Reasoning about disclosure in data integration in the presence of source constraints

Michael Benedikt, Pierre Bourhis, Louis Jachiet, Michaël Thomazo

Reasoning about disclosure in data integration in the presence of source constraints

Michael Benedikt[1] , **Pierre Bourhis**[2] , **Louis Jachiet**[2] and **Michaël Thomazo**[3]
[1]University of Oxford
[2]CNRS CRIStAL, Université Lille, Inria Lille
[3]Inria, DI ENS, ENS, CNRS, PSL University
{pierre.bourhis,louis.jachiet}@univ-lille.fr, michael.thomazo@inria.fr, michael.benedikt@cs.ox.ac.uk

Abstract

Data integration systems allow users to access data sitting in multiple sources by means of queries over a global schema, related to the sources via mappings. Datasources often contain sensitive information, and thus an analysis is needed to verify that a schema satisfies a privacy policy, given as a set of queries whose answers should not be accessible to users. Such an analysis should take into account not only knowledge that an attacker may have about the mappings, but also what they may know about the semantics of the sources. In this paper, we show that *source constraints* can have a dramatic impact on disclosure analysis. We study the problem of determining whether a given data integration system discloses a source query to an attacker in the presence of constraints, providing both lower and upper bounds on source-aware disclosure analysis.

1 Introduction

In data integration, users are shielded from the heterogeneity of multiple datasources by querying via a *global schema*, which provides a unified vocabulary. The relationship between sources and the user-facing schema are specified declaratively via *mapping rules*. In data integration systems based on knowledge representation techniques, users pose queries against the global schema, and these queries are answered using data in the sources and background knowledge. The computation of the answers involves reasoning based on the query, the mappings, and any additional semantic information that is known on the global schema.

Data integration brings with it the danger of disclosing information that data owners wish to keep confidential. In declarative data integration, detection of privacy violations is complex: although explicit access to source information may be masked by the global schema, an attacker can infer source facts via reasoning with schema and mapping information.

Example 1. *We consider an information integration setting for a hospital, which internally stores the following data:*

Predicate	*Meaning*
$\mathsf{IsOpen}(b,t)$	*building b is open on date t*
$\mathsf{PatBdlg}(p,b)$	*patient p is present in building b*
$\mathsf{PatSpec}(p,s)$	*patient p was treated for specialty s*
$\mathsf{PatDoc}(p,d)$	*patient p was treated by doctor d*
$\mathsf{DocBldg}(d,b)$	*doctor d is associated with building b*
$\mathsf{DocSpec}(d,s)$	*doctor d is associated with specialty s*

The hospital publishes the following data: $\mathsf{OpenHours}(b,t)$ *giving opening times t for building b,* $\mathsf{VisitingHours}(p,t)$ *giving times t when a given patient p can be visited, and* $\mathsf{DocList}(d,s,b)$ *listing the doctors d with their specialty s and their building b. Formally the data being exposed is given by the following mappings:*

$$\begin{aligned} \mathsf{IsOpen}(b,t) &\rightarrow \mathsf{OpenHours}(b,t) \\ \mathsf{PatBdlg}(p,b) \wedge \mathsf{IsOpen}(b,t) &\rightarrow \mathsf{VisitingHours}(p,t) \\ \mathsf{DocSpec}(d,s) \wedge \mathsf{DocBldg}(d,b) &\rightarrow \mathsf{DocList}(d,s,b) \end{aligned}$$

Prior work [Benedikt *et al.*, 2018] has studied disclosure in knowledge-based data integration, with an emphasis on the role of semantic information on the *global schema* – in the form of ontological rules that relate the global schema vocabulary. The presence of an ontology can assist in privacy, since distinctions in the source data may become indistinguishable in the ontology. More dangerous from the point of view of protecting information is *semantic information about sources*. For example, the sources in a data integration setting will generally overlap: that is, they will satisfy *referential integrity constraints*, saying that data items in one source link to items in another source. Such constraints should be assumed as public knowledge, and with that knowledge the attacker may be able to infer information that was intended to be secret.

Example 2. *Continuing Example 1, suppose that we know that each patient has a doctor specialized in their condition, which can be formalized as:*

$$\mathsf{PatDoc}(p,d) \rightarrow \exists s\ \mathsf{PatSpec}(p,s) \wedge \mathsf{DocSpec}(d,s)$$

And that we also know that when a patient is in a building, they must have a doctor there:

$$\mathsf{PatBdlg}(p,b) \rightarrow \exists d\ \mathsf{PatDoc}(p,d) \wedge \mathsf{DocBldg}(d,b)$$

Due to the presence of these constraints, there can be a disclosure of the relationship of patient to speciality

$\mathsf{PatSpec}(p, s)$. *Indeed, an attacker can see the* VisitingHours *for* p*, and from this, along with* OpenHours*, they can sometimes infer the building* b *where* p *has visited (e.g. if* b *has a unique set of open hours). From this they may be able to infer, using* DocList*, the specialty that* p *has been treated for – for example, if all the doctors in* b *share a specialty.*

In this work, we perform a detailed examination of the role of source constraints in disclosing information in the context of data integration. We focus on mappings from the sources given by universal Horn rules, where the global schema comes with no constraints. Since our disclosure problem requires reasoning over all sources satisfying the constraints, we need a constraint formalism that admits effective reasoning. We will look at a variety of well-studied rule-based formalisms, with the simplest being referential constraints, and the most complex being the *frontier-guarded rules* [Baget *et al.*, 2011]. While decidability of our disclosure problems will follow from prior work [Benedikt *et al.*, 2016], we will need new tools to analyze the complexity of the problem. In Section 3, we give reductions of disclosure problems to the *query entailment problem* that is heavily-studied in knowledge representation. While a naïve application of the reduction allows us only to conclude very pessimistic bounds, a more fine-grained analysis, combined with some recent results on CQ entailment, will allow us to get much better bounds, in some cases ensuring tractability. In Section 4, we complement these results with lower bounds. Both the upper and lower bounds revolve around a complexity analysis for reasoning with *guarded existential rules and a restricted class of equality rules, where the rule head compares a variable and a distinguished constant*. We believe this exploration of limited equality rules can be productive for other reasoning problems.

Overall we get a complete picture of the complexity of disclosure in the presence of source constraints for many natural classes: see Tables 1 in Section 6 for a summary of our bounds. Full proofs are available in the appendix.

2 Preliminaries

We adopt standard notions from function-free first-order logic over a vocabulary of relational names. An *instance* is a finite set of facts. By a *query* we always mean a *conjunctive query* (CQ), which is a first-order formula of the form $\exists \vec{x} \bigwedge A_i$, where each A_i is an atom. The *arity* of a CQ is the number of its free variables, and CQs of arity 0 are *Boolean*.

Data Integration. Assume that the relational names in the vocabulary are split into two disjoint subsets: *source* and *global schema*. The *arity* of such a schema is the maximal arity of its relational names. We consider a set $\mathcal{M}$ of *mapping rules* between source relations and a global schema relation $\mathcal{T}$ given. We focus on rules

$$\phi(\vec{x}, \vec{y}) \rightarrow \mathcal{T}(\vec{x})$$

where ϕ is a conjunctive query, there are no repeated variables in $\mathcal{T}(\vec{x})$, and where each global schema relation $\mathcal{T}$ is associated with *exactly one rule*. Such rules are sometimes called "GAV mappings" in the database literature [Lenzerini, 2002], and the unique ϕ associated to a global relation $\mathcal{T}$ is referred to as the *definition* of $\mathcal{T}$. The rules are *guarded* ($\mathcal{M} \in \mathsf{GuardedMap}$) if for every rule, there exists an atom in the antecedent ϕ that contains all the variables of ϕ. The rules are *atomic* ($\mathcal{M} \in \mathsf{AtomMap}$) if each ϕ consists of a single atom, and they are *projection maps* ($\mathcal{M} \in \mathsf{ProjMap}$) if each ϕ is a single atom with no repeated variables.

Given an instance $\mathcal{D}$ for the source relations, the *image of* $\mathcal{D}$ *under mapping* $\mathcal{M}$, denoted $\mathcal{M}(\mathcal{D})$, is the instance for the global schema consisting of all facts $\{\mathcal{T}(\vec{c}) \mid \mathcal{D} \models \exists \vec{y}\, \phi(\vec{c})\}$, where ϕ is the definition of $\mathcal{T}$.

Source constraints. We consider restrictions on the sources in the form of rules. A *tuple-generating dependency (TGD)* is a universally quantified sentence of the form $\varphi(\mathbf{x}, \mathbf{z}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y})$, where the *body* $\varphi(\mathbf{x}, \mathbf{z})$ and the *head* $\psi(\mathbf{x}, \mathbf{y})$ are conjunctions of atoms such that each term is either a constant or a variable in $\mathbf{x} \cup \mathbf{z}$ and $\mathbf{x} \cup \mathbf{y}$, respectively. Variables $\mathbf{x}$, common to the head and body, are called the *frontier variables*. A *frontier-guarded TGD* (FGTGD) is a TGD in which there is an atom of the body that contains every frontier variable. We focus on FGTGDs because they have been heavily studied in the database and knowledge representation community, and it is known that many computational problems involving FGTGDs are decidable [Baget *et al.*, 2011]. In particular this is true of the *query entailment problem*, which asks, given a finite collection of facts $\mathcal{F}$, a finite set Σ of sentences, and a CQ Q, whether $\mathcal{F} \wedge \Sigma$ entails Q. We use $\mathsf{QEntail}(\mathcal{F}, \Sigma, Q)$ to denote an instance of this problem and also say that "$\mathcal{F}$ entails Q w.r.t. constraints Σ". A special case of FGTGDs are *Guarded TGDs* (GTGDs), in which there is an atom containing all body variables. These specialize further to *linear TGDs* (LTGDs), whose body consists of a single atom; and even further to *inclusion dependencies* (IncDeps), a linear TGD with a single atom in the head, in which no variable occurs multiple times in the body, and no variable occurs multiple times in the head. Even IncDeps occur quite commonly: for example, the source constraints of Example 2 can be rewritten as IncDeps. The most specialized class we study are the *unary* IncDep*s*:(UIDs), which are IncDeps with at most one frontier variable.

Queries and disclosure. The sensitive information in a data integration setting is given by a CQ p over the source schema, which we refer to as the *policy*. Intuitively, disclosure of sensitive information occurs in a source instance $\mathcal{D}$ whenever the attacker can infer from the image $\mathcal{M}(\mathcal{D})$ that p holds of a tuple in $\mathcal{D}$. Formally, we say an instance $\mathcal{V}$ for the global schema is *realizable*, with respect to mappings $\mathcal{M}$ and source constraints Σ_{Source} if there is some source instance $\mathcal{D}$ that satisfies Σ_{Source} such that $\mathcal{M}(\mathcal{D}) = \mathcal{V}$. For a realizable $\mathcal{V}$, the set of such $\mathcal{D}$ are the *possible source instances for* $\mathcal{V}$. A query result $p(\vec{t})$ is *disclosed* at $\mathcal{V}$ if $p(\vec{t})$ holds on all possible source instances for $\mathcal{V}$. A query p *admits a disclosure* (for mappings $\mathcal{M}$ and source constraints Σ_{Source}) if there is some realizable instance $\mathcal{V}$ and binding $\vec{t}$ for the free variables of p for which $p(\vec{t})$ is disclosed. In this terminology, the conclusion of Example 2 was that policy $\mathsf{PatSpec}(p, s)$ admits a disclosure with respect to the constraints and mappings. For a class of constraints $\mathcal{C}$, a class of mappings Map, a class of policies Policy, we write $\mathsf{Disclose}_{\mathsf{C}}(\mathcal{C}, \mathsf{Map})$ to denote the problem of determining whether a policy (a CQ, unless other-

wise stated) admits a disclosure for a set of mappings in Map and a set of source constraints in $\mathcal{C}$. Given $\Sigma_{\mathsf{Source}}, \mathcal{M}$ and a CQ p, the corresponding instance of this problem is denoted by $\mathsf{Disclose}(\mathcal{C}, \mathcal{M}, p)$. In this paper we will focus on disclosure for queries and constraints *without constants*, although our techniques extend to the setting with constants, as long as distinct constants are not assumed to be unequal.

3 Reducing disclosure to query entailment

Our first goal is to provide a reduction from $\mathsf{Disclose}_{\mathsf{C}}(\mathsf{TGD}, \mathsf{Map})$ to a finite collection of standard query entailment problems. For simplicity we will restrict to Boolean queries p in stating the results, but it is straightforward to extend the reductions and results to the non-Boolean case. We first recall a prior reduction of $\mathsf{Disclose}_{\mathsf{C}}(\mathsf{TGD}, \mathsf{Map})$ to a more complex problem, the *hybrid open and closed world query answering problem* [Lutz *et al.*, 2013; Lutz *et al.*, 2015; Franconi *et al.*, 2011], denoted HOCWQ. HOCWQ takes as input a set of facts $\mathcal{F}$, a collection of constraints Σ, a Boolean query Q, and additionally a subset $\mathcal{C}$ of the vocabulary. A *possible world* for such $\mathsf{HOCWQ}(\mathcal{F}, \Sigma, Q, \mathcal{C})$ is any instance $\mathcal{D}$ containing $\mathcal{F}$, satisfying Σ, and such that for each relation $C \in \mathcal{C}$, the C-facts in $\mathcal{D}$ are the same as the C-facts in $\mathcal{F}$. $\mathsf{HOCWQ}(\mathcal{F}, \Sigma, Q, \mathcal{C})$ holds if Q holds in every possible world. Note that the query entailment problem is a special case of HOCWQ, where $\mathcal{C}$ is empty.

Given a set of mapping rules $\mathcal{M}$ of the form $\phi(\vec{y}, \vec{x}) \rightarrow \mathcal{T}(\vec{x})$, we let $\mathcal{G}(\mathcal{M})$ be the set of global schema predicates, and let $\Sigma_{\mathcal{M}}(\mathcal{M})$ be the mapping rules, considered as bidirectional constraints between global schema predicates and sources.

We now recall one of the main results of [Benedikt *et al.*, 2016]:

Theorem 1. *There is an instance $\mathcal{D}'$ computable in linear time from $\Sigma_{\mathsf{Source}}, \mathcal{M}, p$, such that $\mathsf{Disclose}(\Sigma_{\mathsf{Source}}, \mathcal{M}, p)$ holds if and only if $\mathsf{HOCWQ}(\mathcal{D}', \Sigma_{\mathsf{Source}} \cup \Sigma_{\mathcal{M}}(\mathcal{M}), p, \mathcal{G}(\mathcal{M}))$ holds.*

In fact, the arguments in [Benedikt *et al.*, 2016] show that $\mathcal{D}'$ can be taken to be a very simple instance, the *critical instance* over the global schema $\mathcal{G}(\mathcal{M})$ denoted $\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}}$ where $\mathcal{D}^{\mathcal{S}}_{\mathsf{Crit}}$, for $\mathcal{S}$ a set of predicates, denotes the instance that mentions only a single element c_{Crit}, and contains, for each relation R in $\mathcal{S}$ of arity n, the fact $R(c_{\mathsf{Crit}}, \ldots c_{\mathsf{Crit}})$.

Corollary 1. $\mathsf{Disclose}_{\mathsf{C}}(\mathsf{FGTGD}, \mathsf{CQMap})$ *is in* 2ExpTime.

Proof. The non-classical aspect of HOCWQ comes into play with rules of $\Sigma_{\mathcal{M}}(\mathcal{M})$ of form $\phi(\vec{x}, \vec{y}) \rightarrow \mathcal{T}(\vec{x})$. But in the context of $\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}}$, these can be rewritten as *single-constant equality rules* (SCEQrules) $\phi(\vec{x}, \vec{y}) \rightarrow \bigwedge_i x_i = c_{\mathsf{Crit}}$. Such rules remain in the Guarded Negation Fragment of first-order logic, which also subsumes FGTGDs, while having a query entailment problem in 2ExpTime [Bárány *et al.*, 2015]. □

We now want to conduct a finer-grained analysis, looking for cases that give lower complexity. To do this we will transform further into a classical query entailment problem. This will require a transformation of our query p, a transformation of our source constraints and mappings into a new set of constraints, and a transformation of the instance $\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}}$. The idea of the transformation is that we remove the SCEQrules that are implicit in the HOCWQ problem, replacing them with constraints and queries that reflect all the possible impacts the rules might have on identifying two variables.

We first describe the transformation of the query and the constraints. They will involve introducing a new unary predicate $\mathsf{IsCrit}(x)$; informally this states that x is equal to c_{Crit}. Consider a CQ $Q = \exists \vec{y} \bigwedge A_i$. An *annotation* of Q is a subset of Q's variables. Given an annotation Annot of Q, we let Q_{Annot} be the query obtained from Q by performing the following operation for each v in Annot: for all occurrences j of v except the first one, replacing v with a fresh variable v_j; and adding conjuncts $\mathsf{IsCrit}(v_j)$ as well as $\mathsf{IsCrit}(v)$ to Q_{Annot}. A *critical-instance rewriting* of a CQ Q is a CQ obtained by applying the above process to Q for any annotation. We write $Q_{\mathsf{Annot}} \in \mathsf{CritRewrite}(Q)$ to indicate that Q_{Annot} is such a rewriting.

To transform the mapping rules and constraints to a new set of constraints using $\mathsf{IsCrit}(x)$, we lift the notion of critical-instance rewriting to TGDs in the obvious way: a critical-instance rewriting of a TGD σ (either in Σ_{Source} or $\Sigma_{\mathcal{M}}(\mathcal{M})$), is the set of TGDs formed by applying the above process to the body of σ. We write $\sigma_{\mathsf{Annot}} \in \mathsf{CritRewrite}(\Sigma)$ to indicate that σ_{Annot} is a critical-instance rewriting for a $\sigma \in \Sigma$, and similarly for mappings. For example, the second mapping rule in Example 1 has several rewritings; one of them will change the rule body to $\mathsf{PatBdlg}(p, b) \wedge \mathsf{IsOpen}(b', d) \wedge \mathsf{IsCrit}(b) \wedge \mathsf{IsCrit}(b')$.

Our transformed constraints will additionally use the set of constraints $\mathsf{IsCrit}(\mathcal{M})$, including all rules:

$$\mathcal{T}(x_1 \ldots x_n) \rightarrow \mathsf{IsCrit}(x_i)$$

where $\mathcal{T}$ ranges over the global schema and $1 \leq i \leq n$. Informally $\mathsf{IsCrit}(\mathcal{M})$ states that all elements in the mapping image must be c_{Crit}. We also need to transform the instance, using a source instance with "witnesses for the target facts". Consider a fact $\mathcal{T}(c_{\mathsf{Crit}} \ldots c_{\mathsf{Crit}})$ in $\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}}$ formed by applying a mapping rule $\bigwedge_i A_i(\vec{x}_i, \vec{y}_i) \rightarrow \mathcal{T}(\vec{x})$ in $\mathcal{M}$. The *set of witness tuples for $\mathcal{T}(\vec{x})$* is the set $A_i(\vec{c})$, where $\vec{c}$ contains c_{Crit} in each position containing a variable x_j and containing a constant c_{y_j} in every position containing a variable y_j. That is the witness tuples are witnesses for the fact $\mathcal{T}(c_{\mathsf{Crit}} \ldots c_{\mathsf{Crit}})$, where each existential witness is chosen fresh. Let $\mathsf{Hide}_{\mathcal{M}}(\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}})$ be the instance formed by taking the witness tuples for every fact $\mathcal{T}(c_{\mathsf{Crit}} \ldots c_{\mathsf{Crit}}) \in \mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}}$.

We are now ready to state the reduction of the disclosure problem to query entailment:

Theorem 2. $\mathsf{Disclose}(\Sigma_{\mathsf{Source}}, \mathcal{M}, p)$ *holds exactly when there is a $p_{\mathsf{Annot}} \in \mathsf{CritRewrite}(p)$ such that $\mathsf{Hide}_{\mathcal{M}}(\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}})$ entails p_{Annot} w.r.t. constraints:*

$$\mathsf{CritRewrite}(\Sigma_{\mathsf{Source}}) \cup \mathsf{CritRewrite}(\mathcal{M}) \cup \mathsf{IsCrit}(\mathcal{M})$$

Note that Theorem 2 does not give a polynomial time reduction: both $\mathsf{CritRewrite}(\Sigma_{\mathsf{Source}})$ and $\mathsf{CritRewrite}(\mathcal{M})$ can

contain exponentially many rewritings, and further there can be exponentially many rewritings in $\mathsf{CritRewrite}(p)$.

However, the algorithm does give us a better bound in the case of Guarded TGDs with bounded arity.

Corollary 2. *If we bound the arity of schema relations, then* $\mathsf{Disclose_C}(\mathsf{GTGD}, \mathsf{GuardedMap})$ *is in* EXPTIME.

Proof. First, by introducing additional intermediate relations and source constraints, we can assume that $\mathcal{M}$ contains only projection mappings. Thus we can guarantee that $\mathsf{CritRewrite}(\mathcal{M})$ just contains the rules in $\mathcal{M}$. By introducing intermediate relations and additional source constraints, we can also assume that each $\mathsf{GTGD} \in \Sigma_{\mathsf{Source}}$ has a body with at most two atoms. Since the arity of relations is fixed, the size of such 1- or 2-atom bodies is fixed as well. From this we see that the number of constraints in any $\mathsf{CritRewrite}(\sigma)$ is polynomial. The reduction in Theorem 2 thus gives us exponentially many GTGD entailment problems of polynomial size. Since entailment over Guarded TGDs with bounded arity is in EXPTIME [Calì *et al.*, 2013], we can conclude. □

Refinements of the reduction to identify lower complexity cases. In order to lower the complexity to EXPTIME *without* bounding the arity, we refine the construction of the function $\mathsf{CritRewrite}(\sigma)$ in the case where σ is a linear TGD, providing a function $\mathsf{CritRewrite}_{\text{PTIME}}(\sigma)$ that constructs only *polynomially many rewritten constraints*.

Let $\sigma = B(\vec{x}) \rightarrow \exists \vec{y}\, H(\vec{z})$ be a linear TGD with relation B of arity k, and suppose $\vec{x}$ contains d distinct free variables $V = \{v_1 \dots v_d\}$. Let P be the set of pairs (e, f) with $e < f \leq k$ such that the same variable v_i sits at positions e and f in $\vec{x}$. We order P as $(e_0, f_0) \dots (e_h, f_h)$; for each (e, f) that is not the initial pair (e_0, f_0), we let $(e, f)^-$ be its predecessor in the linear order.

