Static Analysis of Graph Database Transformations

Iovka Boneva, Benoit Groz, Jan Hidders, Filip Murlak, Slawomir Staworko

▶ **To cite this version:**

Iovka Boneva, Benoit Groz, Jan Hidders, Filip Murlak, Slawomir Staworko. Static Analysis of Graph Database Transformations. Symposium on Principles of Database Systems, Jun 2023, Seattle, United States. hal-03937274v1

HAL Id: hal-03937274

https://hal.science/hal-03937274v1

Submitted on 6 Apr 2023 (v1), last revised 17 Apr 2023 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Static Analysis of Graph Database Transformations

Anonymous
Iovka Boneva
iovka.boneva@univ-lille.fr
Univ. Lille, CNRS, UMR 9189
CRIStAL, F-59000 Lille
France

Benoît Groz
groz@lri.fr
LISN, CNRS (UMR 9015)
Paris-Saclay University
France

Jan Hidders
j.hidders@bbk.ac.uk
Birkbeck, University of London
United Kingdom

Filip Murlak
filip.murlak@mimuw.edu.pl
University of Warsaw
Poland

Sławek Staworko
slawek.staworko@relational.ai
RelationalAI
Univ. Lille, CNRS, UMR 9189
CRIStAL, F-59000 Lille
France

ABSTRACT

We investigate graph transformations, defined using Datalog-like rules based on acyclic conjunctive two-way regular path queries (acyclic C2RPQs), and we study two fundamental static analysis problems: *type checking* and *equivalence* of transformations in the presence of graph schemas. Additionally, we investigate the problem of *target schema elicitation*, which aims to construct a schema that closely captures all outputs of a transformation over graphs conforming to the input schema. We show all these problems are in EXPTIME by reducing them to C2RPQ containment modulo schema; we also provide matching lower bounds. We use *cycle reversing* to reduce query containment to the problem of unrestricted (finite or infinite) satisfiability of C2RPQs modulo a theory expressed in a description logic.

CCS CONCEPTS

• **Theory of computation** → *Logic and databases.*

ACM Reference Format:
Anonymous, Iovka Boneva, Benoît Groz, Jan Hidders, Filip Murlak, and Sławek Staworko. 2023. Static Analysis of Graph Database Transformations. In *Proceedings of 42th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS'23).* ACM, New York, NY, USA, 22 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The growing adoption of graph databases calls for suitable data processing methods. Query languages for graph databases typically define their semantics as a set of tuples, which alone is inadequate for scenarios such as (materialized) graph database views and data migration in the context of schema evolution [10]. A more adequate mechanism is that of a *transformation*, which takes a graph as input and produces a graph on the output.

Example 1.1. Consider a scenario where the schema of a medical knowledge graph undergoes changes due to advances in the understanding of biomolecular processes. The purpose of this knowledge graph is to catalog vaccines based on the antigen they are designed to target and to identify the pathogens that exhibit the antigens, each antigen being exhibited by at least one pathogen. Additionally, some pairs of antigens are known to be *cross reacting*: if a vaccine v targets an antigen x that is cross reacting with an antigen y, then v also targets y. Thus, the set of all antigens targeted by a vaccine is represented implicitly.

The schema S_0 of the original knowledge graph is presented in Figure 1 as a graph itself. It specifies the allowed node and edge labels, and expresses participation constraints on edges in a manner that is typical for data modeling languages, e.g., $A \xrightarrow{* \; r \; 1} B$ indicates that every A-node has one outgoing r-edge to a B-node but a B-node may have arbitrarily many incoming r-edges from A-nodes.

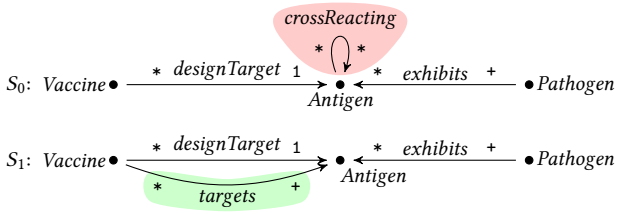

Figure 1: Evolving schema of a medical knowledge graph.

