates are allowed in the schema. Undecidability even holds in the data complexity. Our results imply that it does not seem possible to meaningfully combine the concept of stickiness with disjunction.

References

Bourhis, P.; Manna, M.; Morak, M.; and Pieris, A. 2016. Guarded-based disjunctive tuple-generating dependencies. *ACM Trans. Database Syst.* 41(4):27:1–27:45.

Calì, A.; Gottlob, G.; and Kifer, M. 2013. Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. *J. Artif. Intell. Res.* 48:115–174.

Calì, A.; Gottlob, G.; and Pieris, A. 2011. New Expressive Languages for Ontological Query Answering. In *Proc. AAAI*. AAAI Press.

Calì, A.; Gottlob, G.; and Pieris, A. 2012. Towards More Expressive Ontology Languages: The Query Answering Problem. *Artif. Intell.* 193:87–128.

Carral, D.; Dragoste, I.; and Krötzsch, M. 2017. Restricted chase (non)termination for existential rules with disjunctions. In Sierra, C., ed., *Proc. IJCAI*, 922–928. ijcai.org.

Deutsch, A., and Tannen, V. 2005. XML Queries and Constraints, Containment and Reformulation. *Theor. Comput. Sci.* 336(1):57–87.

Deutsch, A.; Nash, A.; and Remmel, J. B. 2008. The Chase Revisited. In *Proc. PODS*, 149–158. ACM.

Fagin, R.; Kolaitis, P. G.; Miller, R. J.; and Popa, L. 2005. Data Exchange: Semantics and Query Answering. *Theor. Comput. Sci.* 336(1):89–124.

Morak, M. 2014. *The Impact of Disjunction on Reasoning Under Existential Rules*. Ph.D. Dissertation, University of Oxford, Oxford, Oxfordshire, UK.

Vardi, M. Y. 1982. The Complexity of Relational Query Languages (Extended Abstract). In *Proc. STOC*, 137–146. ACM.