We let $B_{e,f}$ denote new predicates of arity k for each $(e, f) \in P$. Let $\vec{w}$ be a set of k distinct variables, and $\vec{w}^{i=j}$ be formed from $\vec{w}$ by replacing w_j with w_i. We begin the construction of $\mathsf{CritRewrite}_{\text{PTIME}}(\sigma)$ with the constraints:

$$B(\vec{w}^{e_0=f_0}) \rightarrow B_{e_0,f_0}(\vec{w}^{e_0=f_0})$$
$$B(\vec{w}) \wedge \mathsf{IsCrit}(w_{e_0}) \wedge \mathsf{IsCrit}(w_{f_0}) \rightarrow B_{e_0,f_0}(\vec{w})$$

We add to $\mathsf{CritRewrite}_{\text{PTIME}}(\sigma)$ the following constraints, for each (e, f) with a predecessor $(e, f)^- = (e', f')$.

$$B_{e',f'}(\vec{w}^{e=f}) \rightarrow B_{e,f}(\vec{w}^{e=f})$$
$$B_{e',f'}(\vec{w}) \wedge \mathsf{IsCrit}(w_e) \wedge \mathsf{IsCrit}(w_f) \rightarrow B_{e,f}(\vec{w})$$

Letting e_h, f_h the final pair in P, we add to $\mathsf{CritRewrite}_{\text{PTIME}}(\sigma)$ the constraint

$$B_{e_h,f_h}(\vec{x}') \rightarrow \exists \vec{y}\, H(\vec{z})$$

where $\vec{x}'$ is obtained from $\vec{x}$ by replacing all but the first occurrence of each variable v by a fresh variable.

If Σ_{Source} consists of LTGDs, we let $\mathsf{CritRewrite}_{\text{PTIME}}(\Sigma_{\mathsf{Source}})$ be the result of applying this process to every $\sigma \in \Sigma_{\mathsf{Source}}$. Similarly, if $\mathcal{M}$ consists of atomic mappings (implying that the associated rules are LTGDs), then we let $\mathsf{CritRewrite}_{\text{PTIME}}(\mathcal{M})$ the result of applying the process above to the rule going from source relation to global schema relation associated to $m \in \mathcal{M}$. Then we have:

Theorem 3. *When* Σ_{Source} *consists of* LTGD*s,* $\mathsf{Disclose}(\Sigma_{\mathsf{Source}}, \mathcal{M}, p)$ *holds exactly when there is a* $p_{\mathsf{Annot}} \in \mathsf{CritRewrite}(p)$ *such that* $\mathsf{Hide}_{\mathcal{M}}(\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}})$ *entails* p_{Annot} *w.r.t. to the constraints*

$$\mathsf{CritRewrite}_{\text{PTIME}}(\Sigma_{\mathsf{Source}}) \cup \mathsf{CritRewrite}_{\text{PTIME}}(\mathcal{M}) \cup \mathsf{IsCrit}(\mathcal{M})$$

We can combine this result with recent work on fine-grained complexity of GTGDs to improve the doubly exponential upper bound of Corollary 1 for linear TGD source constraints and atomic mappings:

Theorem 4. $\mathsf{Disclose_C}(\mathsf{LTGD}, \mathsf{AtomMap})$ *is in* EXPTIME. *If the arity of relations in the source schema is bounded, then the complexity drops to* NP*, while if further the policy is atomic, the problem is in* PTIME.

Proof. It is sufficient to get an EXPTIME algorithm for the entailment problem produced by Theorem 3, since then we can apply it to each p_{Annot} in EXPTIME. The constraints in $\mathsf{CritRewrite}_{\text{PTIME}}(\Sigma_{\mathsf{Source}}) \cup \mathsf{CritRewrite}_{\text{PTIME}}(\mathcal{M})$ are Guarded TGDs that are not necessarily LTGDs. But the bodies of these guarded TGDs consist of a guard predicate and atoms over a fixed "side signature", namely the unary predicate IsCrit. It is known that the query entailment for IncDeps and guarded TGDs with a fixed side signature is in EXPTIME, with the complexity dropping to NP (resp. PTIME) when the arity is fixed (resp. fixed and the query is atomic) [Amarilli and Benedikt, 2018a]. □

Can we do better than EXPTIME? We can note that when the constraints $\sigma \in \Sigma_{\mathsf{Source}}$ are IncDeps, $\mathsf{CritRewrite}(\sigma)$ consists only of σ; similarly if a mapping $m \in \mathcal{M}$ is a projection, then $\mathsf{CritRewrite}(m)$ consists only of m. This gives us a good upper bound in one of the most basic cases:

Corollary 3. $\mathsf{Disclose_C}(\mathsf{IncDep}, \mathsf{ProjMap})$ *is in* PSPACE. *If further a bound is fixed on the arity of relations in the source schema, then the problem becomes* NP*, dropping to* PTIME *when the policy is atomic.*

Proof. Our algorithm will guess a p_{Annot} in $\mathsf{CritRewrite}(Q)$ and checks the entailment of Theorem 2. This gives an entailment problem for IncDeps, known to be in PSPACE in general, in NP for bounded arity, and in PTIME for bounded arity and atomic queries [Johnson and Klug, 1984]. □

Obtaining tractability. Thus far we have seen cases where the complexity drops to PSPACE in the general case and NP in the bounded arity case, and PTIME for atomic queries. We now present a case where we obtain tractability for arbitrary queries and arity. Recall that a UID is an IncDep where at most one variable is exported. They are actually quite common, capturing referential integrity when data is identified by a single attribute. We can show that restricting to UIDs while having only projection maps leads to tractability:

Theorem 5. $\mathsf{Disclose_C}(\mathsf{UID}, \mathsf{ProjMap})$ *is in* PTIME.

Proof. The first step is to refine the reduction of Theorem 2 to get an entailment problem with only UIDs, over an instance consisting of a single unary fact $\mathsf{IsCrit}(c_{\mathsf{Crit}})$. The main issue is avoid the constraints in $\Sigma_{\mathcal{M}}(\mathcal{M})$, corresponding to

the mapping rules. The intuition for this is that on $\mathcal{D}_{\mathsf{Crit}}^{\mathcal{G}(\mathcal{M})}$, the only impact of the backward and forward implications of $\Sigma_{\mathcal{M}}(\mathcal{M})$ is to create new facts among the source relations. In these new facts only c_{Crit}, is propagated. Rather than creating SCEQrules (implicitly what happens in the HOCWQ reduction) or generating classical constraints where the impact of the equalities are "baked in" (as in the critical-instance rewritings of Theorems 2 and 3), we truncate the source relations to the positions where non-visible elements occur, while generating UIDs on these truncated relations that simulate the impact of back-and-forth using $\Sigma_{\mathcal{M}}(\mathcal{M})$.

The second step is to show that query entailment with UIDs over the instance consisting only of $\mathsf{IsCrit}(c_{\mathsf{Crit}})$ is in PTIME. This can be seen as an extension of the PTIME inference algorithm for UIDs [Cosmadakis *et al.*, 1990]. The idea behind this result is to analyze the classical "chase procedure" for query entailment with TGDs [Fagin *et al.*, 2005]. In the case of UIDs over a unary fact, the shape of the chase model is very restricted; roughly speaking, it is a tree where only a single fact connects two values. Based on this, we can simplify the query dramatically, making it into an acyclic query where any two variables co-occur in at most one predicate. Once query simplification is performed, we can reduce query entailment to polynomial many entailment problems involving individual atoms in the query. This in turn can be solved using the UID inference procedure of [Cosmadakis *et al.*, 1990]. □

4 Lower Bounds

We now focus on providing lower bounds for $\mathsf{Disclose}_{\mathsf{C}}(\mathcal{C}, \mathsf{Map})$, showing in particular that the upper bounds provided in Section 3 can not be substantially improved. For many classes of constraints it is easy to see that the complexity of disclosure inherits the lower bounds for the classical entailment problem for the class. From this we get a number of matching lower bounds; e.g. 2EXPTIME for GTGD constraints, PSPACE for IncDep constraints. But note that in some cases the upper bounds we have provided for disclosure in Section 3 are higher than the complexity of entailment over the source constraints. For example, for IncDeps we have provided only a 2EXPTIME upper bound for guarded mappings (from Corollary 1), and only an exponential bound for atomic mappings (from Theorem 4). This suggests that the form of the mappings influences the complexity as well, as we now show.

Most of our proofs for hardness above the entailment bound for source constraints rely on the encoding of a Turing machine. Source constraints are used to generate the underlying structures (tree of configurations, tape of a Turing machine) while mappings are used to ensure consistency (a universal configuration is accepting if and only if all its successor configurations are accepting, the content of the tape is consistently represented,...). To illustrate our approach, we sketch the proof of the following result.

Theorem 6. $\mathsf{Disclose}_{\mathsf{C}}(\mathsf{IncDep}, \mathsf{GuardedMap})$ *and* $\mathsf{Disclose}_{\mathsf{C}}(\mathsf{GTGD}, \mathsf{ProjMap})$ *are* 2EXPTIME*-hard, and are* EXPTIME*-hard even in bounded arity.*

Proof. Recall that Theorem 1 relates disclosure to a HOCWQ problem on $\mathcal{D}_{\mathsf{Crit}}^{\mathcal{G}(\mathcal{M})}$. Also recall from Section 3 the intuition that such a problem amounts to a classical entailment problem for a CQ over a very simple instance, using the source dependencies and SCEQrules: of the form $\phi(\vec{x}) \rightarrow x = c_{\mathsf{Crit}}$, where ϕ will be the body of a mapping. We will sketch how to simulate an alternating EXPSPACE Turing machine $\mathcal{M}$ using a QEntail problem using IncDeps and *guarded* SCEQrules. This can in turn be simulated using our HOCWQ problem.

We first build a tree of configurations using IncDeps, such that each node has a type (existential or universal) and is the parent of two nodes (called α-successor and β-successor) of the opposite type. This tree structure is represented, together with additional information, by atoms such as:

$$\mathsf{Children}_{\forall}(c, c_\alpha, c_\beta, ac, ac_\alpha, ac_\beta, \vec{y_0}, \vec{y_1}, r).$$

Intuitively, this states that c is a universal configuration, parent of c_α and c_β. ac (resp. ac_α, resp. ac_β) is the acceptance bit for c (resp. c_α, resp. c_β), which will be made equal to c_{Crit} if and only if the configuration represented by c (resp. c_α, resp. c_β) is accepting. $\vec{y_0}, \vec{y_1}$ will be used to represent cell addresses, while r is the identifier of the root of the configuration tree. The initial instance is such an atom, where the first position and the last position are the same constant, $\vec{y_0}$ is a vector of n 0's, $\vec{y_1}$ is a vector of n 1's, and all other arguments are distinct constants.

We use SCEQrules to propagate acceptance information up in the tree. For instance, a universal configuration is accepting if both its successors are accepting. This is simulated by the following SCEQrule:

$$\mathsf{Children}_{\forall}(c, c_\alpha, c_\beta, ac_c, c_{\mathsf{Crit}}, c_{\mathsf{Crit}}, \vec{y_0}, \vec{y_1}, r) \rightarrow ac_c = c_{\mathsf{Crit}}.$$

To simulate $\mathcal{M}$, we need access to an exponential number of cells for each configuration. We identify a cell by the configuration it belongs to and an address, which is a vector, generated by IncDeps, of length n whose arguments are either 0 or 1. The atom for representing a cell is thus:

$$\mathsf{Cell}(c_p, c, \vec{addr}, \vec{v}, \vec{v}_{prev}, \vec{v}_{next}),$$

where c_p is the parent configuration of c, which is the configuration to which the represented cell belongs, $\vec{addr}$ is the address of the cell, $\vec{v}$ its content, $\vec{v}_{prev}$ the content of the previous cell, and $\vec{v}_{next}$ the content of the next cell. Note that this representation is redundant, and we need to use SCEQrules to ensure its consistency.

Note that $\vec{v}$ is a tuple of length the size of $(\Sigma \cup \{\flat\}) \times (Q \cup \{\bot\})$. Each position corresponds to an element of that set, and the content of a represented cell is the element which corresponds to the unique position in which c_{Crit} appears.

We now explain how to build the representation of the initial tape, and simulate the transition function. Both steps are done by unifying some nulls with c_{Crit}. W.l.o.g., we assume that the initial tape contains a l in the first cell, on which points the head of $\mathcal{M}$ in a state s, and that (l, s) corresponds to the first bit of $\vec{v}$. We thus use a SCEQrule to set this bit to c_{Crit} in the first cell of the first configuration. We then set (w.l.o.g.) the second bit of all the other cells of that configuration to c_{Crit} (assuming this represents $(\flat, \bot)$).

To simulate the transitions, we note that the content of a cell in a configuration depends only on the content of the

Σ_{Source} \ $\mathcal{M}$	Unbounded arity				Bounded arity			
	ProjMap	AtomMap	GuardedMap	CQMap	ProjMap	AtomMap	GuardedMap	CQMap
IncDep	$\text{PSPACE}^{U=C3}_{L=\mathsf{QEntail}}$	$\text{EXPTIME}_{L=T7}$	$\text{2EXPTIME}_{L=T6}$	2EXPTIME	$\text{NP}_{L=\mathsf{QEntail}}$	NP	$\text{EXPTIME}_{L=T6}$	$\text{2EXPTIME}_{L=T8}$
LTGD	$\text{EXPTIME}_{L=T7}$	$\text{EXPTIME}^{U=T4}$	2EXPTIME	2EXPTIME	NP	$\text{NP}^{U=T4}$	EXPTIME	2EXPTIME
GTGD	$\text{2EXPTIME}_{L=T6}$	2EXPTIME	2EXPTIME	2EXPTIME	$\text{EXPTIME}_{L=T6}$	EXPTIME	$\text{EXPTIME}^{U=C2}$	2EXPTIME
FGTGD	2EXPTIME	2EXPTIME	2EXPTIME	$\text{2EXPTIME}^{U=C1}$	$\text{2EXPTIME}_{L=\mathsf{QEntail}}$	2EXPTIME	2EXPTIME	$\text{2EXPTIME}^{U=C1}$

Table 1: Complexity of disclosure: $\text{PSPACE}^{U=C3}_{L=\mathsf{QEntail}}$ means the corresponding problem is PSPACE-complete, where the Upper bound is given by Corollary 3 (U=C3) and the Lower bound is inherited from entailment. We omit bounds inferred from inclusion ($\mathcal{M}$ or Σ_{Source}).

same cell in the parent configuration, along with the content of parent's previous and next cells. We thus add a SCEQrule that checks for the presence of c_{Crit} specifying the content of three consecutive cells in a configuration, and unify a null with c_{Crit} to specify the content of the corresponding cell of a child configuration.

The argument above uses IncDeps and GuardedMaps, but we can simplify the mappings to ProjMap using GTGDs. □

A simple variation of the construction used for PSPACE-hardness of entailment with IncDeps [Casanova *et al.*, 1984] shows that our upper bounds for IncDep source constraints and atomic maps are tight. The case of LTGD source constraints and projection maps can be done via reduction to that of IncDep source constraints and atomic maps:

Theorem 7. $\mathsf{Disclose}_{\mathsf{C}}(\mathsf{IncDep}, \mathsf{AtomMap})$ *and* $\mathsf{Disclose}_{\mathsf{C}}(\mathsf{LTGD}, \mathsf{ProjMap})$ *are both* EXPTIME*-hard.*

The above results, coupled with argument that the lower bounds for entailment are inherited by disclosure, show tightness of all upper bounds from Table 1 in the unbounded arity case. Another variation of the encoding in Theorem 6 shows that with no restriction on the mappings one can not do better than the 2EXPTIME upper bound of Corollary 1 even for IncDep constraints in bounded arity,

Theorem 8. $\mathsf{Disclose}_{\mathsf{C}}(\mathsf{IncDep}, \mathsf{CQMap})$ *is* 2EXPTIME*-hard in bounded arity.*

The theorem above, again combined with results showing that the lower bounds for entailment are inherited, suffice to show tightness of all upper bounds from Table 1 in the case of bounded arity.

We can also show that our tractability result for UID constraints and projection maps does not extend when either the maps or the constraints are broadened. Informally, this is because with these extensions we can generate an instance on which CQ querying is NP-hard.

5 Related Work

Disclosure analysis has been approached from many angles. We do not compare with the vast amount of work that analyzes probabilistic mechanisms for releasing information, providing probabilistic guarantees on disclosure [Dwork, 2006]. Our work focuses on the impact of reasoning on mapping-based mechanisms used in knowledge-based information integration, which are deterministic; thus one would prefer, and can hope for, deterministic guarantees on disclosure. We deal here with the *analysis* of disclosure, while there is a complementary literature on how to *enforce* privacy [Biskup and Weibert, 2008; Bonatti *et al.*, 1995; Bonatti and Sauro, 2013; Studer and Werner, 2014].

The problem of whether information is disclosed on a particular instance (variation of HOCWQ introduced in Section 3) has been studied in both the knowledge representation [Lutz *et al.*, 2013; Lutz *et al.*, 2015; Franconi *et al.*, 2011; Ahmetaj *et al.*, 2016; Amendola *et al.*, 2018] and database community [Abiteboul and Duschka, 1998]. The corresponding schema-level problem was defined in [Benedikt *et al.*, 2016], which allows arbitrary constraints relating the source and the global schema. However, results are provided only for constraints in guarded logics, which does not subsume the case of mappings given here. Our results clarify some issues in prior work: [Benedikt *et al.*, 2016] claimed that disclosure with IncDep source constraints and atomic maps is in PSPACE, while our Theorem 7 shows that the problem is EXPTIME-hard. Our notion of disclosure corresponds to the complement of [Benedikt *et al.*, 2018]'s "data-independent compliance". The formal framework of [Benedikt *et al.*, 2018] is orthogonal to ours. On the one hand, source constraints are absent; on the other hand a more powerful mapping language is considered, with existentials in the head of rules, while constraints on the global schema, given by ontological axioms, are now allowed. [Benedikt *et al.*, 2018] assume that the attacker has an interface for posing queries against the global schema, with the queries being answered under entailment semantics. In general, the semantic information on the global schema makes disclosure harder, since the outputs of different mapping rules may be indistinguishable by an attacker who only sees the results of reasoning. In contrast, source constraints make disclosure of secrets easier, since they provide additional information to the attacker.

6 Summary and conclusion

We have isolated the complexity of information disclosure from a schema in the presence of commonly-studied sets of source constraints. A summary of many combinations of mappings $\mathcal{M}$ and source constraints Σ_{Source} is given in Table 1: note that *all problems are complete for the complexity classes listed*. We have shown tractability in the case of UIDs and projection maps (omitted in the tables), while showing that lifting the restriction leads to intractability. But we leave open a finer-grained analysis of complexity for frontier-one constraints with more general mappings. Our results depend on a fine-grained analysis of reasoning with TGDs and SCEQrules, a topic we think is of independent interest.

References

[Abiteboul and Duschka, 1998] Serge Abiteboul and Olivier Duschka. Complexity of answering queries using materialized views. In *PODS*, 1998.

[Ahmetaj *et al.*, 2016] Shqiponja Ahmetaj, Magdalena Ortiz, and Mantas Šimkus. Polynomial datalog rewritings for expressive description logics with closed predicates. In *IJCAI*, 2016.

[Amarilli and Benedikt, 2018a] Antoine Amarilli and Michael Benedikt. When Can We Answer Queries Using Result-Bounded Data Interfaces? In *PODS*, 2018.

[Amarilli and Benedikt, 2018b] Antoine Amarilli and Michael Benedikt. When Can We Answer Queries Using Result-Bounded Data Interfaces? In *arxiv*, 2018. available at https://arxiv.org/pdf/1706.07936.pdf.

[Amendola *et al.*, 2018] Giovanni Amendola, Nicola Leone, Marco Manna, and Pierfrancesco Veltri. Enhancing existential rules by closed-world variables. In *IJCAI*, 2018.

[Baget *et al.*, 2011] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. On rules with existential variables: Walking the decidability line. *Artif. Intell.*, 175(9-10), 2011.

[Bárány *et al.*, 2015] Vince Bárány, Balder Ten Cate, and Luc Segoufin. Guarded negation. *J. ACM*, 62(3), 2015.

[Benedikt *et al.*, 2016] Michael Benedikt, Pierre Bourhis, Balder ten Cate, and Gabriele Puppis. Querying visible and invisible information. In *LICS*, 2016.

[Benedikt *et al.*, 2018] Michael Benedikt, Bernardo Cuenca Grau, and Egor V. Kostylev. Logical foundations of information disclosure in ontology-based data integration. *Artif. Intell.*, 262, 2018.

[Bienvenu *et al.*, 2018] Meghyn Bienvenu, Stanislav Kikot, Roman Kontchakov, Vladimir V. Podolskii, and Michael Zakharyaschev. Ontology-mediated queries: Combined complexity and succinctness of rewritings via circuit complexity. *J. ACM*, 65(5), 2018.

[Biskup and Weibert, 2008] Joachim Biskup and Torben Weibert. Keeping Secrets in Incomplete Databases. *Int. J. Inf. Sec.*, 7(3):199–217, 2008.

[Bonatti and Sauro, 2013] Piero A. Bonatti and Luigi Sauro. A confidentiality model for ontologies. In *ISWC*, 2013.

[Bonatti *et al.*, 1995] Piero Bonatti, Sarit Kraus, and V. S. Subrahmanian. Foundations of Secure Deductive Databases. *TKDE*, 7(3), 1995.

[Calì *et al.*, 2013] Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *JAIR*, 2013.

[Casanova *et al.*, 1984] Marco Casanova, Ronald Fagin, and Christos Papadimitriou. Inclusion dependencies and their interaction with functional dependencies. *JCSS*, 28(1), 1984.

[Cosmadakis *et al.*, 1990] Stavros S. Cosmadakis, Paris C. Kanellakis, and Moshe Y. Vardi. Polynomial-time implication problems for unary inclusion dependencies. *J. ACM*, 37(1), 1990.

[Dwork, 2006] Cynthia Dwork. Differential privacy. In *ICALP*, 2006.

[Fagin *et al.*, 2005] Ronald Fagin, Phokion G. Kolaitis, Renee J. Miller, and Lucian Popa. Data Exchange: Semantics and Query Answering. *Theoretical Computer Science*, 336(1), 2005.

[Franconi *et al.*, 2011] Enrico Franconi, Yasmin Ibáñez-García, and Inanç Seylan. Query answering with DBoxes is hard. *ENTCS*, 278, 2011.

[Gottlob *et al.*, 2014] Georg Gottlob, Marco Manna, and Andreas Pieris. Polynomial combined rewritings for existential rules. In *KR*, 2014.

[Johnson and Klug, 1984] David S. Johnson and Anthony C. Klug. Testing Containment of Conjunctive Queries under Functional and Inclusion Dependencies. *JCSS*, 28(1), 1984.

[Kikot *et al.*, 2011] Stanislav Kikot, Roman Kontchakov, and Michael Zakharyaschev. Polynomial conjunctive query rewriting under unary inclusion dependencies. In *RR*, 2011.