Now, suppose that new findings refute the rule of cross-reactivity of antigens. The *cross-reacting* edges between antigens are no longer adequate for representing information about the antigens that a vaccine targets, and so, in the new schema S_1, this information is recorded explicitly with *targets* edges. Since up to that point, the knowledge graph did not contain any data points that contradicted the cross-reactivity rule, the logic of the rule can be used to transform the old knowledge graph to one that conforms to the new schema. Afterwards *cross-reacting* edges are removed. □

In the present paper, we study two classical problems of static analysis on graph transformations: *type checking*, that verifies if for

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
PODS'23, June 18–23, 2023, Seattle, WA, USA
© 2023 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

every graph conforming to the source schema the transformation outputs a graph conforming to the target schema, and *equivalence*, that verifies if two transformations produce the same output for every graph conforming to the source schema. Additionally, when the target schema is not known, we investigate the problem of *target schema elicitation* that constructs the containment-minimal target schema that captures the graphs produced by the transformation.

We study *executable* graph transformations defined with Datalog-like rules. The rules specify how to construct the output graph from the results of regular path queries evaluated over the input graph. To allow multiple copies of the same input node the rules use *node constructors*, essentially explicit Skolem functions that create nodes. As an example, the cross-reactivity rule from Example 1.1 gives rise to the following graph transformation rule.

$$targets(f_V(x), f_A(y)) \leftarrow (designTarget \cdot crossReacting^*)(x, y)\,,$$

where $f_V(x)$ and $f_A(y)$ are constructors of *Vaccine* and *Antigen* nodes respectively.

We investigate transformations that use only *acyclic two-way conjunctive regular path queries* (acyclic C2RPQs), which is arguably of practical relevance in the context of graph transformations. For instance, we have found no cyclic queries in the transformations implementing graph data migration between consecutive versions of the FHIR data format [32, 50] (Fast Healthcare Interoperability Resources is an international standard for interchange of medical healthcare data). Our constructions rely on acyclicity of C2RPQs to obtain relatively low computational complexity. We argue that the acyclicity assumption cannot be lifted without a significant complexity increase (see Section 7).

Node constructors are closely related to object creating functions [34, 35]. Our use of node constructors is inspired by analogous constructions in transformation languages such as R2RML [16, 20, 52], where node IRIs are typically obtained by concatenation of a URL prefix and the key values of a object represented by the constructed node. Our node constructors can have an arbitrary arity, thus allowing for instance to create nodes in the target graph that represent relationships (edges) between nodes in the source graph. To isolate the concern of possible overlaps between node constructors, we make the natural assumption that node constructors are injective, have pair-wise disjoint ranges, and for every node kind (label) a single dedicated node constructor is used. These assumptions allow us to remove the need to analyze the definitions of node constructors, which is out of the scope of the present paper, and they are consistent with how the analogous constructions are used in languages such as R2RML and FHIR mapping language.

For schemas, we employ a natural formalism of *graph schemas with participation constraints*, inspired by standard data modeling languages such as Entity-Relationship diagrams [15], and already studied, for instance, in the context of graph database evolution [10]. Such schemas allow to declare the available labels of nodes and edges and to express participation constraints. In contrast to more expressive languages as ShEx and SHACL [17, 53], our formalism allows a *single label per node*, which determines the node type. Thus, roughly speaking, our schema formalism is to ShEx and SHACL what DTD is to XML Schema.

The key contributions of the present paper are as follows.

(1) We define graph database transformations and we reduce the problems of interest to *containment* of C2RPQs in unions of acyclic C2RPQs *modulo schemas*.
(2) We reduce the query containment problem to the unrestricted (finite or infinite) satisfiabilty of a C2RPQ modulo a set of constraints expressed in the Horn fragment of a description logic known as $\mathcal{ALCIF}$. The reduction involves an application of the *cycle reversing* technique [18, 36], carefully tailored to our needs.
(3) The unrestricted satisfiability problem for $\mathcal{ALCIF}$ can be solved in EXPTIME owing to a simple model property [14]. We provide a new algorithm with *improved complexity bounds*, needed to deal with the exponential blow-up inherent to cycle reversing. We also reformulate the simplicity of models in terms of a graph-theoretical notion of (k, l)*-sparsity* [41], which allows to simplify the reasoning.

These reductions allow to solve all problems of interest in EXPTIME and we also establish the matching lower bounds.