[Lenzerini, 2002] Maurizio Lenzerini. Data integration: A theoretical perspective. In *PODS*, 2002.

[Lutz *et al.*, 2013] Carsten Lutz, Inanç Seylan, and Frank Wolter. Ontology-based data access with closed predicates is inherently intractable(sometimes). In *IJCAI*, 2013.

[Lutz *et al.*, 2015] Carsten Lutz, Inanç Seylan, and Frank Wolter. Ontology-mediated queries with closed predicates. In *IJCAI*, 2015.

[Papadimitriou, 1994] Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[Sagiv and Yannakakis, 1980] Yehoshua Sagiv and Mihalis Yannakakis. Equivalences among relational expressions with the union and difference operators. *J. ACM*, 27(4), 1980.

[Studer and Werner, 2014] Thomas Studer and Johannes Werner. Censors for Boolean Description Logic. *Trans. on Data Privacy*, 7(3), 2014.

A Detailed proofs from Section 3: upper bounds for disclosure

A.1 Proof of Theorem 2: correctness of the basic reduction from disclosure to classical entailment

Recall the statement of Theorem 2, which applies the algorithms $\mathsf{CritRewrite}(\Sigma_{\mathsf{Source}})$ to TGDs and $\mathsf{CritRewrite}(\mathcal{M})$ to mappings.

$\mathsf{Disclose}(\Sigma_{\mathsf{Source}}, \mathcal{M}, p)$ holds exactly when there is a $p_{\mathsf{Annot}} \in \mathsf{CritRewrite}(p)$ such that $\mathsf{Hide}_{\mathcal{M}}(\mathcal{D}_{\mathsf{Crit}}^{\mathcal{G}(\mathcal{M})})$ entails

p_{Annot} w.r.t. constraints:

$$\mathsf{CritRewrite}(\Sigma_{\mathsf{Source}}) \cup \mathsf{CritRewrite}(\mathcal{M}) \cup \mathsf{IsCrit}(\mathcal{M})$$

holds.

By Theorem 1 we know that $\mathsf{Disclose}(\Sigma_{\mathsf{Source}}, \mathcal{M}, p)$ is equivalent to $\mathsf{HOCWQ}(\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}}, \Sigma_{\mathsf{Source}} \cup \Sigma_{\mathcal{M}}(\mathcal{M}), p, \mathcal{G}(\mathcal{M}))$.

This will immediately allow us to prove one direction of the equivalence. Suppose each of our entailments fails. From this, we see using [Sagiv and Yannakakis, 1980] that $\mathsf{Hide}_{\mathcal{M}}(\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}})$ does not entail the disjunction of p_{Annot}. Thus we have an instance $\mathcal{D}$ extending $\mathsf{Hide}_{\mathcal{M}}(\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}})$ with facts that may include the IsCrit predicate, where $\mathcal{D}$ satisfies all the rewritten constraints and no rewritten query p_{Annot}. Note that since $\mathcal{D}$ satisfies the constraints of $\mathsf{CritRewrite}(\mathcal{M})$ as well as $\mathsf{IsCrit}(\mathcal{M})$, we know that the element c_{Crit}, if it occurs in $\mathsf{Hide}_{\mathcal{M}}(\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}})$, will be labeled with IsCrit.

Form an instance $\mathcal{D}'$ by unifying all elements e in $\mathcal{D}$ satisfying IsCrit into a single element c_{Crit}, making c_{Crit} inherit any fact that such an e participates in. That is, we choose $\mathcal{D}'$ so that if h is the mapping taking any element satisfying IsCrit to c_{Crit} and fixing every other element, then h is a homomorphism from $\mathcal{D}$ onto $\mathcal{D}'$. We can easily verify that $\mathcal{D}'$ satisfies the original source constraints Σ_{Source}. For each homomorphism λ' of the body of $\sigma' \in \Sigma_{\mathsf{Source}}$ into $\mathcal{D}'$, there is a homomorphism λ of some $\sigma \in \mathsf{CritRewrite}(\sigma')$ into $\mathcal{D}$. We know σ is satisfied in $\mathcal{D}$, and taking the h-image of the tuples that witness this gives us the required witnesses for σ' in $\mathcal{D}'$. Now let $\mathcal{D}'_0$ be the restriction of $\mathcal{D}'$ to the source relations. We argue that the mapping image of $\mathcal{D}'_0$ under $\mathcal{M}$ is exactly $\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}}$. To see that the image of $\mathcal{D}'_0$ must include all the facts in $\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}}$, note that $\mathcal{D}$ includes all facts of $\mathsf{Hide}_{\mathcal{M}}(\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}})$, which contains witnesses for each such fact. Thus the h-image, namely $\mathcal{D}'$, contains witnesses for each such fact as well. Conversely, suppose the image of $\mathcal{D}'_0$ includes a fact $F(\vec{d})$; we will argue that $F(\vec{d})$ is in $\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}}$. Since $\mathcal{D}$ satisfied $\mathsf{IsCrit}(\mathcal{M})$, any such fact in $\mathcal{D}$ must have all d_i satisfying IsCrit. Thus in $\mathcal{D}'_0$ each such fact must be of the form $F(c_{\mathsf{Crit}} \dots c_{\mathsf{Crit}})$. Thus the $\mathcal{M}$-image of $\mathcal{D}'_0$ is exactly the same $\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}}$.

Finally, we claim that $\mathcal{D}'$ satisfies $\neg p$. If it satisfies p, then $\mathcal{D}$ would satisfy p_{Annot} for some annotation Annot, a contradiction. Putting this all together, we see that $\mathcal{D}'$ contradicts $\mathsf{HOCWQ}(\mathcal{D}_{\mathsf{Crit}}, \Sigma_{\mathsf{Source}} \cup \Sigma_{\mathcal{M}}(\mathcal{M}), p, \mathcal{G}(\mathcal{M}))$.

Before turning to the other direction, we will explain some other results that will be necessary. The first is the *chase procedure* for checking entailment of a query Q from a set of constraints Σ and a set of facts $\mathcal{D}$. This proceeds by building a sequence of instances $\mathcal{D} = \mathcal{D}_0 \dots \mathcal{D}_i \dots$ where each $\mathcal{D}_{i+1}$ is formed from $\mathcal{D}_i$ by "firing a rule" $\sigma \in \Sigma$ $\mathcal{D}_i$. Firing σ in $\mathcal{D}_i$ means finding a homomorphism λ from the body of σ into $\mathcal{D}_i$, and adding facts to extend λ to the head, using fresh values for all existentially quantified variables. Such a homomorphism λ is called a *trigger* for the rule firing. The chase of $\mathcal{D}$ under Σ, denoted $\mathsf{Chase}_{\Sigma}(\mathcal{D})$, is any instance formed as the union of such a sequence having the additional property that every rule that could fire in some $\mathcal{D}_i$ fires in some later $\mathcal{D}_j$. The significance of the chase for query entailment is the following result [Fagin *et al.*, 2005]:

Theorem 9. *For an instance $\mathcal{D}$, set of TGDs Σ, and UCQ Q, we have $\mathsf{QEntail}(\mathcal{D}, \Sigma, Q)$ if and only if some chase model for $\mathcal{D}$ under Σ satisfies Q.*

We will also need a variation of the chase for the problem $\mathsf{HOCWQ}(\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}}, \Sigma_{\mathsf{Source}} \cup \Sigma_{\mathcal{M}}(\mathcal{M}), p, \mathcal{G}(\mathcal{M}))$, taken from [Benedikt *et al.*, 2016]. The *visible chase* is a sequence of source instances $\mathcal{D}_0 \dots \mathcal{D}_n \dots$ that begins with $\mathcal{D}_0 = \mathsf{Hide}_{\mathcal{M}}(\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}})$. $\mathcal{D}_{i+1}$ is formed from $\mathcal{D}_i$ by "chasing and merging". The chase step applies the usual chase procedure described above to $\mathcal{D}_i$ with constraints Σ_{Source}, creating new facts that possibly contain fresh values. In a merge step, we take a mapping $m \in \mathcal{M}$ and a homomorphism λ of the body of m into $\mathcal{D}_i$, and for each free variable x of m, we replace $\lambda(x)$ by c_{Crit} in all facts in which it appears. We say that this is a *merge step with m, λ on $\mathcal{D}_i$*. Since the process is monotone, it must reach a fixpoint, which we refer to as the *visible chase* of $\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}}$, denoted $\mathsf{VisChase}(\Sigma_{\mathsf{Source}}, \mathcal{M})$.

Proposition 1. *[Benedikt* et al.*, 2016] $\mathsf{HOCWQ}(\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}}, \Sigma_{\mathsf{Source}} \cup \Sigma_{\mathcal{M}}(\mathcal{M}), p, \mathcal{G}(\mathcal{M}))$ holds exactly when $\mathsf{VisChase}(\Sigma_{\mathsf{Source}}, \mathcal{M})$ satisfies p.*

We now prove the other direction, assuming that $\mathsf{HOCWQ}(\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}}, \Sigma_{\mathsf{Source}} \cup \Sigma_{\mathcal{M}}(\mathcal{M}), p, \mathcal{G}(\mathcal{M}))$ fails, but one of the entailments holds. By Theorem 9, this means that some chase of $\mathsf{Hide}_{\mathcal{M}}(\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}})$ under the constraints $\mathsf{CritRewrite}(\Sigma_{\mathsf{Source}}) \cup \mathsf{CritRewrite}(\mathcal{M}) \cup \mathsf{IsCrit}(\mathcal{M})$ satisfies p_{Annot} for some annotation Annot. Let $\mathcal{D}'_0 \dots \mathcal{D}'_n \dots$ denote such a chase sequence for $\mathsf{Hide}_{\mathcal{M}}(\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}})$ under $\mathsf{CritRewrite}(\Sigma_{\mathsf{Source}}) \cup \mathsf{CritRewrite}(\mathcal{M}) \cup \mathsf{IsCrit}(\mathcal{M})$. We form another sequence $\mathcal{D}_0 \dots \mathcal{D}_n \dots$, with $\mathcal{D}_0 = \mathcal{D}'_0$, maintaining the invariant that there is a homomorphism h_i from $\mathcal{D}'_i$ to $\mathcal{D}_i$ mapping every element satisfying IsCrit to $\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}}$. The inductive step is performed as follows:

- For every chase step with a rule σ' of $\mathsf{CritRewrite}(\Sigma_{\mathsf{Source}})$ applied in $\mathcal{D}'_i$, having trigger λ', we know that $\sigma' = \mathsf{CritRewrite}(\sigma)$ for some $\sigma \in \Sigma_{\mathsf{Source}}$. We can apply the corresponding rule σ in $\mathcal{D}_i$, with a trigger λ that maps a variable x to the h_i-image of $\lambda'(x)$. Thus λ composed with h_i is λ.
- For every chase step in $\mathcal{D}'_i$ with a rule of $\sigma' \in \mathsf{CritRewrite}(m)$ for $m \in \mathcal{M}$ and a trigger λ, we apply a merge step in $\mathcal{D}_i$ with m and λ.

Since some $\mathcal{D}'_n$ satisfies p_{Annot}, one of the $\mathcal{D}_n$ must satisfy p_{Annot}. Since $\mathcal{D}_n$ contains the image of $\mathcal{D}'_n$ under the homomorphism h_n, and h_n maps p_{Annot} to p, we see that $\mathcal{D}_n$ must satisfy p. But $\mathcal{D}_n$ is a subinstance of the visible chase for our HOCWQ problem. Thus the assumption that $\mathsf{HOCWQ}(\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}}, \Sigma_{\mathsf{Source}} \cup \Sigma_{\mathcal{M}}(\mathcal{M}), p, \mathcal{G}(\mathcal{M}))$ fails and Proposition 1 imply that p cannot hold in $\mathcal{D}_n$, a contradiction.

A.2 Simplifying mappings

In this section, we will see that we can simplify mapping to be projection maps at the cost of moving to a richer class of

source constraints.

Given a problem $\mathsf{Disclose}(\Sigma_{\mathsf{Source}}, \mathcal{M}, p)$ we consider $\Sigma'_{\mathsf{Source}}$ and $\mathcal{M}'$ built in the following way: $\Sigma'_{\mathsf{Source}}$ is composed of Σ_{Source} plus for each mapping $\phi(\vec{x}, \vec{y}) \rightarrow T(\vec{x})$ we create a predicate $R_\phi(\vec{x}, \vec{y})$ and we add to $\Sigma'_{\mathsf{Source}}$ the two constraints $\phi(\vec{x}, \vec{y}) \rightarrow R_\phi(\vec{x}, \vec{y})$ and $R_\phi(\vec{x}, \vec{y}) \rightarrow \phi(\vec{x}, \vec{y})$. $\mathcal{M}'$ is composed of mappings $R_\phi(\vec{x}, \vec{y}) \rightarrow T_\phi(\vec{x})$.

Proposition 2. *We have* $\mathsf{Disclose}(\Sigma_{\mathsf{Source}}, \mathcal{M}, p)$ *if and only if* $\mathsf{Disclose}(\Sigma'_{\mathsf{Source}}, \mathcal{M}', p)$.

Proof. To prove the proposition, it is sufficient to prove that p holds on $\mathsf{VisChase}(\Sigma_{\mathsf{Source}}, \mathcal{M})$ if and only if p holds on $\mathsf{VisChase}(\Sigma'_{\mathsf{Source}}, \mathcal{M}')$ (see Proposition 1). Let $\Pi(\mathcal{D})$ be the instance obtained by removing all the facts $R_\phi(\vec{x}, \vec{y})$ in $\mathcal{D}$.

We recall that the visible chase works iteratively, at each step a database $\mathcal{D}_{i+1}$ is created from $\mathcal{D}_i$ by chasing all facts then merging some values with c_{Crit}. For the sake of simplicity we suppose that each step is composed of either one rule firing or one merging.

- We start by proving that $\mathsf{Disclose}(\Sigma_{\mathsf{Source}}, \mathcal{M}, p)$ implies $\mathsf{Disclose}(\Sigma'_{\mathsf{Source}}, \mathcal{M}', p)$.

  Let $\mathcal{D}_0, \ldots$ be a sequence corresponding to $\mathsf{VisChase}(\Sigma_{\mathsf{Source}}, \mathcal{M})$. We build a sequence $\mathcal{D}'_0, \ldots$ corresponding to $\mathsf{VisChase}(\Sigma'_{\mathsf{Source}}, \mathcal{M}')$. We are trying to build $\mathcal{D}'_0, \ldots$ such that there exists for all i there exists j such that $\mathcal{D}_i = \Pi(\mathcal{D}'_j)$, and $h(x) = c_{\mathsf{Crit}}$ implies $x = c_{\mathsf{Crit}}$.

  We prove by induction:

  - $\mathcal{D}_0$ is composed of witnesses of $\mathcal{M}$ and $\mathcal{D}'_0$ of witnesses of $\mathcal{M}'$. We build $\mathcal{D}'_1, \ldots, \mathcal{D}'_j$ such that each $\mathcal{D}'_i$ is obtained by firing the i-th rule $R_\phi(\vec{x}, \vec{y}) \rightarrow \phi(\vec{x}, \vec{y})$.
  - Let us suppose that $\mathcal{D}_i = \Pi(\mathcal{D}'_j)$ and $\mathcal{D}_{i+1}$ is obtained by firing a rule σ; σ could have been fired on $\mathcal{D}'_j$ and thus we can build $\mathcal{D}'_{j+1}$ such that $\mathcal{D}_{i+1} = \Pi(\mathcal{D}'_{j+1})$.
  - When $\mathcal{D}_{i+1}$ is obtained by merging values then it means that we have $\phi(\vec{x}, \vec{y})$ holding in $\mathcal{D}_i$ and thus $\phi(\vec{x}, \vec{y})$ holding in $\Pi(D'_j)$ therefore we could use the rule $\phi(\vec{x}, \vec{y}) \rightarrow R_\phi(\vec{x}, \vec{y})$ followed by an unification on R_ϕ. Therefore we can build $\mathcal{D}'_{j+1} = \mathcal{D}'_j \cup \{R_\phi(\vec{x}, \vec{y})\}$ and $\mathcal{D}'_{j+2}$ such that $\mathcal{D}_{i+1} = \Pi(\mathcal{D}_{j+2})$.
- For the direction $\mathsf{Disclose}(\Sigma'_{\mathsf{Source}}, \mathcal{M}', p)$ implies $\mathsf{Disclose}(\Sigma_{\mathsf{Source}}, \mathcal{M}, p)$ we start by noticing that, without loss of generality, we can suppose that the sequence $\mathcal{D}'_0, \ldots$ of $\mathsf{VisChase}(\Sigma'_{\mathsf{Source}}, \mathcal{M}')$ starts by firing each rule $R_\phi(\vec{x}, \vec{y}) \rightarrow \phi(\vec{x}, \vec{y})$ (it is always possible to generate more facts) and then we create $\mathcal{D}_0, \ldots$ such that for all i big enough there exists j such that $\mathcal{D}_j = h(\Pi(\mathcal{D}'_i))$
  - Once all rules $R_\phi(\vec{x}, \vec{y}) \rightarrow \phi(\vec{x}, \vec{y})$ have been fired, we see that we obtain an instance isomorphic to $\mathcal{D}_0$.
  - When $\mathcal{D}_j = h(\Pi(\mathcal{D}'_i))$ and $\mathcal{D}'_{i+1}$ is obtained through a merge step, it means that we had $\mathcal{D}'_i \models R_\phi(\vec{x}, \vec{y})$ but we easily see by induction that this means that we had $\mathcal{D}_j \models h(\phi(\vec{x}, \vec{y}))$ and thus that we can also perform the merge step on $\mathcal{D}_j$
  - When $\mathcal{D}'_{i+1}$ is obtained through a rule, it is either a rule in Σ_{Source} that we can reproduce in $\mathcal{D}_j$ or it is a rule $\phi(\vec{x}, \vec{y}) \rightarrow R_\phi(\vec{x}, \vec{y})$. In this latter case, we don't have anything to do as $R_\phi(\vec{x}, \vec{y})$ will be discarded by Π.

  Now, we also see that j will grow as i grows since except for rules $\phi(\vec{x}, \vec{y}) \rightarrow R_\phi(\vec{x}, \vec{y})$, our j increases. Therefore at the limit we have that $\mathsf{VisChase}(\Sigma'_{\mathsf{Source}}, \mathcal{M}') \models p$ implies $\mathsf{VisChase}(\Sigma_{\mathsf{Source}}, \mathcal{M}) \models p$.

□

Corollary 4. $\mathsf{Disclose}_{\mathsf{C}}(\mathsf{GTGD}, \mathsf{GuardedMap})$ *reduces to* $\mathsf{Disclose}_{\mathsf{C}}(\mathsf{GTGD}, \mathsf{ProjMap})$.

A.3 More details for the proof of Corollary 2

We recall the statement of Corollary 2:

If we fix the maximal arity of relations in the schema, then $\mathsf{Disclose}_{\mathsf{C}}(\mathsf{GTGD}, \mathsf{GuardedMap})$ is in EXPTIME.

We now fill in the details of the proof sketch in the body.

Reducing to ProjMap Using Corollary 4, we can reduce the problem to $\mathsf{Disclose}_{\mathsf{C}}(\mathsf{GTGD}, \mathsf{ProjMap})$. We now show that this latter problem is in EXPTIME.

Reducing to two atoms in the body of TGDs Given a set of GTGDs Σ_{Source} and a set of maps $\mathcal{M} \in \mathsf{IncDep}$ we now reduce $\mathsf{Disclose}(\Sigma_{\mathsf{Source}}, \mathcal{M}, p)$ to $\mathsf{Disclose}(\Sigma'_{\mathsf{Source}}, \mathcal{M}, p)$ where each GTGD in $\Sigma'_{\mathsf{Source}}$ holds at most two conjuncts in the rule body.

$\Sigma'_{\mathsf{Source}}$ is composed by applying the following process for each GTGD $\phi(\vec{x}) \rightarrow \exists \vec{y}\ R(\vec{t}) \in \Sigma_{\mathsf{Source}}$. The constraint $\phi(\vec{x}) \rightarrow \exists \vec{y}\ R(\vec{t})$ is guarded, therefore we can select a guarding conjunct $G_\phi(\vec{x})$ such that $\phi(\vec{x}) = G_\phi(\vec{x}) \wedge Q_1(\vec{x}) \wedge \cdots \wedge Q_k(\vec{x})$. When $k \leq 1$ we simply add $\phi(\vec{x}) \rightarrow \exists \vec{y}\ R(\vec{t})$ to $\Sigma'_{\mathsf{Source}}$. When $k > 1$, we rewrite this constraint by introducing k predicates $R_1, \ldots, R_k$, while producing the following constraints $G_\phi(\vec{x}) \wedge Q_1(\vec{x}) \rightarrow R_1(\vec{x})$ and for $1 \leq i \leq k-1$: $R_i(\vec{x}) \wedge Q_{i+1}(\vec{x}) \rightarrow R_{i+1}(\vec{x})$. Finally we also add $R_k(\vec{x}) \rightarrow \exists \vec{y}\ R(\vec{t})$. It is easy to see that this new problem is equivalent because each constraint in Σ_{Source} is implied by its corresponding constraints in $\Sigma'_{\mathsf{Source}}$ and if we look at the result of the visible chase, the only fact derived from a $R_k(\vec{x})$ are facts $R(\vec{y})$ such that $\phi(\vec{x})$.

Rewriting in PTIME Now that maps are ProjMaps and each GTGD has at most two atoms in their body, we can apply the rewriting presented in Theorem 2. Notice that each GTGD will be rewritten to a bounded number of GTGDs, and the rewriting of the maps will be trivial. Since query entailment with GTGDs is EXPTIME when the arity is bounded we can conclude the proof.

A.4 Proof of Theorem 3: more efficient reduction to entailment for LTGD source constraints and atomic mappings

Recall the statement of Theorem 3, which concerns the application of the rewriting algorithms $\mathsf{CritRewrite}_{\text{PTIME}}(\Sigma_{\mathsf{Source}})$ for LTGD source constraints Σ_{Source}, and the algorithm $\mathsf{CritRewrite}_{\text{PTIME}}(\mathcal{M})$ for atomic mappings $\mathcal{M}$:

$\mathsf{Disclose}(\Sigma_{\mathsf{Source}}, \mathcal{M}, p)$ holds exactly when there is a $Q_{\mathsf{Annot}} \in \mathsf{CritRewrite}(Q)$ such that $\mathsf{Hide}_{\mathcal{M}}(\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}})$ entails Q_{Annot} w.r.t. to the constraints

$$\mathsf{CritRewrite}_{\mathrm{PTIME}}(\Sigma_{\mathsf{Source}}) \cup \mathsf{CritRewrite}_{\mathrm{PTIME}}(\mathcal{M}) \cup \mathsf{IsCrit}(\mathcal{M})$$

Let $\Sigma_{\mathsf{simple}} = \mathsf{CritRewrite}_{\mathrm{PTIME}}(\Sigma_{\mathsf{Source}}) \cup \mathsf{CritRewrite}_{\mathrm{PTIME}}(\mathcal{M}) \cup \mathsf{IsCrit}(\mathcal{M})$ and Σ_{PTIME} be the constraints posed in Theorem 3. By Theorem 2, it is enough to show that query entailment involving Σ_{PTIME} is equivalent to entailment involving Σ_{simple}.