The paper is organized as follows. In Section 2 we discuss related work. In Section 3 we introduce basic notions. In Section 4 we define graph transformations and the problems of interest, which we reduce to query containment modulo schema. In Section 5 we reduce the latter to satisfiability of a query modulo Horn-$\mathcal{ALCIF}$ theory, which we solve in Section 6. In Section 7 we summarize our findings and identify directions of future work. Because of space restrictions full proofs and some standard definitions have been moved to Appendix.

2 RELATED WORK

Graph-based data models have been proposed in various forms and shapes since the 1980s [4]. The proposals in the 1980s and 1990s included labeled graphs [30], graphs where certain nodes represent complex values [31, 40], graphs where nodes have associated complex values [1, 2], and graphs where nodes are associated with nested graphs [42]. More recently the RDF data model [28] and the Property Graph data model [3] have become popular. RDF graphs are similar to labeled graphs except that nodes are unlabeled and participate in at least one edge, and the labels of edges can be nodes and participate in edges. Property Graphs are also similar to labeled graphs except that nodes and edges have multiple labels and properties, and edges have identity. In our work we assume one of the simplest models, namely, labeled graphs where nodes have multiple labels and edges have a single label; our schemas require exactly one label per node. Since we focus here on transformations of the graph structure, we have no explicit notion of value associated with nodes and edges, but there are straightforward ways of adding this, as is done for example in [30].

The term **graph transformations** can refer to different formalisms [51]: the purpose of graph grammars is to define graph languages; algebraic graph transformations are mainly used to model systems with infinite behavior and are not functional (they produce multiple outputs on single input). Therefore, not only are these formalisms ill-suited for defining transformations of graph databases, but also the problems studied for them are unrelated to the problems we study here. Monadic second-order (MSO) graph

transductions [19] can capture our transformations only when restricted to unary node constructors; moreover, resorting to MSO logic typically incurs a prohibitive complexity overhead.

Transformation languages for graph databases are often based on Datalog extended with node-creation syntax in the head of the rules. It could be just a variable that is not bound in the body of the rule, like in IQL [2] and G-Log [48]; this ensures a fresh node is created for each valuation that makes the body true. Another option is to replace the unbound variable with a term consisting of a constructor function (sometimes called a Skolem function) applied to bound variables, like in O-logic [43] and F-logic [38]; the constructor creates a fresh node when called for the first time for certain arguments, and after that the same node for the same arguments. We adopt the idea of node constructors because we believe it provides a powerful and intuitive way to control the identity of new nodes. A different proposal, based on structural recursion, is offered by UnQL [11], but the underlying data model considers graphs equivalent if they are bisimilar, which makes the expressive power quite different. Finally, popular graph query languages such as SPARQL and Cypher have the option to output a graph and can therefore express graph transformations, but they are far too expressive to lend themselves to static analysis.

In the context of data exchange, **schema mappings** provide a declarative way to define database transformations [7, 12, 22]. Our transformations could be simulated by considering canonical solutions for plain SO-tgds [5] extended to allow acyclic C2RPQs in rule bodies. Note, however, that equivalence is undecidable for plain SO-tgds with keys [23], and open for plain SO-tgds [39].

The **static type checking problem** originates in formal language theory and has been studied for finite state transducers on words and for various kinds of tree transducers, including some designed to capture XML transformation languages [44–47]. Type checking has also been studied for graph transformations. In [31] labelled graphs are transformed using addition, deletion, and reduction operations, and type checking is investigated for schemas similar to ours but without participation constraints. The typing problem for UnQL is studied in [37], but the approach relies on schemas specifying graphs up to bisimulation, which limits their power to express participation constraints. Regarding transformations defined by schema mappings, if the mapping does not define target constraints, then the target schema is simply a relational signature and type checking is reduced to trivial syntactic check, and as such it is irrelevant. This is most often the case for graph schema mappings [7, 12], with seldom exceptions such as [9] for mapping relational to graph-shaped data. Their notion of consistency is related to type checking, but is studied for a simpler formalism without path queries. In the context of XML schema mappings, absolute consistency can be seen as a counterpart of type checking for non-functional transformations [8].

3 PRELIMINARIES

Graphs. We fix an enumerable set $\mathcal{N}$ of node identifiers, an enumerable set Γ of node labels, and an enumerable set Σ of edge labels. We work with labeled directed graphs, and in general, a node may have multiple labels while an edge has precisely one label. We allow, however, multiple edges between the same pair of nodes, as long as these edges have different labels. We model graphs as relational structures over unary relation symbols Γ and binary relation symbols Σ. That is, a graph G is a pair $\big(dom(G), \cdot^G\big)$ where $dom(G) \subseteq \mathcal{N}$ is the set of nodes of G and the function $\cdot^G$ maps each $A \in \Gamma$ to a set $A^G \subseteq dom(G)$ and each $r \in \Sigma$ to a binary relation $r^G \subseteq dom(G) \times dom(G)$. A graph G is *finite* if $dom(G)$ is finite and A^G and r^G are empty for all but finitely many $A \in \Gamma$ and $r \in \Sigma$. In the sequel, we use $u, v, \ldots$ to range over node identifiers, $A, B, C, \ldots$ to range over node labels, and $r, r', \ldots$ to range over edge labels. Also, we use r^- for inverse edges and let $(r^-)^G = \big\{(u,v) \mid (v,u) \in r^G\big\}$. We let $\Sigma^\pm = \Sigma \cup \{r^- \mid r \in \Sigma\}$ and use $R, R', \ldots$ to range over $\Sigma^\pm$.

Schemas. We consider a class of schemas that constrain the number of edges between nodes of given labels and we express these constraints with the usual symbols: `?` for at most one, `1` for precisely one, `+` for at least one, `*` for arbitrary many, and `0` for none. A *schema* is a triple $S = (\Gamma_S, \Sigma_S, \delta_S)$, where $\Gamma_S \subseteq \Gamma$ is a finite set of allowed node labels, $\Sigma_S \subseteq \Sigma$ is a finite set of allowed edge labels, and $\delta_S : \Gamma_S \times \Sigma_S^\pm \times \Gamma_S \to \{\mathtt{?}, \mathtt{1}, \mathtt{+}, \mathtt{*}, \mathtt{0}\}$. Schemas can be presented as graphs themselves, interpreted as illustrated next.

Example 3.1. Take the schema S_0 in Figure 1 and consider, for instance, the *designTarget* edge. It indicates that every *Vaccine* has a single design target *Antigen*, in symbols

$$\delta_{S_0}(\mathit{Vaccine}, \mathit{designTarget}, \mathit{Antigen}) = \mathtt{1}\,,$$

and that every *Antigen* may be the design target of an arbitrary number of *Vaccine*s, in symbols

$$\delta_{S_0}(\mathit{Antigen}, \mathit{designTarget}^-, \mathit{Vaccine}) = \mathtt{*}\,.$$

Edges that are not present are implicitly forbidden, e.g., no *exhibits* edge is allowed from *Vaccine* to *Pathogen*:

$$\delta_{S_0}(\mathit{Vaccine}, \mathit{exhibits}, \mathit{Pathogen}) = \mathtt{0}\,,$$
$$\delta_{S_0}(\mathit{Pathogen}, \mathit{exhibits}^-, \mathit{Vaccine}) = \mathtt{0}\,. \qquad \square$$

Now, a graph G *conforms* to a schema S if 1) every node in G has a single node label in Γ_S and every edge has a label in Σ_S, and 2) for all $A, B \in \Gamma_S$ and $R \in \Sigma_S^\pm$, for every node with label A the number of its R-successors with label B is as specified by $\delta_S(A, R, B)$. By $L(S)$ we denote the set of all *finite* graphs that conform to S.

Queries. We work with conjunctive two-way regular path queries (C2RPQs) that have the form

$$q(\bar{x}) = \exists \bar{y}.\varphi_1(z_1, z_1') \wedge \ldots \wedge \varphi_k(z_k, z_k')\,,$$

where $\bar{x} = \{z_1, z_1', \ldots, z_k, z_k'\} \setminus \bar{y}$ and for every $i \in \{1, \ldots, k\}$ the formula φ_i is a regular expression that follows the grammar

$$\varphi ::= \varnothing \mid \epsilon \mid A \mid R \mid \varphi \cdot \varphi \mid \varphi + \varphi \mid \varphi^*\,,$$

where $A \in \Gamma$ matches nodes, $R \in \Sigma^\pm$ matches edges, ϵ matches empty paths, and $\varnothing$ matches no path. The semantics of C2RPQs is defined in the standard fashion [13] and we denote the set of *answers* to $q(\bar{x})$ in G by $[q(\bar{x})]^G$ (see Appendix A for formal definitions).