In one direction, suppose that I is a counterexample to entailment involving Σ_{simple}. We fire the rules generating atoms $B_{e,f}$ to get instance I'. We claim that the constraints of Σ_{PTIME} hold. Clearly, the rules generating atoms $B_{e,f}$ hold. Further, by construction, for any e, f if $B_{e,f}$ holds exactly when there is an annotation We now consider the rule $B_{e_h,f_h}(\vec{x}) \rightarrow \exists \vec{z}\, H(\vec{z})$. Considering a $\vec{c}$ such that $B_{e_h,f_h}(\vec{c})$ holds, we want to claim that there is an annotation Annot such that $B_{\mathsf{Annot}}(\vec{c})$ holds.

Recall that each e_i, f_i is associated with some variable v that occurs as both x_{e_i} and x_{f_i} in $B(\vec{x})$. If $B_{e_i,f_i}(\vec{c})$ holds, we know that either $c_{e_i} = c_{f_i}$ or $\mathsf{IsCrit}(c_{e_i}) \wedge \mathsf{IsCrit}(c_{f_i})$ holds. If the latter happens, then we add the variable v to our annotation. We can then verify that $B_{\mathsf{Annot}}(\vec{c})$ holds.

Since we are assuming that the corresponding constraint of Σ_{simple} holds in I, we can conclude that $I', \vec{c} \models \exists \vec{z}\, H(\vec{z})$. From this we see that I' is a counterexample to the entailment involving Σ_{PTIME}.

In the other direction, let I' be a counterexample to the entailment for the constraints in Σ_{PTIME}. We claim that the constraints of Σ_{simple} hold of I'. For constraints corresponding to source constraints with no repeated variables in the body, this is easy to verify, so we concentrate on constraints deriving from source constraints that do have repeated variables in the body.

Each of these constraints is of the form $B_{\mathsf{Annot}}(\vec{x}) \rightarrow \exists \vec{z}\, H(\vec{z})$ for some annotation Annot. Fix a $\vec{c}$ such that $B_{\mathsf{Annot}}(\vec{c})$ holds. We claim that $B_{e,f}(\vec{c})$ holds for all $(e, f) \in P$. We prove this by induction on the position of (e, f) in the ordering of pairs in P. Each (e, f) corresponds to some variable v that is repeated. If v is in Annot, then $B_{\mathsf{Annot}}(\vec{c})$ implies that $\mathsf{IsCrit}(c_e) \wedge \mathsf{IsCrit}(c_f)$ hold. Using the corresponding rule and the induction hypothesis we conclude that $B_{e,f}(\vec{c})$ holds. If v is not in Annot then $B_{\mathsf{Annot}}(\vec{c})$ implies that $c_e = c_f$. Using the other rule generating $B_{e,f}$ in Σ_{PTIME}, as well as the induction hypothesis, we conclude that $B_{e,f}(\vec{c})$ holds. This completes the inductive proof that $B_{e,f}(\vec{c})$ holds. Now using the corresponding constraint of Σ_{PTIME} we conclude that $I', \vec{c} \models \exists \vec{z}\, H(\vec{z})$. Since the constraints of Σ_{simple} hold, I' is also a counterexample to the entailment involving Σ_{simple}.

A.5 More details in proof of Theorem 4: upper bounds for LTGD source constraints and atomic maps

Recall the statement of Theorem 4

The problem $\mathsf{Disclose}_{\mathsf{C}}(\mathsf{LTGD}, \mathsf{AtomMap})$ is in ExpTime. If the arity of relations in the source schema is bounded, then the complexity drops to NP. If further the query is atomic, the problem is in PTime.

We now give more details on the proof. As mentioned in the body, is sufficient to get an ExpTime algorithm for the entailment problem produced by Theorem 3, since then we can apply it to each p_{Annot} in ExpTime. The constraints in $\mathsf{CritRewrite}_{\mathrm{PTIME}}(\Sigma_{\mathsf{Source}}) \cup \mathsf{CritRewrite}_{\mathrm{PTIME}}(\mathcal{M})$ are Guarded TGDs that are not necessarily LTGDs. But the bodies of these guarded TGDs consist of a guard predicate and atoms over a fixed "side signature", namely the unary predicate IsCrit. We can apply now the *linearization technique*, originating in [Gottlob *et al.*, 2014] and refined in [Amarilli and Benedikt, 2018a]. Given a side signature $\mathcal{S}_{\mathsf{Side}}$ this is an algorithm that converts an entailment problem involving ta set of non-full IncDeps and Guarded TGDs using $\mathcal{S}_{\mathsf{Side}}$, producing an equivalent entailment problem involving the same query, but only LTGDs. Further:

- The algorithm runs in ExpTime in general, and in PTime when the arity of the relations in the input is fixed
- The algorithm does not increase the arity of the signature, and thus the size of each output LTGD is polynomially-bounded in the input.

See also Appendix G of [Amarilli and Benedikt, 2018b] for a longer exposition of the linearization technique. Thus for general arity, we can use this algorithm to get an entailment problem with the same query, a data set exponentially bounded in the input data I' and a set of LTGDs, each polynomially-sized in the inputs. By applying a standard first-order query-rewriting algorithm to the query, we reduce this problem to evaluation of a union of conjunctive queries get a UCQ Q' on I'. The size of each conjunct in Q' is polynomially-bounded in the inputs, and so each conjunct C can be evaluated in time $|I'|^{|C'|}$, giving an ExpTime algorithm in total.

For fixed arity, we apply the same algorithm to get an entailment problem using IncDeps of bounded arity, which is known [Johnson and Klug, 1984] to be solvable in NP. Further, when the query is atomic, entailment with IncDeps is in PTime.

A.6 Proof of Theorem 5: disclosure for UID source constraints and ProjMap is PTime

We prove that when the source constraints are UIDs and the mappings are projections, disclosure analysis is in PTime. By Theorem 1, it suffices to show that the problem $\mathsf{HOCWQ}(\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}}, \Sigma_{\mathsf{Source}} \cup \Sigma_{\mathcal{M}}(\mathcal{M}), p, \mathcal{G}(\mathcal{M}))$ is PTime. We will thus first reduce this problem a problem $\mathsf{QEntail}(\mathcal{D}, \Sigma, p)$ where Σ is composed of UID constraints and $\mathcal{D}$ is composed of a single unary fact $\mathsf{IsCrit}(c_{\mathsf{Crit}})$.

Reachable predicates. We define the entailment graph over a set of IncDep constraints Σ. In this graph, nodes correspond to predicates and there is an edge $P \rightarrow R$ for each constraint $P(\vec{x}) \rightarrow R(\vec{y})$. Given an initial set of facts $\mathcal{D}$, one can compute the set $\mathsf{Reachable}(\Sigma, \mathcal{D})$ of entailed predicates. This set is defined as the set of predicates reachable in the entailment graph starting from the predicates appearing in $\mathcal{D}$.

Visible position graph. In studying tuple-generating dependencies, one often associates a set of dependencies with a graph whose edges represent the flow of data from one relation to another via the dependencies. See, for example the position graph used in defining the class of weakly acyclic sets of TGDs [Fagin *et al.*, 2005].

We develop another such graph, the *visible position graph* associated with a set of source constraints and mappings. The nodes are the pairs (P, i) where P is a predicate, $1 \leq i \leq ar(P)$ and there is an edge $(P, i) \to (R, j)$ when we have an IncDep (either a source constraint or a mapping rule) $P(\vec{x}) \to \exists \vec{y}\ R(\vec{t})$ with $x_i = t_j$. We refer to a node in this graph as a *position*. A position of a relation in the source schema is said to be *visible* if there is a path from (P, i) to a node (R, j) such that R belongs to the global schema. Another other position is said to be *invisible*. We see that when a position (P, i) is visible then for any fact $P(\vec{c})$ that holds in a possible world for $\mathsf{HOCWQ}(\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}}, \Sigma_{\mathsf{Source}} \cup \Sigma_{\mathcal{M}}(\mathcal{M}), p, \mathcal{G}(\mathcal{M}))$ we must have $c_i = c_{\mathsf{Crit}}$.

Note that if we have $P(\vec{x}) \to \exists \vec{y}\ R(\vec{t})$, x_i is exported to t_j, and position j of R is visible, then position i of P is visible as well.

Reduction to entailment. Let $\Sigma = \Sigma_{\mathsf{Source}} \cup \Sigma_{\mathcal{M}}(\mathcal{M})$ and $\mathcal{D}_0 = \mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}}$. We will reduce the problem $\mathsf{HOCWQ}(\mathcal{D}_0, \Sigma, p, \mathcal{G}(\mathcal{M}))$ to the problem $\mathsf{QEntail}(\mathcal{D}'_0, \tilde{\Sigma}, \tilde{p})$, where $\tilde{\Sigma} = \Sigma_{reach} \cup \Sigma_1 \cup \Sigma_{c_{\mathsf{Crit}}}$ is a set of UIDs, and $\tilde{p}$ is a CQ. Our reduction proceeds as follows:

- We transform the schema for sources creating a predicate $\tilde{P}$ for each source predicate P, where the arity of $\tilde{P}$ is the arity of P minus the number of positions (P, i) that are visible.
- $\mathcal{D}'_0 = \{\mathsf{IsCrit}(c_{\mathsf{Crit}})\}$.
- Σ_{reach} is built as the set of constraints $\mathsf{IsCrit}(w) \to \exists \vec{x}\ P(\vec{x})$ where $\vec{x}$ are fresh distinct variables and $P \in \mathsf{Reachable}(\mathsf{Hide}_{\mathcal{M}}(\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}}), \Sigma)$.
- Σ_1 is formed from the set of constraints $P(\vec{x}) \to \exists \vec{y}\ R(\vec{t}) \in \Sigma$ such that there is an exported variable lying in an invisible position of $P(\vec{x})$. For each such constraint, Σ_1 contains the constraint $\tilde{P}(\vec{x}^*) \to \exists \vec{y^*}\ \tilde{R}(\vec{t^*})$ where $\vec{x}^*$ denotes the projection of $\vec{x}$ to the invisible positions of P, and similarly for $\vec{y^*}$ and $\vec{t^*}$.
- $\Sigma_{c_{\mathsf{Crit}}}$ is formed from constraints $P(\vec{x}) \to \exists \vec{y}\ R(\vec{t}) \in \Sigma$ such that $P \in \mathsf{Reachable}(\mathsf{Hide}_{\mathcal{M}}(\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}}), \Sigma)$ and there is an exported variable x lying in a visible position of $P(\vec{x})$, exported to an invisible position of R. For each such constraint $\Sigma_{c_{\mathsf{Crit}}}$ includes the constraint $\mathsf{IsCrit}(x) \to \exists \vec{y}^*\ \tilde{R}(\vec{t}^*)$ where $\vec{y}^*$ denotes the projection of $\vec{y}$ to the invisible positions of P and similarly for $\vec{t}^*$.
- the query $\tilde{p}$ is built from p by first replacing each conjunct $P(\vec{x})$ with its corresponding predicate $\tilde{P}(\tilde{\vec{x}})$, projecting out the visible positions. After this, for every variable x that occurred in p within both a visible and an invisible position, x is replaced by v, while we add a conjunct $\mathsf{IsCrit}(v)$.

Correctness of the reduction. The correctness of the reduction is captured in the following result:

Proposition 3. *For any source constraints Σ_{Source} consisting of IncDeps and $\mathcal{M}$ consisting of projection mappings, there is a disclosure over a schema $\mathcal{S}$ with constraints Σ_{Source} mappings $\mathcal{M}$ and secret query p if and only if $\mathsf{QEntail}(\mathcal{D}'_0, \tilde{\Sigma}, \tilde{p})$ holds.*

Proof. We start with the argument for the left to right direction. We let $\mathcal{D}'$ be a counterexample to the entailment $\mathsf{QEntail}(\mathcal{D}'_0, \tilde{\Sigma}, \tilde{p})$. By Theorem 9, we can assume that $\mathcal{D}'$ is formed by applying the chase procedure to $\mathcal{D}'_0$. In particular, each fact in $\mathcal{D}'$ can be assumed to use a predicate in $\mathsf{Reachable}(\mathsf{Hide}_{\mathcal{M}}(\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}}), \Sigma)$.

We show that there is an instance $\mathcal{D}$ that is a counterexample to

$$\mathsf{HOCWQ}(\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}}, \Sigma_{\mathsf{Source}} \cup \Sigma_{\mathcal{M}}(\mathcal{M}), p, \mathcal{G}(\mathcal{M}))$$

and thus (by Theorem 1) we cannot have a disclosure. We form $\mathcal{D}$ by filling out each visible position with c_{Crit}. We claim that $\mathcal{D}$ satisfies each source constraint $\sigma = P(\vec{x}) \to \exists \vec{y}\ R(\vec{t})$. Suppose that $P(\vec{c})$ holds in $\mathcal{D}$. Then $\tilde{P}(\vec{c}')$ holds in $\mathcal{D}'$, where $\vec{c}'$ projects $\vec{c}$ on to the invisible positions.

- First, suppose there is a variable x in an invisible position of $P(\vec{x})$ exported to an invisible position in $R(\vec{t})$. Then since $\mathcal{D}'$ satisfies Σ_1, we know that for some $\vec{d}$, $\tilde{R}(\vec{d})$ holds in $\mathcal{D}'$, By the definition of $\mathcal{D}$, we have that $R(\vec{d^*})$ holds, where $\vec{d^*}$ fills out each visible position with c_{Crit}. We can see that $R(\vec{d^*})$ is the required witness for $P(\vec{c})$.
- Next, suppose there is a variable x in a visible position j of $P(\vec{x})$ exported to an invisible position in $R(\vec{t})$. Then we must have $c_j = c_{\mathsf{Crit}}$. Since P is in $\mathsf{Reachable}(\mathsf{Hide}_{\mathcal{M}}(\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}}), \Sigma)$ and $\mathcal{D}'$ satisfies $\Sigma_{c_{\mathsf{Crit}}}$, we have $\tilde{R}(\vec{e})$ holding in $\mathcal{D}'$ for some $\vec{e}$, and hence $R(\vec{f})$ holding in $\mathcal{D}$ for some tuple where c_{Crit} fills all the visible positions. Thus σ holds in this case as well.
- Finally, note that a variable at an invisible position cannot be exported to a visible position. Therefore the only remaining case is the case where no variable has been exported. Since P is reachable, then R is also reachable therefore there is a constraints $\mathsf{IsCrit}(x) \to \exists \vec{y}^* R(\vec{y}^*) \in \Sigma_{\mathsf{Reachable}}$ and thus $\tilde{R}(\vec{d^*})$ holds in $\mathcal{D}'$

We next claim that the image of $\mathcal{D}$ under $\mathcal{M}$ agrees with $\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}}$.

- For every global schema predicate G, $G(c_{\mathsf{Crit}} \ldots c_{\mathsf{Crit}})$ occurs in the the image of $\mathcal{D}$ under $\mathcal{M}$. This follows easily from the fact that $\mathcal{D}'$ contains $\mathcal{D}'_0$.
- If $G(\vec{c})$ holds in the $\mathcal{M}$-image, then because each visible position was filled out with c_{Crit}, we must have each $c_i = c_{\mathsf{Crit}}$. Thus the result follows.

Note that from the preceding claims, we know that $\mathcal{D}$ is a possible world for $\mathsf{HOCWQ}(\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}}, \Sigma_{\mathsf{Source}} \cup \Sigma_{\mathcal{M}}(\mathcal{M}), p, \mathcal{G}(\mathcal{M}))$. Finally, we claim that $\mathcal{D}$ does not satisfy p.

- Suppose $\mathcal{D} \models p$ with homomorphism h as a witness. Since $\mathcal{D}$ is a possible world for $\mathsf{HOCWQ}(\mathcal{D}_0, \Sigma, p, \mathcal{G}(\mathcal{M}))$, for any variable v occurring in a visible position, $h(v) = c_{\mathsf{Crit}}$. Let h' be formed from the restriction of h to variables that occur in $\tilde{p}$, by mapping the additional variable v to c_{Crit}. Note that in $\mathcal{D}'$, $\mathsf{IsCrit}(c_{\mathsf{Crit}})$ holds. For this, we see that h' is a homomorphism witnessing that $\mathcal{D}' \models \tilde{p}$. This is a contradiction to the fact that $\mathcal{D}'$ is a counterexample to the entailment.

We now have argued that $\mathcal{D}$ is a counterexample to $\mathsf{HOCWQ}(\mathcal{D}_0, \Sigma, p, \mathcal{G}(\mathcal{M}))$, which completes the proof of the left to right direction.

For the other direction, suppose that $\mathcal{D}$ is a counterexample to $\mathsf{HOCWQ}(\mathcal{D}_0, \Sigma, p, \mathcal{G}(\mathcal{M}))$. Note that for any fact $R(\vec{c})$ over the source relations in $\mathcal{D}$, for any visible position i of R, we must have $c_i = c_{\mathsf{Crit}}$. Form $\mathcal{D}'$ by projecting each fact in $\mathcal{D}$ to the invisible positions of the relation. We will argue that $\mathcal{D}'$ is a counterexample to the entailment produced by the reduction.

- $\mathcal{D}$ should contain $\mathsf{IsCrit}(c_{\mathsf{Crit}})$ therefore $\mathcal{D}'$ extends $\mathcal{D}'_0$.
- The fact that $\mathcal{D}$ was a solution to $\mathsf{HOCWQ}(\mathcal{D}_0, \Sigma, p, \mathcal{G}(\mathcal{M}))$ also guarantees that for all reachable predicates P we have $\mathcal{D} \models \exists \vec{x}\ P(\vec{x})$ and thus $\mathcal{D}' \models \exists \vec{x}^*\ \tilde{P}(\vec{x}^*)$ and thus all constraints in $\Sigma_{\mathsf{Reachable}}$ are satisfied.
- Let us show that the constraints in Σ_1 are satisfied: fix a constraint $\sigma' \in \Sigma_1 = \tilde{P}(\vec{x}^*) \rightarrow \exists \vec{y}'\ \tilde{R}(\tilde{\vec{t}})$, derived from source constraint $\sigma = P(\vec{x}) \rightarrow \exists \vec{y}\ R(\vec{t})$. Fix a fact $F' = \tilde{P}(\vec{c}^*)$ in $\mathcal{D}'$. By definition of $\mathcal{D}'$, $\vec{c}^*$ extends to a $\vec{c}$ satisfying P in $\mathcal{D}$. Thus, since $\mathcal{D} \models \Sigma$, there is a fact $G = R(\vec{d})$ that holds in $\mathcal{D}$ with $d_i = c_j$ whenever $t_i = x_j$. We can project to the invisible positions to get a fact $G' = \tilde{R}(d_{j_1} \ldots d_{j_n})$ in $\mathcal{D}'$. We claim that G' is a witness for the satisfaction of σ' with respect to F'. Consider any variable x exported from F' to position j' of G' where x is mapped to value c in $\vec{c}^*$. Then in σ, x was exported to the corresponding invisible position j in $R(\vec{y})$, and from this we see that $d_j = c$ as required.
- Now consider a constraint $\sigma' \in \Sigma_{c_{\mathsf{Crit}}} = \mathsf{IsCrit}(x) \rightarrow \exists \vec{y}^*\ \tilde{R}(\tilde{\vec{t}}^*)$. Since $\mathsf{IsCrit}(x)$ holds only for $x = c_{\mathsf{Crit}}$ in $\mathcal{D}$, we only have to verify that $\tilde{R}(\vec{e}^*)$ holds for some $\vec{e}$ such that $e_\ell = c_{\mathsf{Crit}}$ (where ℓ is the position of x in $\vec{y}^*$). Let us suppose that σ' was derived from source constraint $\sigma = P(\vec{x}) \rightarrow \exists \vec{y}\ R(\vec{t})$ where j is the position of the exported in $\vec{y}$ and i is the position of the exported variable in $\vec{t}$. By the definition of $\Sigma_{c_{\mathsf{Crit}}}$, we know that P is a reachable predicate, and hence $P(\vec{d})$ must hold for some $\vec{d}$ in $\mathcal{D}$ and since d_j is visible we have $d_j = c_{\mathsf{Crit}}$. Because $\mathcal{D} \models \sigma$ we have $\tilde{R}(\vec{e})$ holds in $\mathcal{D}$ for some $\vec{e}$ such that $e_i = c_{\mathsf{Crit}}$ and thus $R(\vec{e}^*)$ is the required witness for σ'.
- Finally, we argue that $\mathcal{D}'$ does not satisfy $\tilde{p}$. Suppose by way of contradiction that $\mathcal{D}'$ satisfies $\tilde{p}$ via homomorphism h'. Note that the variables of p that do not occur in $\tilde{p}$ are those that occur only in visible positions within an atom of p. We extend h' to a mapping h from the variables of p to $\mathcal{D}$ by mapping each such variable x to c_{Crit}. We argue that h is a homomorphism of p to $\mathcal{D}$. Consider an atom $R(\vec{t}, \vec{t}')$ of p, where $\vec{t}$ correspond to the invisible positions. Suppose first that the corresponding atom of $\tilde{p}$ is of the form $\tilde{R}(\vec{t}^*)$ where $\vec{t}^*$ is obtained from $\vec{t}$ by replacing any variable shared with a visible position by v. We know that $\tilde{R}(h(t_1^*) \ldots h(t_j^*))$ holds in $\mathcal{D}'$ because h is a homomorphism. Thus $R(h(t_1^*), \ldots h(t_j^*), \vec{e})$ holds in $\mathcal{D}$ for some $\vec{e}$. By the properties of visible positions and the fact that $\mathcal{D}$ is a possible world for $\mathsf{HOCWQ}(\mathcal{D}_0, \Sigma, p, \mathcal{G}(\mathcal{M}))$, we see that each $e_i = c_{\mathsf{Crit}}$. Thus h not only preserves the atom $\tilde{R}(\vec{t}^*)$, but it also preserves the additional atom $\mathsf{IsCrit}(v)$, since $\mathsf{IsCrit}(c_{\mathsf{Crit}})$ holds in $\mathcal{D}'$. thus h is a homomorphism, contradicting the fact that $\mathcal{D}$ is a counterexample to $\mathsf{HOCWQ}(\mathcal{D}_0, \Sigma, p, \mathcal{G}(\mathcal{M}))$.