Example 3.2. Recall the schema S_0 in Figure 1. The following query selects vaccines together with the antigens they are designed to target or target through cross-reaction.

$q(x, y) = (\mathit{Vaccine} \cdot \mathit{designTarget} \cdot \mathit{crossReacting}^* \cdot \mathit{Antigen})(x, y)$. $\square$

In the sequel, we abuse notation and write trivial atoms of the form $\varnothing(x,x)$, $\epsilon(x,x)$, and $A(x,x)$ simply as $\varnothing(x)$, $\epsilon(x)$, and $A(x)$ respectively. The subclass of *acyclic* C2RPQs consists of queries that have no cycle in the implied multigraph of variables containing for every atom an edge connecting the two variables of the atom (except for trivial atoms).

A *Boolean* C2RPQ q has all its variables existentially quantified, and it may have only a single answer, the empty tuple, in which case, we say that q is *satisfied* in G and write $G \models q$. We also use *unions* of C2RPQs (abbreviated as UC2RPQs) represented as sets of C2RPQs $Q(\bar{x}) = \{q_1(\bar{x}), \ldots, q_k(\bar{x})\}$ and extend the notions of answers, satisfaction, and acyclicity to UC2RPQs in the natural fashion. Given two UC2RPQs $P(\bar{x})$ and $Q(\bar{x})$, and a schema S, we say that $P(\bar{x})$ *is contained in* $Q(\bar{x})$ *modulo* S, in symbols $P(\bar{x}) \subseteq_S Q(\bar{x})$, if $[P(\bar{x})]^G \subseteq [Q(\bar{x})]^G$ for every $G \in L(S)$.

Description logics. We operate on properties of graphs formulated in the *description logic* $\mathcal{ALCIF}$ (and its fragments) [6]. In description logics, elements of Γ and Σ are called *concept names* and *role names*, respectively. $\mathcal{ALCIF}$ allows to build more complex concepts with the following grammar:

$$C ::= \bot \mid A \mid C \sqcap C \mid \neg C \mid \exists R.C \mid \exists^{\leq 1} R.C\,,$$

where $A \in \Gamma$ and $R \in \Sigma^{\pm}$. We also use additional operators that are redundant but useful when defining fragments; for brevity we introduce them as syntactic sugar: $\top := \neg\bot$, $C_1 \sqcup C_2 := \neg(\neg C_1 \sqcap \neg C_2)$, $\forall R.C := \neg\exists R.\neg C$, $\nexists R.C := \neg\exists R.C$. We extend the interpretation $\cdot^G$ to complex concepts as follows:

$$\bot^G = \emptyset\,, \quad (C_1 \sqcap C_2)^G = C_1^G \cap C_2^G\,, \quad (\neg C)^G = dom(G) \setminus C^G\,,$$
$$(\exists R.C)^G = \left\{u \in dom(G) \mid \exists v.\ (u,v) \in R^G \wedge v \in C^G\right\},$$
$$(\exists^{\leq 1} R.C)^G = \left\{u \in dom(G) \mid \exists^{\leq 1} v.\ (u,v) \in R^G \wedge v \in C^G\right\}.$$

Statements in description logics have the form of *concept inclusions*,

$$C \sqsubseteq D$$

where C and D are concepts. A graph G *satisfies* $C \sqsubseteq D$, in symbols $G \models C \sqsubseteq D$, if $C^G \subseteq D^G$. A set $\mathcal{T}$ of concept inclusions is traditionally called a *TBox* and we extend satisfaction to TBoxes in the canonical fashion: $G \models \mathcal{T}$ if $G \models C \sqsubseteq D$ for each $C \sqsubseteq D \in \mathcal{T}$.

In the *Horn fragment* of $\mathcal{ALCIF}$, written Horn-$\mathcal{ALCIF}$, we only allow concept inclusions in the following normal forms:

$$K \sqsubseteq A\,, \qquad K \sqsubseteq \bot\,, \qquad K \sqsubseteq \forall R.K'\,,$$
$$K \sqsubseteq \exists R.K'\,, \qquad K \sqsubseteq \nexists R.K'\,, \qquad K \sqsubseteq \exists^{\leq 1} R.K'\,,$$

where $A \in \Gamma$, $R \in \Sigma^{\pm}$, and K, K' are intersections of concept names (intersection of the empty set of concepts is $\top$). If statements of the form $K \sqsubseteq A_1 \sqcup A_2 \sqcup \cdots \sqcup A_n$ are allowed too, then we recover the full power of $\mathcal{ALCIF}$ (up to introducing auxiliary concept names).

Participation constraints of schemas can be expressed with simple Horn-$\mathcal{ALCIF}$ statements as illustrated in following example.

Example 3.3. For instance, the assertion in S_0 (Figure 1) that *Pathogen* manifests at least one *Antigen* is expressed with the statement $Pathogen \sqsubseteq \exists exhibits.Antigen$. The assertion that an *Antigen* may be exhibited by an arbitrary number of *Pathogens* needs no Horn-$\mathcal{ALCIF}$ statement. However, statements are needed for implicitly forbidden edges, e.g., $Vaccine \sqsubseteq \nexists exhibits.Antigen$. □

4 GRAPH TRANSFORMATIONS

We propose transformations of graphs defined with Datalog-like rules that use acyclic C2RPQs in their bodies. To allow multiple copies of the same source node we use node constructors. Formally, a *k-ary node constructor* is a function $f : \mathcal{N}^k \to \mathcal{N}$ and we denote the set of node constructors by $\mathcal{F}$. To remove the concern of overlapping node constructors, and the need to analyze their definitions, we assume that for every node label $A \in \Gamma$ we have precisely one node constructor f_A, all node constructors are injective, and their ranges are pairwise disjoint.

We introduce two kinds of *graph transformation rules*: node rules and edge rules. A *node rule* has the form

$$A\big(f_A(\bar{x})\big) \leftarrow q(\bar{x})\,,$$

where $A \in \Gamma$, $f \in \mathcal{F}$, and q is an acyclic C2RPQ. An *edge rule* has the form

$$r\big(f(\bar{x}), f'(\bar{y})\big) \leftarrow q(\bar{x}, \bar{y})\,,$$

where $r \in \Sigma$, $f, f' \in \mathcal{F}$, and q is an acyclic C2RPQ. Now, a *graph transformation* T is a finite set of graph transformation rules. By Γ_T and Σ_T we denote the finite sets of node and edge labels, respectively, used in the heads of the rules of T.

Example 4.1. Below we present rules defining the transformation T_0 of the medical database, described in Example 1.1. We use 3 unary node constructors $f_A(x)$ for *Antigen* nodes, $f_P(x)$ for *Pathogen* nodes, and $f_V(x)$ for *Vaccine* nodes.

$$\begin{aligned}
Vaccine(f_V(x)) &\leftarrow (Vaccine)(x)\,,\\
Antigen(f_A(x)) &\leftarrow (Antigen)(x)\,,\\
designTarget(f_V(x), f_A(y)) &\leftarrow (designTarget)(x,y)\,,\\
targets(f_V(x), f_A(y)) &\leftarrow (designTarget \cdot crossReacting^*)(x,y)\,,\\
Pathogen(f_P(x)) &\leftarrow (Pathogen)(x)\,,\\
exhibits(f_P(x), f_A(y)) &\leftarrow (exhibits)(x)\,.
\end{aligned}$$
□

Now, given a graph G and a graph transformation T the *result of applying* T *to* G is a graph $T(G)$ such that (for $A \in \Gamma$ and $r \in \Sigma$)

$$A^{T(G)} = \left\{f_A(t) \;\middle|\; A\big(f_A(\bar{x})\big) \leftarrow q(\bar{x}) \in T,\ t \in [q(\bar{x})]^G\right\},$$
$$r^{T(G)} = \left\{\big(f(t), f'(t')\big) \;\middle|\; r\big(f(\bar{x}), f'(\bar{y})\big) \leftarrow q(\bar{x},\bar{y}) \in T,\ (t,t') \in [q(\bar{x},\bar{y})]^G\right\}.$$

We are interested in the following two classical static analysis tasks.

Type checking Given a transformation T, a source schema S, and a target schema S' check whether for every G that conforms to S the output of transformation $T(G)$ conforms to S'.

Equivalence Given a source schema S and two transformations T_1 and T_2 check whether T_1 and T_2 agree on every graph that conforms to S.

In settings where the target schema is not known, it might be useful to construct one.

Schema elicitation Given a transformation T and a source schema S, construct the containment minimal target schema S' such that $T(G) \in L(S')$ for every $G \in L(S)$.