Since $\mathcal{D}'$ extends $\mathcal{D}_0$, satisfies the constraints $\tilde{\Sigma}$, and does not satisfy the query $\tilde{p}$, it is a counterexample to the entailment, completing this direction of the argument. □

Overview of PTIME algorithm for entailment with UIDs over a single fact. At this point we have restricted to a CQ entailment problem for a set of UIDs and a single fact. It was claimed in [Kikot *et al.*, 2011] that there is a polynomial time query rewriting for UIDs, and from this it would easily follow that our entailment problem is in PTIME (query evaluation is PTIME when these is a single fact). However later work (footnote on page 38 of [Bienvenu *et al.*, 2018]) refers to flaws in this argument, and says that polynomial rewritability is open. We therefore give a direct proof that such an entailment problems are in PTIME. This will proceed via several steps:

- A reduction to the case of "binary schemas": those where the arity of each predicate is at most 2.
- Query simplification, which will reduce the query to a connected acyclic query.
- Reduction to atomic entailment.

Reduction to binary schemas. We begin by using verbatim an idea of [Kikot *et al.*, 2011], reducing to the same problem but when the input schema is binary. We do this via a standard reduction of general arity reasoning to binary reasoning, introducing predicates $R_i(t, v)$ for every relation R of arity $n \geq 1$ and each $1 \leq i \leq n$; informally these state that v is the value in position i of n-tuple t. We also introduce a predicate $R_\exists(t)$ for each predicate R; informally this states that there is some tuple t in the predicate R. We translate each UID $B(\vec{x}) \rightarrow \exists \vec{y}\ H(\vec{t})$ exporting a variable x_i from position i to position j to a UID $B_i(t, x_i) \rightarrow \exists t'\ H_j(t', x_i)$. For each UID, $H(\vec{x}) \rightarrow B(\vec{y})$ that is not exporting a variable, we create a rule $H_\exists(t) \rightarrow \exists t'\ B_\exists(t')$. We also create rules $R_i(t, x) \rightarrow R_\exists(t)$ and $R_\exists(t) \rightarrow \exists x\ R_i(t, x)$ for each predicate R and $1 \leq i \leq n$ where n is the arity of R. Finally the query p is transformed into p' where each conjunct $R(x_1, \ldots, x_n)$ is transformed into the conjunction $R_1(t, x_1) \wedge \cdots \wedge R_n(t, x_n) \wedge R_\exists(t)$, for a fresh variable t.

Finally the database over the binary schema is built in the following way: for each fact $R(\vec{v})$ of the initial database, we create a fresh value t and we add the conjunct $R_i(t, x)$ for $1 \leq i \leq n$ where n is the arity of R and we also add $R_\exists(t)$. Further details can be found in [Kikot *et al.*, 2011]. Note that, in the resulting problem, each frontier-0 rule produced has a body with an atom over a unary predicate.

Proposition 4. *The transformation above preserves query entailment.*

Special form of the chase: annotated chase forest. In the case of UIDs the chase process applied to our single-fact instance $\mathcal{D}_0$ produces an in instance $\mathsf{Chase}_\Sigma(\mathcal{D}_0)$ that will be infinite. However, it has a special shape that we can exploit. For the remainder of this section, by $\mathsf{Chase}_\Sigma(\mathcal{D}_0)$ we consider an instance formed from a *restricted chase sequence*, in which a witness to a TGD $\phi(\vec{x}) \rightarrow \exists \vec{y}\, H(\vec{t})$ is added to instance $\mathcal{D}_i$ for binding $\vec{c}$ to $\vec{x}$ only if $\mathcal{D}, \vec{c} \models \phi(\vec{x}) \wedge \neg \exists \vec{y}\, H(\vec{t})$. It is known [Fagin *et al.*, 2005] that in Theorem 9 it suffices to consider such instances. The *annotated chase* is a node- and edge-labelled forest formed from $\mathsf{Chase}_\Sigma(\mathcal{D}_0)$ as follows:

- the nodes are the values of $\mathsf{Chase}(\mathcal{D})$
- the node label of a value v is the collection of unary predicates holding at v
- an edge labeled by fact F mentioning v_1 and v_2 connects a value v_1 to a value v_2 if F holds in $\mathsf{Chase}(\mathcal{D})$ and v_2 is generated in the chase step that produces F.

We can see that this graph is a forest where he roots are c_{Crit} (the value where $\mathsf{IsCrit}(c_{\mathsf{Crit}})$ holds) as well as some other trees rooted to reachable facts generated from frontier-0 dependencies and thus rooted at elements t where $R_\exists(t)$ holds for some R. Further, since the chase is restricted, we can see that this graph has the *unique adjoining label property*: for each v_1, for each predicate P, there cannot be two nodes v_2, v_2' adjacent to v_1 such that the edge e from v_1 to v_2 and e' from v_1 to v_2 both are labelled with the same predicate and have v_1 in the same position. Furthermore, the restricted chase also ensures that the forest is composed of at most one tree per predicate since all the roots that are produced needs to be different.

First query simplification: eliminating forking pairs. Given a CQ Q, a pair of distinct atoms A_1 and A_2 sharing the same predicate and a variable at the same position (i.e. $q_1 = R(x, z)$ and $q_2 = R(x, y)$ or $q_1 = R(z, x)$ and $q_2 = R(y, x)$) is a *forking pair* of Q. We say that a query Q is *non-forking* when there are no forking pairs.

Proposition 5. *If a CQ Q has a forking pair $A_1 = R(x, z)$ and $A_2 = R(x, y)$ and Q' is the query Q where the variable z is replaced with y, then* $\mathsf{QEntail}(\mathcal{D}_0, \Sigma, Q) = \mathsf{QEntail}(\mathcal{D}_0, \Sigma, Q')$

Proof. Let $\mathcal{D} = \mathsf{Chase}_\Sigma(\mathcal{D}_0, \Sigma)$. If p' holds in $\mathcal{D}$, then clearly the same holds of p. Conversely suppose p holds in $\mathcal{D}$ via homomorphism h, and suppose $h(y) \neq h(z)$. This gives us a violation of the unique adjoining label property. □

Applying the proposition above, we can assume that Q is non-forking. Without loss of generality, we can also assume that Q is connected (otherwise we can test the entailment of each connected part).

Second simplification: reducing to acyclic queries. The *CQ-graph* of a CQ Q is the node- and edge-labelled graph whose nodes are the variables of Q and whose edges are labelled with atoms of Q such that:

- an edge between variables labelled with x and y is labelled with the binary atoms containing both x and y;
- a node x is labelled with the set of unary predicates in Q containing x.

The CQ-graph said embedded in some annotated chase forest T if there is a homomorphism $h : \mathcal{A} \rightarrow T$ preserving edges, i.e. if there is an edge x to y labeled with $R(a, b)$ then T should contain an $\mathcal{A}(x)$ to $\mathcal{A}(y)$ labeled with $R(\mathcal{A}(a), \mathcal{A}(b))$ and nodes, i.e. if there is a predicate $P(x)$ on the node x then there should be $P(\mathcal{A}(x))$ in T. The homomorphism h is called an embedding of Q in T.

It is immediate from the completeness of the chase procedure that for any annotated chase forest T for $\mathsf{Chase}_\Sigma(\mathcal{D})$, a query is entailed if and only if its CQ-graph is embedded in T. Our reduction to the case of a CQ with acyclic CQ-graph will depend heavily on the following observation:

Proposition 6. *Any embedding of a connected and non-forking CQ Q into an annotated chase forest for* $\mathsf{Chase}_\Sigma(\mathcal{D}_0)$ *must be injective.*

Proof. Let Q be a connected and non-forking and let h be an embedding. Let us prove by induction on the size of the path between x and y that $h(x) \neq h(y)$ when $x \neq y$.

Two neighboring nodes cannot be sent to the same value. For a path of size 2, if we have z such that x, z, y forms a path in the CQ-graph of Q then $h(x)$ has to be different than $h(y)$ otherwise the label from x to z and from z to y would be the same and there would be a forking pair in Q.

Let $x = p_1, p_2, \ldots, p_k = y$ with $k \geq 4$ be a path in the CQ-graph between x and y. By induction the $h(p_i)$ for $i < k$ are all distinct and thus the distance between $h(x)$ and $h(p_{k-1})$ is at least $k - 2$ hence $h(y)$ is at least at distance $k - 3 > 0$ of $h(x)$. □

Our reduction to the acyclic case follows immediately:

Corollary 5. *If a connected non-forking CQ Q is entailed by Σ over then the CQ-graph of Q is acyclic.*

Proof. Q is entailed The image of the CQ-graph through the injective homomorphism is a forest. □

Determining entailment for acyclic connected graphs. We now give the final setp in our algorithm, which deals with deciding entailment of a connected, non-forking query Q, which by Corollary 5 must have an acyclic CQ-graph. Given an acyclic connected undirected graph and any vertex v of the graph, we can direct it be a tree with v as the root. Thus for such a Q having n variables, the *tree arrangements* are the n possible ways to root the CQ-graph of the query Q. We are particularly interested in arrangements of Q where the directionality from parent to child reflects the entailment structure relative to Σ between atoms in the query. A tree arrangement $\mathcal{A}$ of Q is *faithfully entailed* if for every variable y in Q with parent x in the tree, there is an atom A containing x and not containing y such that $A \wedge \Sigma$ entails $\exists y\, B_{x,y}$, where $B_{x,y}$ is

the conjunction of all atoms whose variables are contained in $\{x, y\}$; in the case that y is the root, we require Σ alone to entail $\exists y\ B_{x,y}$.

In a faithfully-entailed tree arrangement, the conjunction of atoms holding at the root of the tree entails the existence of the whole tree. We can further find a single atom that entails the whole tree. A *root-generating atom* of a tree arrangement is an atom A (not necessarily in Q) containing the root variable r, such that $A \wedge \Sigma$ generates all atoms mentioning r.

Proposition 7. *A faithfully entailed tree arrangement for Q must have a root-generating atom.*

Proof. We know that Q must hold in the chase of the initial fact under Σ, and by Proposition 6 we know that there is an injective homomorphism h from Q to the chase. Consider the point in the chase process where value $h(r)$ is first generated. This occurs by firing some rule with an atom, where the head has either a binary atom $A(x, y)$ or a unary atom $B(x)$. We consider the case where the atom is binary, and where the generated atom is $A(h(r), s)$. In this case the fact $A(h(r), s)$ must generate every fact containing r. Thus we can take the atom $A(r, w)$, where w is a fresh variable, as a root-generating atom. The case of unary atoms and the case where r is in the second position of the fact is similar. □

Given a tree arrangement T of Q and variable x of Q, T_x denotes the the restriction of T to the variables that are descendants of x in T.

The main idea of our PTIME algorithm is that it suffices to descend through the tree arrangement, checking some entailments for each parent-child pair in isolation.

Proposition 8. *There is a* PTIME *algorithm taking as input a variable x in a CQ Q, a tree arrangement of Q, and an atom A containing x such that the existential quantification of A is entailed by Σ, and determining whether T_x is faithfully entailed and A is a root-generating atom.*

Proof. We first check whether A is a root-generating atom, using a PTIME inference algorithm for UIDs [Cosmadakis *et al.*, 1990]. We then consider each child y of x in the tree arrangement. We know that there is exactly one conjunct B containing x and y. We check whether A entails $\exists y\ B$, and then call the algorithm recursively for y and B. If each recursive call succeeds, the algorithm succeeds. □

From the prior proposition we get a PTIME algorithm for the arrangement as a whole:

Proposition 9. *There is a* PTIME *algorithm taking a tree arrangement of CQ Q, and an atom A containing the root of the arrangement, and determines whether the whole tree arrangement can be faithfully entailed and A is a root-generating atom.*

Proof. We first need to check that A is entailed, which amounts to checking that $\Sigma \models \mathsf{IsCrit}(c) \rightarrow \exists y\ A$. As before this can be done using [Cosmadakis *et al.*, 1990]. We then utilize the algorithm of Proposition 8. □

Note that Proposition 9 gives a polynomial time algorithm for checking whether a tree arrangement can be faithfully entailed. We can apply the algorithm of the proposition with every possible unary and binary atom A containing the root variable. In the binary case, we consider all atoms containing the root variable and an additional fresh variable.

Putting it all together. Putting together our reduction to UID-entailment (Proposition 3), our schema simplification (Proposition 4) the query simplifications (the reduction to connected CQs, Proposition 5, and Corollary 5), and our PTIME algorithm for simplified queries (Proposition 9) we obtain the proof of Theorem 5.

B Detailed proofs from Section 4: lower bounds for disclosure

B.1 Proof of the first part: Theorem 6: 2ExpTime-hardness for IncDep and GuardedMap without arity bound

Recall the first part of Theorem 6:

$\mathsf{Disclose}_{\mathsf{C}}(\mathsf{IncDep}, \mathsf{GuardedMap})$ is 2ExpTime-hard.

Recall that Theorem 1 relates disclosure to a HOCWQ problem on a very simple instance. Also recall from Section 3 the intuition that such a problem amounts to a classical entailment problem for a CQ over a very simple instance, using the source dependencies and SCEQrules: of the form $\phi(\vec{x}) \rightarrow x = c_{\mathsf{Crit}}$, where ϕ will be the body of a mapping. We show here how to simulate the run of an alternating ExpSpace Turing machine $\mathscr{T}$ without explicitly using SCEQrules, instead using inclusion dependencies as source constraints coupled with guarded mappings. An alternating Turing machine $\mathscr{T}$ is a 6-tuple $(Q, \Sigma, \delta_\alpha, \delta_\beta, q_0, g)$ where:

- Q is the finite set of states
- Σ is the finite tape alphabet
- δ_α and δ_β are functions from $Q \times \Sigma$ to $Q \times \Sigma \times \{L, R\}$
- $q_0 \in Q$ is the initial state
- g is a function from Q to $\{accept, reject, \forall, \exists\}$ that specifies the type of each state.

We assume that $\mathscr{T}$ always alternates between existential and universal states, and that there is a unique final state, that can be reached only if the head is in the first cell and contains a specific symbol. All of these assumptions can be made without loss of generality. If $\mathscr{T}$ is in a configuration where whose state q is such that $g(q) = accept$, the configuration is said to be accepting. If $\mathscr{T}$ is in a configuration where whose state q is such that $g(q) = \forall$, the configuration is said to be accepting if its α and β successors (obtained after applying δ_α or δ_β) are accepting. If $\mathscr{T}$ is in a configuration whose state q is such that $g(q) = \exists$, the configuration is said to be accepting if its α-successor or its β-successor is accepting. A more thorough introduction to Turing machines can be found in [Papadimitriou, 1994].

We first present the reduction, and show its correctness in the next subsection.

B.2 The Reduction

We will create constraints and mappings that will serve to perform the following tasks:

- generate addresses for cells of $\mathscr{T}$ in such a way that one can check whether two addresses are consecutive in a guarded way. The same addresses will be used for all the configurations. This will be done by a mapping creating k copies of two individuals that represent 0 and 1, along with inclusion dependencies that perform permutations and generate 2^k addresses;
- encode the content of a cell, the position of the head, and the state of the head: for each cell, we store a vector whose length is the size of $(\Sigma \cup \{\flat\}) \times (Q \cup \bot)$. Each position corresponds to an element (l, s) of that set; we will arrange that the position contains c_{Crit} if and only if the cell contains l, and either the head is over that cell and is in state s, or the head is not over that cell and $s = \flat$. All values are first freshly instantiated by inclusion dependencies, and mappings are then responsible for unifying the correct positions with c_{Crit};
- ensure that the tape that is associated with a successor of a configuration can be obtained by a transition of the Turing machine: this is also performed by using a mapping to enforce the correct positions of the cell to be unified with c_{Crit};
- check that configurations are accepting: this is the case either when the corresponding tape is in a final accepting state, or when it is in an existential state and one of the two successor configurations is accepting, or it is in a universal state, and both successor configurations are accepting.

Let us describe the source signature. For each predicate, we will explain what feature of the ATM $\mathscr{T}$ it should represent in the appropriate instance generated by the constraints. By "the appropriate instance", we mean the visible chase of the initial instance over the source constraints and mappings: this was introduced after Theorem 9, and it was noted that it is the canonical instance for the source and targets to consider for disclosure.

We use $\mathbf{y^{1,k}}$ to represent a tuple $(y^1, \ldots, y^k)$, and $\mathbf{y^k}$ to represent the tuple $(y, \ldots, y)$ of size k.

- $\mathsf{Children}_\forall(c, c_\alpha, c_\beta, ac, ac_\alpha, ac_\beta, \mathbf{y_0^{1,k}}, \mathbf{y_1^{1,k}}, r, z, y_0, y_1)$. The intended meaning is that a configuration c is universal and has as children c_α and c_β, and that the acceptance bit of c is ac, of c_α is ac_α and of c_β is ac_β. The last four positions are placeholders: r for the root of the tree of configurations, z for c_{Crit}, y_0 for a value representing 0 and y_1 for a value representing 1.
- $\mathsf{Children}_\exists(c, c_\alpha, c_\beta, ac, ac_\alpha, ac_\beta, \mathbf{y_0^{1,k}}, \mathbf{y_1^{1,k}}, r, z, y_0, y_1)$: same intended meaning, except that c is existential.
- $\mathsf{Cell}(c_p, c_n, \mathbf{y^{1,k}}, \mathbf{v}, \mathbf{v_{prev}}, \mathbf{v_{next}}, r, z, y_0, y_1)$ with intended meaning that the cell of address $\mathbf{y^{1,k}}$ of the tape represented by c_n has a content represented by $\mathbf{v}$, while the previous cell has a content represented by $\mathbf{v_{prev}}$ and the next cell has content represented by $\mathbf{v_{next}}$. The last four positions are placeholders for the root of the tree of configurations, c_{Crit}, a value representing 0 and a value representing 1.
- $\mathsf{Cell}_i^c(c, \mathbf{y^{1,k}}, x, z, y_0, y_1)$ with intended meaning that the cell of address $\mathbf{y^{1,k}}$ in configuration c contains x at the i^{th} position of the representation of its content. Cell_i^p and Cell_i^n play similar roles for the cell before and after the cell of address $\mathbf{y^{1,k}}$.
- GenAddr is an auxiliary predicate used to generated an exponential number of addresses.
- $\mathsf{succ}_\alpha(c_p, c_n)$ states that c_n is the α-successor of c_p (and similarly for β)

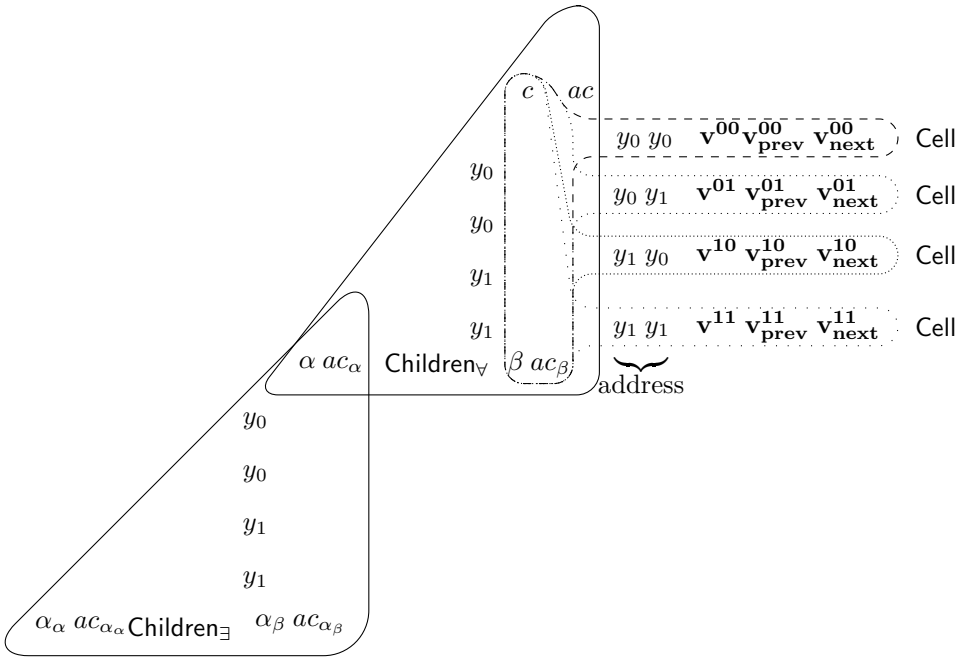

Figure 1: The generated structure

Below we will always use the symbol $\mathscr{Q}$ to range over $\{\forall, \exists\}$.

The structure generated by the inclusion dependencies is represented Figure 1. Atoms are represented by geometric shapes in the inside of which are arguments (some are omitted to ease the reading). The $\mathsf{Children}_{\mathscr{Q}}$ atoms form a tree shaped structure, and induce a tree structure on the configuration identifiers: for instance, c is the parent of α and β. Cell atoms are associated with a configuration identifier (for instance, those represented are associated with β), and has the parent configuration identifier to ensure guardedness of the mappings used in the following reduction. Note that the elements used to describe the cell's addresses (y_0 and y_1) also appear in the $\mathsf{Children}_{\mathscr{Q}}$ atoms, to ensure guardedness.

Initialization We first define a mapping $\mathcal{T}_{\mathsf{Init}}(x)$, introducing some elements in the visible chase. The definition of this mapping is:

$$\mathsf{Children}_\exists(c_{root}, c_\alpha, c_\beta, ac_{root}, ac_\alpha, ac_\beta, \mathbf{y_0^{1,k}}, \mathbf{y_1^{1,k}}, c_{root}, x, y_0^0, y_1^0)$$

Generation of the tree of configuration α-successors have themselves α- and β-successors, and are existential if their parent is universal:

$$\mathsf{Children}_\forall(c, \alpha, \beta, ac, ac_\alpha, ac_\beta, \mathbf{y_0^{1,k}}, \mathbf{y_1^{1,k}}, r, z, y_0, y_1) \rightarrow \exists \alpha_\alpha, \alpha_\beta, ac_{\alpha_\alpha}, ac_{\alpha_\beta}$$

$$\mathsf{Children}_\exists(\alpha, \alpha_\alpha, \alpha_\beta, ac_\alpha, ac_{\alpha_\alpha}, ac_{\alpha_\beta}, \mathbf{y_0^{1,k}}, \mathbf{y_1^{1,k}}, r, z, y_0, y_1)$$

And similarly for $\mathsf{Children}_\exists$ and for the β-successor.