We observe that $T(G)$ may have nodes with no label, which may preclude it from satisfying any schema, and consequently, schema elicitation may also return error.

We prove the main result by reducing the problems of interest to query containment modulo schema (and vice versa), which we later show to be EXPTIME-complete.

Theorem 4.2. *Type checking, schema elicitation, and equivalence of graph transformations are* EXPTIME*-complete.*

We outline the main ideas of the proof by illustrating how a transformation T can be analyzed with a toolbox of methods based on query containment modulo source schema S. We formulate these methods with an entailment relation:

$$(T,S) \models K \sqsubseteq K' \quad \textit{iff} \quad T(G) \models K \sqsubseteq K' \text{ for every } G \in L(S).$$

W.l.o.g. we assume that every rule of transformation T is trim i.e., it uses in its body a query $q(\bar{x})$ that is satisfiable modulo S, in symbols $\exists \bar{x}.q(\bar{x}) \not\subseteq_S \varnothing$; otherwise, T can be trimmed.

First, we group queries from rules of T based on the labels of nodes and edges they create. For $A, B \in \Gamma_T$ and $r \in \Sigma_T$ we define

$$\begin{aligned}
&Q_A(\bar{x}) = \left\{ q(\bar{x}) \;\middle|\; A\big(f_A(\bar{x})\big) \leftarrow q(\bar{x}) \in T \right\},\\
&Q_{A,r,B}(\bar{x},\bar{y}) = \left\{ q(\bar{x},\bar{y}) \;\middle|\; r\big(f_A(\bar{x}), f_B(\bar{y})\big) \leftarrow q(\bar{x},\bar{y}) \in T \right\},\\
&Q_{A,r^-,B}(\bar{x},\bar{y}) = \left\{ q(\bar{y},\bar{x}) \;\middle|\; r\big(f_B(\bar{y}), f_A(\bar{x})\big) \leftarrow q(\bar{y},\bar{x}) \in T \right\}.
\end{aligned}$$

In essence, $Q_A(\bar{x})$ identifies tuples over the input graph that yield a node constructed with f_A and with label A while $Q_{A,R,B}(\bar{x},\bar{y})$ identifies tuples that yield R-edges from a node created with f_A to a node created with f_B.

Example 4.3. A couple of examples of above queries for the transformation T_0 in Example 4.1 follow.

$$\begin{aligned}
&Q_{Vaccine}(x) = (Vaccine)(x),\\
&Q_{Vaccine,targets,Antigen}(x,y) = (designTarget \cdot crossReacting^*)(x,y),\\
&Q_{Vaccine,designTarget,Antigen}(x,y) = (designTarget)(x,y).
\end{aligned}$$ □

Since an edge rule does not assign labels to nodes it creates, the result of a transformation may be a graph with nodes without a label. Such a situation precludes type checking from passing and prevents schema elicitation from producing meaningful output. Consequently, we first verify that every node in every output graph has exactly one label, in symbols $(T,S) \models \top \sqsubseteq \bigsqcup \Gamma_T$, where $\bigsqcup\{A_1,\ldots,A_k\}$ is a shorthand for $A_1 \sqcup \ldots \sqcup A_k$. We prove the following (Lemma B.6).

$$\begin{aligned}
&(T,S) \models \top \sqsubseteq \bigsqcup \Gamma_T \quad \textit{iff}\\
&\qquad \exists \bar{y}.Q_{A,R,B}(\bar{x},\bar{y}) \subseteq_S Q_A(\bar{x}) \quad \text{for all } A,B \in \Gamma_T \text{ and } R \in \Sigma_T^{\pm}.
\end{aligned}$$

We point out that the restriction of one node constructor per node label ensures that each node of the output has at most one label.

Example 4.4. Take T_0 from Example 4.1 and S_0 in Figure 1. Verifying that $(T_0,S_0) \models \top \sqsubseteq \bigsqcup \Gamma_{T_0}$ requires a number of containment tests including the following two.

$$\begin{aligned}
&\exists y.(designTarget \cdot crossReacting^*)(x,y) \subseteq_{S_0} (Vaccine)(x),\\
&\exists y.(designTarget)(x,y) \subseteq_{S_0} (Vaccine)(x).
\end{aligned}$$ □

Now, to perform type checking against a given target schema S', we verify that $\Gamma_T \subseteq \Gamma_{S'}$ and $\Sigma_T \subseteq \Sigma_{S'}$. Then, we take the TBox $\mathcal{T}_{S'}$ of concept inclusions that expresses participation constraints of the target schema S' and we verify that $(T,S) \models \mathcal{T}_{S'}$. Type checking succeeds if and only if all the above tests succeed (Lemma B.2).