Universal and Existential Acceptance Condition If both successors of a universal configuration n are accepting, so is n. We create a mapping $\mathcal{T}_\forall(x)$ with definition:

$$\mathsf{Children}_\forall(c, \alpha, \beta, x, z, z, \mathbf{y_0^k}, \mathbf{y_1^k}, r, z, y_0, y_1)$$

If the α-successor of an existential configuration n is accepting, so is n. We create a mapping $\mathcal{T}_{\exists,\alpha}(x)$ with definition:

$$\mathsf{Children}_\exists(c, \alpha, \beta, x, z, ac_\beta, \mathbf{y_0^k}, \mathbf{y_1^k}, r, z, y_0, y_1)$$

We create a similar mapping $\mathcal{T}_{\exists,\beta}$ for the β-successor.

Tape Representation and Consistency of Tapes We now focus on the representation of the tape and its consistency. We generate 2^k addresses and associated values:

$$\mathsf{Children}_{\mathscr{Q}}(c, \alpha, \beta, ac, ac_\alpha, ac_\beta, \mathbf{y_0^{1,k}}, \mathbf{y_1^{1,k}}, r, z, y_0, y_1) \rightarrow \mathsf{GenAddr}(c, \alpha, \mathbf{y_0^{1,k}}, \mathbf{y_1^{1,k}}, r, z, y_0, y_1)$$

$$\mathsf{Children}_{\mathscr{Q}}(c, \alpha, \beta, \mathbf{y_0^{1,k}}, \mathbf{y_1^{1,k}}, r, z, y_0, y_1) \rightarrow \mathsf{GenAddr}(c, \beta, \mathbf{y_0^{1,k}}, \mathbf{y_1^{1,k}}, r, z, y_0, y_1)$$

GenAddr will generate addresses to represent the tape associated with its fifth argument. To emphasize this, we use the letter n (as node) at this position, while the fourth argument contains its parent configuration, denoted by p.

$$\mathsf{GenAddr}(c_p, c_n, a_1, \ldots, a_i, \ldots, a_{k+i}, \ldots, a_{2k}, r, z, y_0, y_1) \rightarrow \mathsf{GenAddr}(c_p, c_n, a_1, \ldots, a_{k+i}, \ldots, a_i, \ldots, a_{2k}, r, z, y_0, y_1)$$

For each address, we initialize its content (as well as the content of the previous and next cells) by fresh values $\mathbf{v}, \mathbf{v_{prev}}, \mathbf{v_{next}}$.

$$\mathsf{GenAddr}(c_p, c_n, a_1, \ldots, a_{2k}, z, y_0, y_1) \rightarrow \exists \mathbf{v}, \mathbf{v_{prev}}, \mathbf{v_{next}}$$
$$\mathsf{Cell}(c_p, c_n, a_1, \ldots, a_k, \mathbf{v}, \mathbf{v_{prev}}, \mathbf{v_{next}}, r, z, y_0, y_1)$$

Note that the values $\mathbf{v}$, $\mathbf{v_{prev}}$ and $\mathbf{v_{next}}$ are vectors of length the size of $(\Sigma \cup \{\flat\}) \times (Q \cup \bot)$. In particular, we use the notation $\mathbf{l_i}(x)$ to represent a vector of same length, composed of fresh variables, except for the position i, that contains x.

We now use mappings to force some of these values to be equal to c_{Crit}. Each position of $\mathbf{v}$ represents an element of $(\Sigma \cup \{\flat\}) \times (Q \cup \bot)$, and we will enforce exactly one of these positions to contain c_{Crit}. If the head of the Turing machine is on the cell represented, then the position of v corresponding to (a, q) where a is the letter in the cell and q the state of the Turing machine, will contain c_{Crit}. Otherwise, the position of v corresponding to $(a, \bot)$ will contain c_{Crit}.

As we store the content of a cell in several atoms, we must ensure that the tape associated with a configuration is consistent, by checking that $\mathbf{v_{next}}$ is consistent with $\mathbf{v}$ from the next cell. To ensure guardedness, we first introduce auxiliary predicates Cell_i^c, Cell_i^p and Cell_i^n that define the content of the i^{th} bit of the value of the current, previous and next cells:

$$\mathsf{Cell}(c_p, c_n, \mathbf{y^{1,k}}, \mathbf{l_i}(x), \mathbf{v'_{prev}}, \mathbf{v'_{nxt}}, r, z, y_0, y_1)$$
$$\rightarrow \mathsf{Cell}_i^c(c_n, \mathbf{y^{1,k}}, x)$$

We now introduce the definition of a mapping $\mathcal{T}_{\mathsf{data}_n}(x)$ which ensures the consistency of the tape content (note that the first atom is a guard):

$$\mathsf{Cell}(c_p, c_n, y_{b_1}, \ldots, y_{b_j}, y_0, \mathbf{y_1}, \mathbf{v}, \mathbf{v_{prev}}, \mathbf{l_i}(x), r, z, y_0, y_1)$$
$$\wedge\, \mathsf{Cell}_i^c(c_n, y_{b_1}, \ldots, y_{b_j}, y_1, \mathbf{y_0}, z)$$

$\mathcal{T}_{\mathsf{data}_p}(x)$ is defined similarly to deal with the previous cell.

We enforce the tape of the initial configuration to have the head of the Turing machine on the first cell (and assume w.l.o.g that this is represented by the first position of $\mathbf{v}$ containing c_{Crit}) and all the other cells containing $\flat$ (and we assume w.l.o.g that this is represented by the second position of $\mathbf{v}$ containing c_{Crit}). We thus create the mappings $\mathcal{T}_{tape_i}(x)$, for the the first cell, having definition:

$$\mathsf{Cell}(c_p, c_n, \mathbf{y_0}, l_1(x), \mathbf{v_{prev}}, \mathbf{v_{next}}, c_p, z, y_0, y_1)$$

and we introduce the mappings $\mathcal{T}_{tape_o}(x)$, for all the other cells, having definition:

$$\mathsf{Cell}(c_p, c_n, \ldots, y_1, \ldots, a_n, l_2(x), \mathbf{v_{prev}}, \mathbf{v_{next}}, c_p, z, y_0, y_1)$$

Note that this data is associated with the children of the root (as p is both in the fourth and last minus three positions of the atoms), and not with the root itself, due to the choice of keeping in Cell the identifier of the parent of the considered configuration.

We then check that the tape associated with the α-successor of a configuration is indeed obtained by applying an α-transition. This is done by noticing that the value of each cell of the α-successor is deterministically defined by the value of the cell and its two neighbors in the original configuration (the neighbors are necessary to know whether the head of the Turing machine is now in the considered cell). To ensure guardedness, we first define a predicate marking α-successors (and similarly for β-successors):

$$\mathsf{Children}_{\mathscr{Q}}(c, \alpha, \beta, z, ac_\alpha, ac_\beta, \mathbf{y_0^k}, \mathbf{y_1^k}, c_{root}, z, y_0, y_1)$$
$$\rightarrow \mathsf{succ}_\alpha(c, \alpha)$$

Let us consider a cell of address $\mathbf{b^{1,k}}$ in c_p. We assume that its content is represented by i, while the content of its left (resp. right) neighbor is represented by j (resp. k). We represent the fact that this implies that the content of the cell of address $\mathbf{b^{1,k}}$ is w in the α-successor of c_p by the following mapping $\mathcal{T}^{\alpha}_{i,j,k\rightarrow w}(x)$:

$$\mathsf{Cell}(c_p, c_n, \mathbf{b^{1,k}}, \mathbf{l_w}(x), \mathbf{v_{prev}}, \mathbf{v_{next}}, r, z, y_0, y_1)$$
$$\wedge\, \mathsf{Cell}_i^c(c_p, \mathbf{b^{1,k}}, z)$$
$$\wedge\, \mathsf{Cell}_j^p(c_p, \mathbf{b^{1,k}}, z)$$
$$\wedge\, \mathsf{Cell}_k^n(c_p, \mathbf{b^{1,k}}, z)$$
$$\wedge\, \mathsf{succ}_\alpha(c_p, c_n)$$

Note that the above formulation requires the content of the previous and of the next cells, which makes this mappings not applicable when $\mathbf{b^{1,k}}$ is the address of either the first or the last cell. We thus add rules to specifically deal with these two cases (that looks at the content of the current and next cell when $\mathbf{b^{1,k}}$ is a vector of y_0, and at the content of current and previous cell when $\mathbf{b^{1,k}}$ is a vector of y_1). Note that there is only polynomially such mappings to be built. And we finally create a mapping $\mathcal{T}_{accept}(x)$ enforcing that configurations whose tape is in an accepting state (which we assume w.l.o.g. corresponds to the case where the first cell contains the l^{th} bit) are declared as accepting.

$$\mathsf{Cell}_l^c(c_n, \mathbf{y_0^k}, z)$$
$$\wedge\, \mathsf{Children}_{\mathscr{Q}}(c_n, \alpha, \beta, x, ac_\alpha, ac_\beta, \mathbf{y_0^k}, \mathbf{y_1^k}, r, z, y_0, y_1)$$

The policy query is

$$\mathsf{Children}_{\exists}(root, \alpha, \beta, z, ac_\alpha, ac_\beta, \mathbf{y_0^{1,k}}, \mathbf{y_1^{1,k}}, root, z, y_0, y_1),$$

We will show that this policy query is disclosed if and only if the original Turing machine accepts on the empty tape.

B.3 Proof of Correctness

We show that the policy is disclosed if and only if $\mathscr{T}$ accepts on the empty tape. By Theorem 1, the policy is disclosed if and only if the corresponding HOCWQ problem has a positive answer. Further, this holds if and only if the policy query holds on the result of the visible chase (introduced after Theorem 9). We thus focus on showing the equivalence of the acceptance of the empty tape by $\mathscr{T}$ and the satisfaction of the policy in the visible chase.

Let us start by describing some relationships between the visible chase of $\mathcal{D}_{\mathsf{Crit}}^{\mathcal{G}(\mathcal{M})}$ and the run of $\mathscr{T}$. As $\mathcal{D}_{\mathsf{Crit}}^{\mathcal{G}(\mathcal{M})}$ contains $\mathcal{T}_{\mathsf{Init}}(c_{\mathsf{Crit}})$, there is in the visible chase the atom

$$\mathsf{Children}_{\exists}(c_{root}, c_\alpha, c_\beta, ac_{root}, ac_\alpha, ac_\beta, \mathbf{y_0^k}, \mathbf{y_1^k}, c_{root}, c_{\mathsf{Crit}}, y_0^0, y_1^0),$$

where all individuals but c_{Crit} are nulls.

Definition 1 (Tape Representation)**.** *Let T be a tape (with head position and state included) of $\mathscr{T}$. A representation of T is a set of atoms*

$$\{\mathsf{Cell}(c_p, c_n, \mathbf{a}, \mathbf{v^a}, \mathbf{v_{prev}}^{\mathbf{a}}, \mathbf{v_{next}}^{\mathbf{a}}, c_{\mathsf{Crit}}, y_0, y_1)\}_{\mathbf{a}},$$

where $\mathbf{a}$ ranges over the binary representations of the addresses of T, and such that for any cell of T the following holds:

- *for any a, $\mathbf{v}$ contains fresh nulls except for the bit that represents the content of T at address $\mathbf{a}$, where it contains c_{Crit}*
- *for any a except the representation of the leftmost cell, $\mathbf{v_{prev}}$ contains fresh nulls except for the bit that represents the content of T at address $\mathbf{a}-1$; in this bit it contains c_{Crit} ($\mathbf{v_{prev}}$ exclusively contains fresh nulls for the leftmost cell)*
- *for any a except the representation of the rightmost cell, $\mathbf{v_{next}}$ contains fresh nulls except for the bit that represents the content of T at address $\mathbf{a}+1$; on this bit it contains c_{Crit} ($\mathbf{v_{next}}$ exclusively contains fresh nulls for the rightmost cell)*

In that case, c_n is called a representative of T.

Lemma 1. *c_α and c_β, as defined above Definition 1, are representatives of the initial tape.*

Proof. We show the result for c_α, the same reasoning being applicable to c_β. As the atom

$$\mathsf{Children}_\exists(c_{root}, c_\alpha, c_\beta, ac_{root}, ac_\alpha, ac_\beta, \mathbf{y_0^k}, \mathbf{y_1^k}, c_{root}, c_{\mathsf{Crit}}, y_0^0, y_1^0),$$

belongs to the visible chase, atoms of the shape

$$\mathsf{Cell}(c_{root}, c_\alpha, a_1, \ldots, a_k, \mathbf{v}, \mathbf{v_{prev}}, \mathbf{v_{next}}, c_{root}, z, y_0, y_1)$$

for any vector $a_1, \ldots, a_k$ with $a_i \in \{y_0, y_1\}$ for any i, are generated, where all nulls from $\mathbf{v}$, $\mathbf{v_{prev}}$ and $\mathbf{v_{next}}$ are fresh (thanks to the rules involving GenAddr). As the first argument and the ante-ante-penultimate argument of such an atom are equal, the definition of $\mathcal{T}_{\mathsf{tape}_i}(x)$ maps to the atom of address $y_0, \ldots, y_0$, and the body of $\mathcal{T}_{\mathsf{tape}_\mathsf{o}}(x)$ maps to all the other atoms. Applying $\mathcal{T}_{\mathsf{data}_n}$ and $\mathcal{T}_{\mathsf{data}_p}$ then ensures that c_α is a representative for the initial tape, as no other mapping may merge a term of these atoms. □

Lemma 2. *If c_p is a representative of a tape T and if the visible chase contains*

$$\mathsf{Children}_\mathscr{Q}(c_p, \alpha, \beta, ac, ac_\alpha, ac_\beta, \mathbf{y_0^k}, \mathbf{y_1^k}, c_{root}, c_{\mathsf{Crit}}, y_0, y_1),$$

then α (resp. β) is a representative of the tape T_α (resp. T_β) obtained by applying the α-transition (resp. β-transition) applicable to T.

Proof. We show the result for the α-successor, the same reasoning being applicable for the β-successor. As the visible chase contains

$$\mathsf{Children}_\mathscr{Q}(c_p, \alpha, \beta, ac, ac_\alpha, ac_\beta, \mathbf{y_0^k}, \mathbf{y_1^k}, c_{root}, c_{\mathsf{Crit}}, y_0, y_1),$$

it also contains atoms of the shape:

$$\mathsf{Cell}(c_p, \alpha, a_1, \ldots, a_k, \mathbf{v}, \mathbf{v_{prev}}, \mathbf{v_{next}}, c_{root}, c_{\mathsf{Crit}}, y_0, y_1),$$

for any vector $a_1, \ldots, a_k$, where all nulls from $\mathbf{v}$, $\mathbf{v_{prev}}$ and $\mathbf{v_{next}}$ are fresh. Note that c_p is necessary distinct from c_{root} (as it is the representative of a tape). Hence neither $\mathcal{T}_{\mathsf{tape}_i}(x)$ nor $\mathcal{T}_{\mathsf{tape}_\mathsf{o}}(x)$ may unify a term with c_{Crit}. As c_p is a representative of T, for any address, if the i^{th} bit of $\mathbf{v}$ represents the actual value in T at address ad, then the visible chase contains $\mathsf{Cell}_i^c(c_p, \mathbf{b^{1,k}}, c_{\mathsf{Crit}})$ where $\mathbf{b^{1,k}}$ is the binary encoding of ad. Similarly, $\mathsf{Cell}_i^n(c_p, \mathbf{b^{1,k}}, c_{\mathsf{Crit}})$ and $\mathsf{Cell}_i^p(c_p, \mathbf{b^{1,k}}, c_{\mathsf{Crit}})$ also belong to the visible chase where applicable. Then for all addresses, an application of the relevant mapping of the shape $\mathcal{T}^\alpha_{i,j,k\to w}(x)$ merges the null at the position representing the content of T_α with c_{Crit}. Applying $\mathcal{T}_{\mathsf{data}_n}$ and $\mathcal{T}_{\mathsf{data}_p}$ then ensures that α is a representative for $\mathcal{T}_\alpha$. □

Wrapping up the previous two lemmas, we get that there is a tree structure in the visible chase that corresponds exactly to the tree of configurations of the run of $\mathscr{T}$: the two individuals c_α and c_β are representatives of the initial configuration, and their children (which are the individuals at the second and third individuals in the Children atom in which they appear at the first position) are representatives of the configurations that can be reached with an α or β transition. It remains to check that the argument representing the accepting status of a configuration are correctly set, which is the topic of the following lemma.

Lemma 3. *If c_p is the representative of a tape, there is in the visible chase an atom of the shape*

$$\mathsf{Children}_\mathscr{Q}(c_p, \alpha, \beta, c_{\mathsf{Crit}}, ac_\alpha, ac_\beta, \mathbf{y_0^k}, \mathbf{y_1^k}, c_{root}, c_{\mathsf{Crit}}, y_0, y_1),$$

if and only if $\mathscr{T}$ accepts on T.

Proof. Let T be a tape of representative c_p. There are four cases in which $\mathscr{T}$ accepts on T:

- the state of $\mathscr{T}$ is final in T: this is the case if and only if $\mathcal{T}_{\mathsf{accept}}$ merges the fourth argument of $\mathsf{Children}_\mathscr{Q}(c_p, \alpha, \beta, ac, ac_\alpha, ac_\beta, \mathbf{y_0^k}, \mathbf{y_1^k}, c_{root}, c_{\mathsf{Crit}}, y_0, y_1)$ with c_{Crit}
- the state of $\mathscr{T}$ is universal in T and both its successors are accepting: by induction assumption (on the number of transitions that need to be applied to prove acceptance of a tape), both accepting bits of α and β are unified with c_{Crit}, and thus the accepting bit of c_p is unified with c_{Crit} thanks to $\mathcal{T}_\forall$
- the state of $\mathscr{T}$ is existential in T and its α-successor is accepting: by induction assumption, the accepting bit of α is unified with c_{Crit}, and thus the accepting bit of c_p is unified with c_{Crit} thanks to $\mathcal{T}_{\exists,\alpha}$
- similar case, with the β-successor.

□

Let us now use the above lemmas to show that $\mathscr{T}$ accepts if and only if the policy query p holds in the visible chase. If $\mathscr{T}$ accepts, let us consider an accepting run of $\mathscr{T}$. From

Lemmas 1 and 2, we can build a configuration tree that contains a representative for all the tapes that are involved in this run. From Lemma 3, the accepting bits of the representative are set adequately, and the policy query holds in the visible chase.

Conversely, let us consider a visible chase sequence such that the policy query holds in its result. Let us first remark that the argument of the policy query appearing in the first position is equal to the argument in the ante-ante-penultimate position. This implies that none of the witnesses of mappings other than $\mathcal{T}_{\mathsf{Init}}$ need to be applied in order to entail the policy query, which can be seen from the following three facts: *(i)* none of the positions that may contain a configuration identifier may be unified with c_{Crit}; *(ii)* all mappings contain configuration identifiers *(iii)* only the witness associated with $\mathcal{T}_{\mathsf{Init}}$ may generate an atom of the shape

$$\mathsf{Children}_{\exists}(root, \alpha, \beta, z, ac_\alpha, ac_\beta, \mathbf{y_0^{1,k}}, \mathbf{y_1^{1,k}}, root, z, y_0, y_1),$$

This implies that in the visible chase sequence entailing the policy query, we start by introducing α and β as in Lemma 1. Let us now consider the smallest set S of configuration representatives that fulfills the following conditions:

- α and β are in S
- if c is in S and the tape associated with c is in a universal state, then both successors of c are in S
- if c is in S and the tape associated with c is an existential state, then a successor of c having its acceptance bit equal to c_{Crit} is in S.

By the previous lemmas, there exists an accepting run of $\mathscr{T}$ going exactly through the represented configurations.

B.4 Second part of proof of Theorem 6: EXPTIME-hardness for inclusion dependencies and guarded maps in bounded arity

Recall the statement of the second part of Theorem 6:

$\mathsf{Disclose}_{\mathsf{C}}(\mathsf{IncDep}, \mathsf{GuardedMap})$ is EXPTIME-hard even in bounded arity.

In the proof of Theorem 6, we used predicates of unbounded arity only to generate exponentially many cell addresses. Here, we use only k addresses, and can encode their content through k predicates Cell_1 to Cell_k. However, the proof follows the same line of argumentation as in Theorem 6.

Let us describe the source signature.

- $\mathsf{Children}_{\forall}(c, c_\alpha, c_\beta, ac, ac_\alpha, ac_\beta, r, z)$ states that a configuration c is universal and has as children c_α and c_β, and that the acceptance bit of c is ac, of c_α is ac_α and of c_β is ac_β. The last two positions are placeholders for the root of the tree of configurations, and c_{Crit}.
- $\mathsf{Children}_{\exists}(c, c_\alpha, c_\beta, ac, ac_\alpha, ac_\beta, r, z)$: same meaning, except that c is existential.
- $\mathsf{Cell}^l(c_p, c_n, \mathbf{v}, z)$ states that the cell of address l of the tape represented by c_n has a content represented by $\mathbf{v}$. The last position is a placeholder for c_{Crit}.
- $\mathsf{Cell}_i^l(c, x, z)$ states that the cell of address l in configuration c contains x at the i^{th} position of the representation of its content.
- $\mathsf{succ}_\alpha(c_p, c_n)$ states that c_n is the α-successor of c_p (and similarly for β)

The symbol $\mathscr{Q}$ always ranges over $\{\forall, \exists\}$.

Initialization We first define a mapping $\mathcal{T}_{\mathsf{Init}}(x)$, introducing some elements in the visible chase, whose definition is:

$$\mathsf{Children}_{\exists}(c_{root}, c_\alpha, c_\beta, ac_{root}, ac_\alpha, ac_\beta, c_{root}, x)$$

Generation of the tree of configuration α-successors have themselves α- and β-successors, and are existential if their parent is universal:

$$\mathsf{Children}_{\forall}(c, \alpha, \beta, ac, ac_\alpha, ac_\beta, r, z)$$
$$\rightarrow \exists \alpha_\alpha, \alpha_\beta, ac_{\alpha_\alpha}, ac_{\alpha_\beta}$$
$$\mathsf{Children}_{\exists}(\alpha, \alpha_\alpha, \alpha_\beta, ac_\alpha, ac_{\alpha_\alpha}, ac_{\alpha_\beta}, r, z)$$

And similarly for $\mathsf{Children}_{\exists}$ and for the β-successor.

Universal and Existential Acceptance Condition If both successors of a universal configuration n are accepting, so is n. We create a mapping $\mathcal{T}_{\forall}(x)$ having definition:

$$\mathsf{Children}_{\forall}(c, \alpha, \beta, x, z, z, r, z)$$

If the α-successor of an existential configuration n is accepting, so is n. We create a mapping $\mathcal{T}_{\exists,\alpha}(x)$ having definition:

$$\mathsf{Children}_{\exists}(c, \alpha, \beta, x, z, ac_\beta, r, z)$$

We create a similar mapping $\mathcal{T}_{\exists,\beta}$ for the β-successor.

Tape Representation and Consistency of Tapes We now focus on the representation of the tape and its consistency. For each configuration, we generate k cells whose content is initialized freshly:

$$\mathsf{Children}_{\forall}(c, \alpha, \beta, ac, ac_\alpha, ac_\beta, r, z)$$
$$\rightarrow \exists \mathbf{v}\ \mathsf{Cell}^l(c, \alpha, \mathbf{v}, z)$$

and similarly for existential configurations and for the β-successors. Note that the values $\mathbf{v}$, $\mathbf{v_{prev}}$ and $\mathbf{v_{next}}$ are again vectors of length the size of $(\Sigma \cup \{\flat\}) \times (Q \cup \bot)$. We again use the notation $\mathbf{l_i}(x)$ to represent a vector of same length, composed of fresh variables, except for the position i, which contains x.

To ensure guardedness, we first introduce auxiliary predicates Cell_i^c, Cell_i^p and Cell_i^n that define the content of the i^{th} bit of the value of the current, previous and next cells:

$$\mathsf{Cell}^l(c_p, c_n, \mathbf{l_i}(x), z)$$
$$\rightarrow\ \mathsf{Cell}_i^l(c_n, x)$$

We enforce that the tape of the initial configuration has the head of the Turing machine on the first cell (and assume w.l.o.g that this is represented by the first position of $\mathbf{v}$ containing c_{Crit}) with all the other cells containing $\flat$. We also

assume w.l.o.g that the other cells containing $\flat$ is represented by the second position of $\mathbf{v}$ containing c_{Crit}. We thus create the mappings $\mathcal{T}_{tape_i}(x)$, for the the first cell, having definition:

$$\mathsf{Cell}^1(c_p, c_n, l_1(x), c_p)$$

and $\mathcal{T}^l_{tape_o}(x)$, for all the other cells ($2 \leq l \leq n$), with definition:

$$\mathsf{Cell}^l(c_p, c_n, l_2(x), c_p)$$

Note that this data is associated with the children of the root (as c_p is both in the first and the penultimate positions of the atoms), and not with the root itself, due to the choice of keeping in Cell^l the identifier of the parent of the considered configuration.

We then check that the tape associated with the α-successor of a configuration is indeed obtained by applying an α-transition. This is done by noticing that the value of each cell of the α-successor is determined by the value of the cell and its two neighbors in the original configuration (the neighbors are necessary to know whether the head of the Turing machine is now in the considered cell). To ensure guardedness, we first define a predicate marking α-successors (and similarly for β-successors):

$$\begin{aligned}&\mathsf{Children}_{\mathscr{Q}}(c, \alpha, \beta, z, ac_\alpha, ac_\beta, r, z)\\ &\rightarrow \mathsf{succ}_\alpha(c, \alpha)\end{aligned}$$

Let us consider a cell of address l in c_p. We assume that its content is represented by i, while the content of its left (resp. right) neighbor is represented by j (resp. k). We represent the fact that this implies that the content of the cell of address l is w in the α-successor of c_p by the following mapping $\mathcal{T}^\alpha_{i,j,k\rightarrow w}(x)$:

$$\begin{aligned}&\mathsf{Cell}^l(c_p, c_n, \mathbf{l_w}(x), z)\\ &\wedge\ \mathsf{Cell}^l_i(c_p, z)\\ &\wedge\ \mathsf{Cell}^{l-1}_j(c_p, z)\\ &\wedge \mathsf{succ}_\alpha(c_p, c_n)\end{aligned}$$

As in the non-bounded case, the first (resp. last) cell should be dealt with separately, as there is no content in the (non-existent) previous (resp. next) cell. And we finally create a mapping $\mathcal{T}_{accept}(x)$ enforcing that configurations whose tape is in an accepting state (which we assume w.l.o.g. corresponds to the case where the first cell contains the l^{th}_f bit) are declared as accepting.

$$\begin{aligned}&\mathsf{Cell}^1_{l_f}(c_n, z)\\ &\wedge \mathsf{Children}_{\mathscr{Q}}(c_n, \alpha, \beta, x, ac_\alpha, ac_\beta, r, z)\end{aligned}$$

The policy is

$$\mathsf{Children}_{\exists}(root, \alpha, \beta, z, ac_\alpha, ac_\beta, root, z)$$

We can verify that this policy is disclosed if and only if the original Turing machine accepts on the empty tape, using a similar reasoning to the unbounded case.

B.5 Final part of proof of Theorem 6: reduction from GTGD and ProjMap to IncDep and GuardedMap

Theorem 6 states a 2EXPTIME lower bound for general arity and an EXPTIME lower bound in bounded arity for two different cases. The first case was when the source constraints are IncDeps and the mappings are guarded. The previous sections of the appendix have gone through the proofs of this case in detail. We now finish the proof of Theorem 6 showing:

$\mathsf{Disclose}_{\mathsf{C}}(\mathsf{GTGD}, \mathsf{ProjMap})$ is 2EXPTIME-hard, and is EXPTIME-hard even in bounded arity.

The proof of both of these assertions follows directly from Corollary 4 (the general reduction of maps presented in section A.2).

We have seen that IncDep and GuardedMap reduces to GTGD and ProjMap, therefore we have the lower bound for $\mathsf{Disclose}(\mathsf{GTGD}, \mathsf{ProjMap}, p)$ from the lower bound $\mathsf{Disclose}(\mathsf{IncDep}, \mathsf{GuardedMap}, p)$

B.6 Proof of Theorem 7: EXPTIME-hardness for inclusion dependencies and atomic maps, and for LTGDs with projection maps

Recall the statement of Theorem 7:

$\mathsf{Disclose}_{\mathsf{C}}(\mathsf{IncDep}, \mathsf{AtomMap})$ and $\mathsf{Disclose}_{\mathsf{C}}(\mathsf{LTGD}, \mathsf{ProjMap})$ are both EXPTIME-hard.

We first focus on the case of $\mathsf{Disclose}_{\mathsf{C}}(\mathsf{IncDep}, \mathsf{AtomMap})$. We adapt the construction used for PSPACE-hardness of entailment with IncDeps [Casanova *et al.*, 1984] to show EXPTIME-hardness for IncDep source constraints and atomic maps. We start with an alternating (rather than deterministic in [Casanova *et al.*, 1984]) Turing machine $\mathcal{M}$ and an input x, and consider the problem asking whether there exists a halting computation of $\mathcal{M}$ that uses at most $|x|$ cells. As in the original reduction, we use inclusion dependencies to simulate the transition relation of $\mathcal{M}$. The adaptation lies in the additional use of a fresh position holding a configuration identifier, and the generation of a tree of configurations, as in the reduction presented in Theorem 6.

Let us describe the signature:

- $\mathsf{Config}^{\mathscr{Q}}(c, ac, \mathbf{v}, z)$ states the configuration c has quantification $\mathscr{Q}$, has accepting bit ac, a tape represented by $\mathbf{c}$. the last argument will always hold c_{Crit} in the visible chase;
- $\mathsf{Transition}^{\mathscr{Q}}_{t_\alpha, t_\beta}(c, ac, \mathbf{v}, \alpha, ac_\alpha, \beta, ac_\beta, z)$ names two successors configurations α and β, with the configurations consisting of acceptance bits ac_α and ac_β, which are obtained from c by applying transitions t_α and t_β.

Let us turn to the description of $\mathbf{v}$ and subsequently t_α. $\mathbf{v}$ represents the content of the tape: for each position of the tape, there is an argument for each pair of $\Sigma \times (Q \cup \{\bot\})$. Intuitively, this argument is equal to c_{Crit} if and only if the position contains the corresponding letter and head, and a fresh null otherwise.

We introduce a mapping that initializes the tape:

$$\mathsf{Config}^{\forall}(x, ac, \mathbf{v}, x)$$

As in the proof of Theorem 6, we propagate the acceptance information using mappings. For a universal state, we use a mapping with definition:

$$\mathsf{Transition}^{\forall}_{t_\alpha, t_\beta}(c, x, \mathbf{v}, \alpha, z, \beta, z, z)$$

For an existential state, we use two mappings with definitions:

$$\mathsf{Transition}^{\exists}_{t_\alpha, t_\beta}(c, x, \mathbf{v}, \alpha, z, \beta, ac_\alpha, z)$$

and

$$\mathsf{Transition}^{\exists}_{t_\alpha, t_\beta}(c, x, \mathbf{v}, \alpha, ac_\alpha, \beta, z, z)$$

As before, we notice that the state of a cell after applying a transition is deterministically defined by its content as well as the content of its left and right neighbor. The following inclusion dependency states that from any configuration, we can try to apply all possible transitions to generate the α- and β-successors:

$$\mathsf{Config}^{\mathscr{Q}}(c, ac, \mathbf{v}, z) \rightarrow \exists \alpha, ac_\alpha, \beta, ac_\beta$$
$$\mathsf{Transition}^{\mathscr{Q}}_{t_\alpha, t_\beta}(c, ac, \mathbf{v}, \alpha, ac_\alpha, \beta, ac_\beta, z)$$

We now generate the tape associated with the α-transition (and similarly for the β-transition):

$$\mathsf{Transition}^{\mathscr{Q}}_{t_\alpha, t_\beta}(c, ac, \mathbf{v}, \alpha, ac_\alpha, \beta, ac_\beta, z) \rightarrow$$
$$\exists \mathbf{v}'_\alpha\ \mathsf{Config}^{\vec{\mathscr{Q}}}(\alpha, ac_\alpha, \mathbf{v}_\alpha \oplus \mathbf{v}'_\alpha, z),$$

where $\vec{\mathscr{Q}}$ denotes the dual quantifier. Let us describe the vector $\mathbf{v}_\alpha \oplus \mathbf{v}'_\alpha$. Suppose t_α is the transition that checks whether position i contains a, position $i+1$ contains b and the head in state s, and position $i+2$ contains c; changes b to b', moves the head to the right and goes into state s'. Then $\mathbf{v}_\alpha \oplus \mathbf{v}'_\alpha$ is defined as follows:

- any argument that corresponds to a position distinct from $i+1$ or $i+2$ is chosen equal to the argument at the same position in $\mathbf{v}$;
- the argument that corresponds to $(i+1, (b', \bot))$ now contains the value of $\mathbf{v}$ at position $((i+1), (b, s))$, and all other variables appearing in an argument corresponding to position $(i+1)$ are existentially quantified;
- the argument that corresponds to $((i+2), (c, s'))$ now contains the value of $\mathbf{v}$ at position $((i+2), (b, \bot))$, and all other variables appearing in an argument corresponding to position $(i+1)$ are existentially quantified.

Note that here we have a distinction with the previous reduction: we do not check that a transition is applicable before applying it, as this would be out of the capabilities of IncDep. However, the same argument as in [Casanova *et al.*, 1984] proves that a configuration reached from simulating a non-applicable transition cannot lead to an accepting state. We choose as a policy:

$$\mathsf{Config}^{\forall}(x, x, \mathbf{v}, x),$$

Proposition 10. *The policy is disclosed if and only if there is an accepting computation that uses at most $|x|$ cells.*

The lower bound for $\mathsf{Disclose}_{\mathsf{C}}(\mathsf{LTGD}, \mathsf{ProjMap})$ follows by reduction:

Proposition 11. *There is a polynomial time reduction from* $\mathsf{Disclose}_{\mathsf{C}}(\mathsf{IncDep}, \mathsf{AtomMap})$ *to* $\mathsf{Disclose}_{\mathsf{C}}(\mathsf{LTGD}, \mathsf{ProjMap})$.

Proof. Given a mapping $\phi(\vec{x}) \rightarrow \exists \vec{y}\ H(\vec{t})$ where there may be repeated variables in the head atom, we replace it by a projection mapping

$$\phi(\vec{x}) \rightarrow \exists \vec{y}\ H'(\vec{t})$$

where H' is a new predicate whose arity is the number of distinct variables in $H(\vec{t})$. $H'(\vec{t'})$ has the same variables as H, but with no repetition. For example, if the head of the original rule is $H(x, x, y)$, then the new rule has head $H'(x, y)$.

We additionally add the source constraint:

$$\forall \vec{t}\ H'(\vec{t}) \rightarrow H(\vec{t})$$

It is easy to see that this transformation preserves disclosure. □

B.7 Proof of Theorem 8: lower bounds for IncDeps in bounded arity

Recall the statement of Theorem 8:

$\mathsf{Disclose}_{\mathsf{C}}(\mathsf{IncDep}, \mathsf{Map})$ is 2EXPTIME-hard in bounded arity.

This proof will be very similar to the proof of Theorem 6. We will provide a reduction from an alternating EXPSPACE Turing machine to IncDep and SCEQrules. We show how to simulate the run of an alternating EXPSPACE Turing machine $\mathcal{M}$ with inclusion dependencies and SCEQrules.

The main difference between the proof of Theorem 6 and the proof here is that in Theorem 6 each cell carried n bits $b_1 \dots b_n$ specifying the address of the cell. In this version we cannot use this trick as we are using a reduction where all predicates are bounded. For each configuration, the tape will represented in the leaves of a full binary tree of depth n. For a cell c, the n bits specifying the address of a c will scattered across the n predicates in its lineage, each holding one bit of the address: an internal node has two descendants each carrying four values $b, \vec{b}, y_0, y_1$. We will have $b = y_0$ and $\vec{b} = y_1$ when then node represents the addresses where the i-th bit is 0 and $b = y_1$, $\vec{b} = y_0$ when it is 1.

Let us describe our source signature:

- $\mathsf{Children}_{\forall}(c, c_\alpha, c_\beta, ac, ac_\alpha, ac_\beta, r, y_0, y_0^{bis}, y_1, y_1^{bis})$ states that a configuration c is universal and has children c_α and c_β, and that the acceptance bit of c is ac, of c_α is ac_α and of c_β is ac_β. The last four positions are placeholders for r the root of the tree of configurations, two values $y_0 = y_0^{bis}$ representing 0 and two values $y_1 = y_1^{bis}$ representing 1.
- $\mathsf{Children}_{\exists}(c, c_\alpha, c_\beta, ac, ac_\alpha, ac_\beta, r, y_0, y_0^{bis}, y_1, y_1^{bis})$: same meaning, except that c is existential.

- for $i \in 1..n$, $\mathsf{Address}_i(c_p, c_n, b_i, \vec{b_i}, y_0, y_1)$ corresponds to a node of depth i in the binary tree representing the tape of a configuration. In this predicate c_p is the parent of the node, c_n is the current node, b_i will be equal to y_0 when the node if the first child of c_p and equal to y_1 otherwise. $\vec{b_i}$ will be the complement of b_i (i.e. $y_0 = b_i$ implies $y_1 = \vec{i}$ and $y_1 = b_i$ implies $y_0 = \vec{i}$).
- $\mathsf{Cell}^c(c, \vec{v})$ states that the cell at position c contains the data represented by $\vec{v}$. Cell^p and Cell^n play similar roles for the previous cell and the next cell.

Critical element We create a mapping $T_{c_{\mathsf{Crit}}}(x)$ defined as $\mathsf{IsCrit}(x)$. The relation IsCrit will allow us to test whether a variable is equal to c_{Crit}.

Initialization We first define a mapping $\mathcal{T}_{init}()$ introducing some elements in the visible chase, whose definition is:

$$\mathsf{Children}_\exists(c_{root}, c_\alpha, c_\beta, ac_{root}, ac_\alpha, ac_\beta, c_{root}, y_0, y_0, y_1, y_1)$$

Generating the tree of configuration α-successors have themselves α- and β-successors, and are existential if their parent is universal:

$$\mathsf{Children}_\forall(c, \alpha, \beta, ac, ac_\alpha, ac_\beta, r, y_0, y_0^b, y_1, y_1^b) \rightarrow \exists \alpha_\alpha, \alpha_\beta, ac_{\alpha_\alpha}, ac_{\alpha_\beta}\ \mathsf{Children}_\exists(\alpha, \alpha_\alpha, \alpha_\beta, ac_\alpha, ac_{\alpha_\alpha}, ac_{\alpha_\beta}, r, y_0, y_0^b, y_1, y_1^b)$$

And similarly for $\mathsf{Children}_\exists$ and for the β-successor.

Universal and Existential Acceptance Condition If both successors of a universal configuration n are accepting, so is n. We create a mapping $\mathcal{T}_\forall(x)$ with definition:

$$\mathsf{Children}_\forall(c, \alpha, \beta, x, ac, ac, r, y_0, y_0^b, y_1, y_1^b) \wedge \mathsf{IsCrit}(ac)$$

If the α-successor of an existential configuration n is accepting, so is n. We create a mapping $\mathcal{T}_{\exists,\alpha}(x)$ of definition:

$$\mathsf{Children}_\exists(c, \alpha, \beta, x, ac_\alpha, ac_\beta, r, y_0, y_0^b, y_1, y_1^b) \wedge \mathsf{IsCrit}(ac_\alpha)$$

We create a similar mapping $\mathcal{T}_{\exists,\beta}$ for the β-successor.

Generating the tape cells We now focus on the representation of the tape and its consistency. We generate 2^k addresses and associated values:

$$\mathsf{Children}_{\mathscr{Q}}(c, \alpha, \beta, ac, ac_\alpha, ac_\beta, r, y_0, y_0^b, y_1, y_1^b) \rightarrow \mathsf{Address}_1(c, c_1, y_0, y_1, y_0^b, y_1^b)$$

$$\mathsf{Children}_{\mathscr{Q}}(c, \alpha, \beta, ac, ac_\alpha, ac_\beta, r, y_0, y_0^b, y_1, y_1^b) \rightarrow \mathsf{Address}_1(c, c_1, y_1, y_0, y_0^b, y_1^b)$$

And for $i \in 1..n-1$ we have:

$$\mathsf{Address}_i(c_{i-1}, c_i, b, \vec{b}, y_0^b, y_1^b) \rightarrow \mathsf{Address}_{i+1}(c_i, c_{i+1}, , b, \vec{b}, y_0^b, y_1^b)$$

$$\mathsf{Address}_i(c_{i-1}, c_i, b, \vec{b}, y_0^b, y_1^b) \rightarrow \mathsf{Address}_{i+1}(c_i, c_{i+1}, \vec{b}, b, y_0^b, y_1^b)$$

Finally for n we have:

$$\mathsf{Address}_n(c_{n-1}, c_n, b, \vec{b}, y_0^b, y_1^b) \rightarrow \mathsf{Cell}^c(c_n, \vec{v})$$

Initialization of the tape For the case 0, we use the pattern l_1 and introduce a mapping $\mathcal{T}_{tape0}(x)$ defined as:

$$\begin{aligned} & \mathsf{Children}_\exists(c_{root}, \alpha, \beta, ac, ac_\alpha, ac_\beta, y_0, y_0, y_1, y_1) \\ \wedge\ & \mathsf{Address}_1(root, id_1, y_0, y_1, y_0, y_1) \\ \wedge\ & \dots \\ \wedge\ & \mathsf{Address}_n(id_{n-1}, id_n, y_0, y_1, y_0, y_1) \\ \wedge\ & \mathsf{Cell}^c(id_n, l_1(x)) \end{aligned}$$

For all others cases, with a first 1 at the i-th bit, we use the pattern l_0 and introduce $\mathcal{T}_{tape\ i}(x)$:

$$\begin{aligned} & \mathsf{Children}_\exists(c_{root}, \alpha, \beta, ac, ac_\alpha, ac_\beta, y_0, y_0, y_1, y_1) \\ \wedge\ & \mathsf{Address}_1(root, id_1, y_0, y_1, y_0, y_1) \\ \wedge\ & \dots \\ \wedge\ & \mathsf{Address}_i(id_{i-1}, id_i, y_1, y_0, y_0, y_1) \\ \wedge\ & \mathsf{Address}_{i+1}(id_i, id_{i+1}, a^i, b^i, y_0, y_1) \\ \wedge\ & \dots \\ \wedge\ & \mathsf{Address}_n(id_{n-1}, id_n, a^n, b^n, y_0, y_1) \\ \wedge\ & \mathsf{Cell}^c(id_n, l_0(x)) \end{aligned}$$

Ensuring the coherence between Cell^c and Cell^p We need to check the coherence between Cell^c in an address and Cell^p at the previous address. As usual when v is at the address $\vec{b}10^j$ then Cell^c needs to be checked against the Cell^p at the address $\vec{b}01^j$. We introduce the mapping $\mathcal{T}_{prev\ j}(x)$:

$$\begin{aligned} & \mathsf{Address}_{n-j-1}(id, id_1, y_1, y_0, y_0, y_1) \\ \wedge\ & \mathsf{Address}_{n-j-1}(id, id_0, y_0, y_1, y_0, y_1) \\ \wedge\ & \mathsf{Address}_{n-j}(id_1, id_{10}, y_0, y_1, y_0, y_1) \\ \wedge\ & \mathsf{Address}_{n-j}(id_0, id_{01}, y_1, y_0, y_0, y_1) \\ & \dots \\ \wedge\ & \mathsf{Address}_n(id_{10^{j-1}}, id_{10^j}, y_0, y_1, y_0, y_1) \\ \wedge\ & \mathsf{Address}_n(id_{01^{j-1}}, id_{01^j}, y_1, y_0, y_0, y_1) \\ \wedge\ & \mathsf{Cell}^c(id_{10^j}, l_k(\mathbf{s})) \\ \wedge\ & \mathsf{Cell}^p(id_{10^j}, l_k(v)) \\ \wedge\ & \mathsf{IsCrit}(v) \end{aligned}$$

Encoding transitions As in previous reductions, we encode the transitions of δ_ϵ as a set of $(i, j, k) \rightarrow w$ (where i is the value of the cell, j is the value at the cell before and k at the cell after and w is the written value).

For our transition, we need to write w at the same address s where l lies in the ϵ-child of the configuration of c. To be at the same address, we need to check that the path follows the same bits (that we note here $b_n \dots b_1$). We use the mapping $\mathcal{T}^\epsilon_{i,j,k\rightarrow w}(x)$ defined as:

$$
\begin{aligned}
& \mathsf{IsCrit}(v) \\
\wedge\ & \mathsf{Cell}^p(id_n, l_i(v)) \\
\wedge\ & \mathsf{Cell}^c(id_n, l_j(c)) \\
\wedge\ & \mathsf{Cell}^n(id_n, l_k(c)) \\
\wedge\ & \mathsf{Cell}^c(id'_n, l_w(x)) \\
\wedge\ & \mathsf{Address}_n(id_n, id_{n-1}, b_n, \vec{b}_n, y_0, y_1, y_0, y_1) \\
\wedge\ & \mathsf{Address}_n(id'_n, id'_{n-1}, b_n, \vec{b}_n, y_0, y_1) \\
\wedge\ & \mathsf{Address}_{n-1}(id_{n-1}, id_{n-2}, b_{n-1}, \vec{b}_{n-1}, y_0, y_1) \\
\wedge\ & \mathsf{Address}_{n-1}(id'_{n-1}, id'_{n-2}, b_{n-1}, \vec{b}_{n-1}, y_0, y_1) \\
& \dots \\
\wedge\ & \mathsf{Address}_1(id_p, id_1, b_1, \vec{b}_1, y_0, y_1) \\
\wedge\ & \mathsf{Address}_1(id'_c, id'_1, \vec{b}_1, b_1, y_0, y_1) \\
\wedge\ & Tree_\ell(id_g, id_p, y_0, y_0, y_1, y_1) \\
\wedge\ & \mathsf{Children}_{\mathscr{Q}}(c, c_\alpha, c_\beta, ac, ac_\alpha, ac_\beta, r, y_0, y_0, y_1, y_1)
\end{aligned}
$$

Encoding final states Whenever the current state is q_{accept} we need to enforce that the ac bit is set. To enforce that the ac bit is set, we introduce the following mapping, for each value $k \in \{q_{accept}\} \times \Sigma$ marking a final state:

$$
\begin{aligned}
& \mathsf{IsCrit}(v) \\
\wedge\ & \mathsf{Cell}^c(id_n, l_k(v)) \\
\wedge\ & \mathsf{Address}_n(id_{n-1}, id_n, x^{n-1}, z^{n-1}, y_0, y_1) \\
& \dots \\
\wedge\ & \mathsf{Address}_1(c, id_1, x^1, z^1, y_0, y_1) \\
\wedge\ & \mathsf{Children}_{\mathscr{Q}}(c, \alpha, \beta, x, ac_\alpha, ac_\beta, r, y_0, y_0, y_1, y_1)
\end{aligned}
$$

Policy The policy query is

$$\mathsf{Children}_{\exists}(root, \alpha, \beta, z, ac_\alpha, ac_\beta, r, y_0, y_0, y_1, y_1),$$

We can verify that the policy query is disclosed if and only if the original Turing machine accepts on the empty tape.

B.8 Maximality of our tractability conditions

Recall that Theorem 5 shows that we can get tractability by simultaneously restricting our constraints to be UIDs and our mappings to be ProjMaps. Recall also that a UID is an IncDep with at most one exported variable. Here we show that these restrictions are maximal in the following sense: if we increase from UIDs to LTGDs with frontier one we get intractability. We also get intractability if we stick with UIDs but we allow the mappings to be atomic. Let Fr1LTGD denote the LTGDs with at most one exported variable. In fact, we will show something stronger (here $\emptyset$ denotes no constraints):

Theorem 10. $\mathsf{Disclose}_{\mathsf{C}}(\emptyset, \mathsf{AtomMap})$ *and* $\mathsf{Disclose}_{\mathsf{C}}(\mathsf{Fr1LTGD}, \mathsf{ProjMap})$ *are both* NP*-hard.*

In order to prove our results, we will rely again on Proposition 1, which states that testing for disclosure is equivalent to evaluating the policy query on the result of the visible chase process. The process starts with the instance $\mathsf{Hide}_{\mathcal{M}}(\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}})$, which has source witnesses for each tuple in $\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}}$. It proceeds by alternating traditional chase steps and merge steps, which are applications of a SCEQrule. It is well known that query evaluation is NP-hard on arbitrary instances. But the constraints that we are considering in this section do not allow us to generate arbitrary instances as a visible sections. In this section we will exhibit a instance $\mathcal{D}$ on which query evaluation is NP-hard, but where $\mathcal{D}$ can be the result of the visible chase using AtomMaps but no constraints, or with a visible chase using Fr1LTGD constraints and ProjMaps.

The instance $\mathcal{D}$ $\mathcal{D}$ will have one relation R with 6 atoms. We present the content of R below. Empty cells are filled with fresh nulls, c is the only value shared by two tuples and n_i correspond to nulls that are shared inside a tuple:

a	b	$\neg a$	$\neg b$	$a \vee b$	$\neg^2 b$	d	$\neg d$
n_1	n_1	c	c	n_1		c	n_1
c	n_2	n_2	c	c			
n_3	c	c	n_3	c			
c	c	n_4	n_4	c			
c		n_5	n_5		c		
			c		n_6	n_6	c

Note that this is a *single-shared value instance*: only one value, namely c, is shared among multiple tuples. Such instances can be produced as the result of the visible chase over atomic mappings with no constraints. In this case:

$$
\begin{aligned}
R(y, y, x, x, y, v_1, x, y) &\rightarrow T_1(x) \\
R(x, y, y, x, x, v_1, v_2, v_3) &\rightarrow T_2(x) \\
R(y, x, x, y, x, v_1, v_2, v_3) &\rightarrow T_3(x) \\
R(x, x, y, y, x, v_1, v_2, v_3) &\rightarrow T_4(x) \\
R(x, u, y, y, v_1, x, v_2, v_3) &\rightarrow T_5(x) \\
R(v_1, v_2, v_3, x, v_4, y, y, x) &\rightarrow T_6(x)
\end{aligned}
$$

They can also be produced as the result of the visible chase over one projection mapping $A(x) \rightarrow T(x)$ with 6 Fr1LTGDs:

$$
\begin{aligned}
A(x) &\rightarrow R(y, y, x, x, y, v_1, x, y) \\
A(x) &\rightarrow R(x, y, y, x, x, v_1, v_2, v_3) \\
A(x) &\rightarrow R(y, x, x, y, x, v_1, v_2, v_3) \\
A(x) &\rightarrow R(x, x, y, y, x, v_1, v_2, v_3) \\
A(x) &\rightarrow R(x, u, y, y, v_1, x, v_2, v_3) \\
A(x) &\rightarrow R(v_1, v_2, v_3, x, v_4, y, y, x)
\end{aligned}
$$

The remainder of the argument is to show that CQ evaluation is NP-hard over this instance, via reduction from satisfiability of a propositional circuit (Circuit SAT).

General idea of the reduction The reduction that we provide will create a query Q for each instance $\mathcal{I}$ of Circuit SAT. Without loss of generality, we suppose that $\mathcal{I}$ is composed of wires $w_1, \dots, w_k$, of negation gates $N_1, \dots N_l$ and of binary OR gates $O_1, \dots, O_m$. Wire that are not the output of any

gate are the inputs of the circuit. We will suppose that the output corresponds to the wire 1.

We will build the query Q to contain conjuncts for each wire, each negation gate and each binary OR. Furthermore we will create a variable v_i for each wire w_i.

For the sake of readability, we present the conjuncts graphically, with each row representing an R atom. A row with entries $t_{j_1} \dots t_{j_k}$ represents an atom $R(\vec{w})$ where w_i is a fresh existentially quantified variable when the cell is empty and the variable t_{j_i} in the cell otherwise.

Wires For each wire w_i, we will force the value of its associated variable v_i to be either c (when the wire carries the value true) or n_1 (when the wire carries false).

For each wire i, we will have a conjunct:

a	b	$\neg a$	$\neg b$	$a \vee b$	$\neg^2 b$	d	$\neg d$
v_i	v_i						

For the variable v_1 corresponding to the output wire we also add a conjunct:

a	b	$\neg a$	$\neg b$	$a \vee b$	$\neg^2 b$	d	$\neg d$
		v_1	v_1				

Negation For each negation gate N_k, whose input is the wire i and output is the wire j, we will have the following conjuncts:

a	b	$\neg a$	$\neg b$	$a \vee b$	$\neg^2 b$	d	$\neg d$
v_i		r_k					
			r_k		p_k		
						p_k	v_j

Computing binary OR For the binary OR O_ℓ gate whose inputs are the wires v_i and v_j and the output is v_k, we introduce the following conjuncts:

a	b	$\neg a$	$\neg b$	$a \vee b$	$\neg^2 b$	d	$\neg d$
v_i		x_ℓ					
	v_j		y_ℓ				
		x_ℓ	y_ℓ	v_k			

Proof that this reduction captures Circuit-SAT Let us suppose that the circuit is satisfied. Towards showing that the query is satisfied in the instance $\mathcal{D}$, we first build a binding for the variables that are shared between multiple of the conjunct grouping above, which are exactly the "wire variables" v_i. We do this by setting $v_i = c$ when $w_i = \top$ and $v_i = n_1$ when $w_i = \bot$. We now show that this binding extends to a valuation making the query true. Since all the other variables are not shared between the conjunct groups, it suffices to show satisfiability of each conjunct group in isolation.

- We see that all the conjuncts corresponding to wires are satisfied (even the special conjunct corresponding to the output).
- For the negation gate N_k whose input is v_i and output is v_j. When $w_i = \top$ and thus $v_i = c$, we can set $r_k = n_5$, $p_k = c$ and satisfy all 3 conjuncts. When $w_i = \bot$ and thus $v_i = n_1$, we can set $r_k = c$ and $p_k = n_6$ and satisfy all 3 conjuncts.
- For an OR gate O_ℓ whose inputs are v_i and v_j, and whose output is v_k. There are four cases:
  - when $w_i = w_j = \top$ and thus $v_i = v_j = c$ we can set $x_\ell = y_\ell = n_4$
  - when $w_i = \bot$ and $w_j = \top$ and thus $v_i = n_1$, $v_j = c$ we can set $x_\ell = c$, $y_\ell = n_3$
  - when $w_j = \bot$ and $w_i = \top$ and thus $v_j = n_1$, $v_i = c$ we can set $x_\ell = n_2$, $y_\ell = c$
  - when $w_i = w_j = \bot$ and thus $v_i = v_j = n)1$ we can set $x_\ell = y_\ell = c$

In all cases, our conjuncts are satisfied.

Conversely, let us show that when the query is satisfied in our instance $\mathcal{D}$, then the circuit is satisfiable. Let h be a homomorphism from the query variables to values. Since we have wire conjuncts constraining v_i for each wire w_i, we can see that $h(v_i) = n_1$ or $h(v_i) = c$. We now consider the circuit assignment such that $w_i = \top$ when $h(v_i) = c$ and $w_i = \bot$ when $h(v_i) = n_1$. Let us show that this assignment witnesses the satisfiability of the circuit.

- The output wire is already constrained such that $h(v_1) \in \{n_1, c\}$ but it also has a special conjunct and the only remaining possibility for $h(v_1)$ is c and thus the output gate is set at $\top$.
- For each negation gate whose input is w_i and output is w_j:
  - when $h(v_i) = n_1$ then the conjunct holding v_i and r_k (i.e. the first row in the graphical representation) forces that $h(r_k) = c$. The conjunct holding r_k and p_k forces p_k to be a fresh null or n_6. But since p_k appears in the column $\neg^2 b$ and in the column d, we can only have $p_k = n_6$ and thus $v_j = c$.
  - when $h(v_i) = c$ then the conjunct holding v_i and r_k forces that $h(r_k) = n_5$ or $h(r_k) = n4$. Then the conjunct holding r_k and p_k forces p_k to be either a fresh null (when $h(r_k) = n_4$) or c (when $h(r_k) = n_5$). Since p_k appears in the column $\neg^2 b$ and in the column d we cannot have p_k fresh null, we conclude that$p_k = c$, and thus $v_j = n_1$.

  In both cases, the semantics of the negation gate is respected.
- Consider each OR gate whose inputs are w_i, w_j and output is w_k. First we have:
  - when $h(v_i) = n_1$ then necessarily $h(x_\ell) = c$
  - when $h(v_i) = c$ then $h(x_\ell) = n_2$ or $h(x_\ell) = n_4$ or $h(x_\ell) = n_5$
  - when $h(v_j) = n_1$ then necessarily $h(y_\ell) = c$
  - when $h(v_j) = c$ then $h(x_\ell) = n_3$ or $h(x_\ell) = n_4$.

  Therefore we see that:
  - when $h(v_i) = n_1 = h(v_j)$ then necessarily $h(x_\ell) = h(y_\ell) = c$ and thus $h(v_k) = n_1$

- when $h(v_i) = c$ and $h(v_j) = n_1$ then $h(x_\ell) = n_2$ and thus $h(v_k) = c$
- when $h(v_i) = n_1$ and $h(v_j) = c$ then necessarily $h(y_\ell) = n_3$ and thus $h(v_k) = c$
- when $h(v_j) = c = h(v_i)$ then $h(x_\ell) = n_4 = h(y_\ell)$ and thus $h(v_k) = c$

in all cases we do have that the semantics of the OR gate is respected.

All in all, we have seen that the circuit is satisfiable if and only if the query has a solution on the visible chase.

B.9 Lower bounds inherited from entailment

In the body of the paper we claimed that in several cases, we could show that the complexity of disclosure for a class was at least as hard as the complexity of query entailment for the class. We do *not* claim that there is a generic reduction from query entailment to disclosure. There is a simple reduction from entailment for special classes of instances to disclosure. More specifically, disclosure is easily seen to subsume entailment on instances of the form $\mathsf{Hide}_{\mathcal{M}}(\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}})$. But one needs to see that entailment on these specialized instances is as hard as entailment in general; this requires a separate argument for each class.

There are three cases of "lower bounds from entailment" that are used in the body of the paper: those whose lower bound is annotated with QEntail in Table 1. We give the details of each argument below.

In bounded arity, disclosure with IncDep source constraints and projection maps is NP-hard

We begin by showing that disclosure for IncDep source constraints and projection maps inherits the NP-hardness that is known for query entailment with IncDeps. We do this via a direct reduction from 3-coloring. We make use again of the characterization of disclosure using the visible chase.

Let us take a graph $G = (V, E)$ that is an input to 3-coloring. In our reduction, the schema, the constraints and the mapping will not depend on this actual graph reduced. Only the query will depend on the graph.

We will have a single source relation $OK(x, y, z)$ and one mapping $OK(x, y, z) \rightarrow M()$ to create canonical values for (x_0, y_0, z_0) in $\mathsf{Hide}_{\mathcal{M}}(\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}})$, which is the initial instance in the visible chase. Then we will use two IncDep constraints to create all permutations for these values: $OK(x, y, z) \rightarrow OK(x, z, y)$ and $OK(x, y, z) \rightarrow OK(y, x, z)$.

Because of the mapping, $\mathsf{Hide}_{\mathcal{M}}(\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}})$ will have three values x_0, y_0, z_0 with $OK(x_0, y_0, z_0)$. Then the constraints ensure that the canonical model contains the six permutations of arguments for OK: $OK(x_0, y_0, z_0)$, $OK(x_0, z_0, y_0)$, $OK(y_0, x_0, z_0)$, $OK(y_0, z_0, x_0)$, $OK(z_0, x_0, y_0)$, $OK(z_0, y_0, z_0)$.

In our query $|V|$ variables will capture the coloring of each node, we note $v(n)$ the variable associated with node n. For each $(f, t) \in E$ the query will include a conjunct $\exists c\ OK(v(f), v(t), c)$.

We sketch the correctness of this reduction. The three values x_0, y_0, z_0 in $\mathsf{Hide}_{\mathcal{M}}(\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}})$ encode the three possible colors in a coloring. The conjunct $\exists c\ OK(v(f), v(t), c)$ forbids the nodes f and t to be mapped to the same value (x_0, y_0 or z_0) as $\exists c\ OK(v, v, c)$ has no solution in $\mathsf{Hide}_{\mathcal{M}}(\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}})$. Therefore if we have disclosure have a 3-coloring.

Conversely, if we have a 3-coloring, we can find a solution for the query in the visible chase.

In general arity, disclosure with IncDep source constraints and projection maps is PSPACE-hard

Here we give a direct reduction from the implication problem for IncDeps, or equivalently, the entailment problem for a single-atom instance and an atomic query. This is known to be PSPACE-hard [Casanova *et al.*, 1984]. Given a problem $\Sigma \models R_1(\vec{x}) \subseteq R_2(\vec{X})$ (where $\vec{x}$ has no repeated variables and Σ is composed of IDs), we reduce it to disclosure problem with the query $R_1(\vec{X}) \wedge R_2(\vec{X})$ on the constraints Σ plus the mapping $V() := \exists \vec{x}\ \ R_1(\vec{x}) \rightarrow V()$.

In bounded arity, disclosure with FGTGD source constraints and projection maps is 2EXPTIME-hard

The last place where we claim that disclosure is at least as hard as entailment is for FGTGD source constraints and projection maps in bounded arity. Here we will proceed by modifying the reduction used in Theorem 8. In this proof we used mappings for two distinct purposes. The initialization mapping $\mathcal{T}_{init}()$ was used to generate some values in the initial instance of the visible chase. In the proof, this mapping is an atomic map but not a projection map; but we can easily change this to use a projection map and an LTGD.

The remaining maps are used to ensure that certain values get merged with c_{Crit} in the visible chase. Put another way, they are used to enforce certain SCEQrules. But with the mappings $\mathcal{T}_{init}()$ and $T_{c_{\mathsf{Crit}}}(x)$, we can ensure that the initial instance of the visible chase includes exactly one element satisfying IsCrit. Once we have done this, we can mimic a SCEQrule

$$\phi(\vec{x}) \rightarrow x_i = c_{\mathsf{Crit}}$$

by a source constraint

$$\phi(\vec{x}) \rightarrow \mathsf{IsCrit}(x_i)$$

This must be a FGTGD, since the frontier has size one. Transforming the mappings according to this methodology, while leaving the query the same as in Theorem 8 gives us a modification of the hardness proof using FGTGD source constraints and projection maps, as required.

C Refinements of our results

Atomic queries. We have focused in the body of the paper on policy queries given as general CQs. But almost all of our lower bounds can be seen to hold for atomic queries. The only exceptions are stated in Theorem 4 and Corollary 3, where we claim PTIME membership in bounded arity when restricting to atomic queries. Note that the NP-hardness bounds for general CQs corresponding to these upper bounds do not follow from our custom reductions, but using the simple reduction from entailment of CQs for the corresponding classes (e.g. IncDeps).

Non-Boolean queries. In this appendix we have provided details of our upper bounds, assuming for simplicity that the queries p are Boolean. But the proofs all extend to the non-Boolean case, as we now explain. To see this we need to go back to Theorem 1. We restate the theorem in a slightly different variant:

Theorem 11. *[Benedikt* et al.*, 2016] When source constraints are TGDs and mapping rules are given by CQ definitions, then if a disclosure of a CQ (Boolean or non-Boolean) occurs, then the source instance which witnesses this can be taken to be* $\mathcal{D}^{\mathcal{S}}_{\mathsf{Crit}}$.

The statement differs slightly from that of Theorem 1, since this version talks about getting an instance that agrees with $\mathcal{D}^{\mathcal{S}}_{\mathsf{Crit}}$ on the mapping images, rather than having one that extends $\mathsf{Hide}_{\mathcal{M}}(\mathcal{D}^{\mathcal{G}(\mathcal{M})}_{\mathsf{Crit}})$ and satisfies the constraints.

The important point is that the result holds for non-Boolean queries as well as Boolean queries. Note that for a Non-Boolean query $p(\vec{x})$, all the facts that an attacker will see in the mapping image of $\mathcal{D}^{\mathcal{S}}_{\mathsf{Crit}}$ will contain only the value c_{Crit}. Thus the only query answers that can be disclosed to the attacker will involve the value c_{Crit}. Inspection of each of the upper bound reductions will show that to detect such disclosures, it suffices to pre-process the query to add conjuncts $\mathsf{IsCrit}(x_i)$ for each variable x_i, treating the result as a Boolean query.

Note that this transformation converts atomic queries to queries consisting of a single atom and an additional set of unary atoms. However, this will not impact the PTIME claims in Theorem 4 and Corollary 3. For example in Theorem 4, we will need only to note that for atomic queries on a bounded arity schema, we will get an atomic query with a bounded number of additional atoms of the form $\mathsf{IsCrit}(x_i)$. Entailment of such queries over a bounded arity schema with IncDeps is still in PTIME.

Dependencies with multiple atoms in the head. In some of our upper bound proofs, we assumed that the dependencies had a single atom in the head for simplicity, even when the classes in question (e.g. GTGDs) does not impose this. In our results that are stated for general arity, this assumption can be made without loss of generality, since one can simplify the heads by introducing intermediate predicates. In bounded arity, one must take some care, since one cannot polynomially reduce to the case of a single atom in the head. All of our results for bounded arity do in fact hold as stated, without any additional restrictions on the head. We explain how the argument needs to be customized for the most subtle case, Theorem 4.

Recall that the bounded arity case of Theorem 4 starts with the critical-instance rewriting, which reduces to reasoning with Guarded TGDs having a fixed side signature, the unary predicate $\mathsf{IsCrit}(x)$. The linearization of [Amarilli and Benedikt, 2018a; Amarilli and Benedikt, 2018b], applied in this context, proceeds in two steps. First we generate all derived rules of the form:

$$R(\vec{x}) \wedge \bigwedge_i \mathsf{IsCrit}(x_i) \rightarrow \mathsf{IsCrit}(x_j)$$

Notice that these are *full-dependencies*: no existentials in the head. This generation can be done inductively, via the dynamic programming steps in [Amarilli and Benedikt, 2018b]: one inductive steps composes a derived rule with one of the original non-full dependencies. A second step composes two derived full rules. This can be applied directly to the case of rules with multiple atoms in the head.

After this is done, the second step of linearization moves to an extended signature described as follows: for every relation R of arity k in the original signature (without IsCrit), and for each set of positions P of R, we introduce predicates R^P of arity k. Informally, $R^P(\vec{x})$ stands in for $R(\vec{x}) \wedge \bigwedge_{i \in P} \mathsf{IsCrit}(x_i)$. We lift every original dependency:

$$R(\vec{x}) \wedge \bigwedge_{i \in P} \mathsf{IsCrit}(x_i) \rightarrow \exists \vec{y} \bigwedge H_j(\vec{t_j})$$

to a linear TGD:

$$\bigwedge R^P(\vec{x}) \rightarrow \exists \vec{y} \bigwedge H_j^{P_j}(\vec{t_j})$$

where P_j contains the positions corresponding to exported variables in P. We lift every derived full dependency of the form:

$$R(\vec{x}) \wedge \bigwedge_{i \in P} \mathsf{IsCrit}(x_i) \rightarrow \mathsf{IsCrit}(x_j)$$

to a linear TGD:

$$R^P(\vec{x}) \rightarrow R^{P \cup \{j\}}(\vec{x})$$

Finally we have linear TGD asserting that the semantics of R^P become stronger as one adds to the set of positions P:

$$R^{P'}(\vec{x}) \rightarrow R^P(\vec{x})$$

for $P \subset P'$.

We rewrite the query to the extended signature in the analogous way. The correctness of this transformation is given by an argument identical to that in the single-headed case in [Amarilli and Benedikt, 2018b].

Note that this transformation is in PTIME when the arity is fixed. It reduces us to an entailment problem with LTGDs, still with bounded arity, but with multiple atoms in the head. Such an entailment problem can be shown to be in NP using a simple variation of the algorithm for IncDeps of [Johnson and Klug, 1984].