The TBox $\mathcal{T}_{S'}$ consists of statements from a small fragment $\mathcal{L}_0$ of Horn-$\mathcal{ALCIF}$ which allows only statements of the forms

$$A \sqsubseteq \exists R.B, \qquad A \sqsubseteq \nexists R.B, \qquad A \sqsubseteq \exists^{\leq 1} R.B,$$

where $A, B \in \Gamma$ and $R \in \Sigma^{\pm}$. The entailment of such statements is also reduced to query containment (Lemma B.7):

$$\begin{aligned}
&(T,S) \models A \sqsubseteq \exists R.B \quad \textit{iff} \quad Q_A(\bar{x}) \subseteq_S \exists \bar{y}.Q_{A,R,B}(\bar{x},\bar{y}),\\
&(T,S) \models A \sqsubseteq \nexists R.B \quad \textit{iff} \quad \exists \bar{y}.Q_A(\bar{x}) \wedge Q_{A,R,B}(\bar{x},\bar{y}) \subseteq_S \bigwedge_i \varnothing(x_i),\\
&(T,S) \models A \sqsubseteq \exists^{\leq 1} R.B \quad \textit{iff}\\
&\qquad \exists \bar{x}.Q_A(\bar{x}) \wedge Q_{A,R,B}(\bar{x},\bar{y}) \wedge Q_{A,R,B}(\bar{x},\bar{z}) \subseteq_S \bigwedge_i \epsilon(y_i,z_i).
\end{aligned}$$

Example 4.5. Take the transformation T_0 and the schemas S_0 and S_1 in Figure 1. The schema S_1 requires every vaccine to target at least one antigen, in symbols $Vaccine \sqsubseteq \exists targets.Antigen$. This statement is entailed by T_0 and S_0 if and only if the following holds

$$(Vaccine)(x) \subseteq_{S_0} \exists y.(designTarget \cdot crossReacting^*)(x,y).$$ □

For schema elicitation, we use a close correspondence between schemas and $\mathcal{L}_0$ TBoxes. It is sufficient to construct the TBox $\mathcal{T}$ containing all $\mathcal{L}_0$ statements that are entailed by T and S; $\mathcal{T}$ corresponds to the containment-minimal target schema (Lemma B.5).

Finally, the equivalence of two transformations T_1 and T_2 is essentially the equivalence (modulo S) of the respective queries Q_A and $Q_{A,R,B}$ of both transformations (Lemma B.8). Naturally, query equivalence is reduced to query containment, as usual.

We have shown that type checking, schema elicitation, and equivalence of graph transformations are Turing-reducible in polynomial time to testing containment of UC2RPQs in acyclic UC2RPQs modulo schema. We also show polynomial-time reductions of containment of 2RPQs modulo schema to all above problems of interest (Lemma F.2). With that, Theorem 4.2 follows from Theorem 5.1.

5 QUERY CONTAINMENT MODULO SCHEMA

The aim of this section is to show the following result.

Theorem 5.1. *Containment of UC2RPQs in acyclic UC2RPQs modulo schema is* EXPTIME*-complete.*

The lower bound can be derived from the EXPTIME-hardness of unrestricted containment of 2RPQs (using only edge labels) modulo very simple TBoxes. The latter is obtained by reduction from another reasoning task (satisfiability of $\mathcal{ALCI}$ TBoxes) and relies on the inner workings of its hardness proof. For completeness, we provide a direct reduction from the acceptance problem for polynomial-space alternating Turing machines (Theorem F.1). The remainder of this section is devoted to the upper bound. We show it by reduction to unrestricted (finite or infinite) satisfiability of C2RPQs modulo a Horn-$\mathcal{ALCIF}$ TBox, which we discuss in Section 6. The principal technique applied in the reduction is *cycle reversing* [18].

Let S be a schema, P a UC2RPQ, and Q an acyclic UC2RPQ. Without loss of generality we may assume that P and Q are Boolean (see Lemma D.1). The key idea is to pass from finite to possibly infinite graphs, thus making canonical witnesses for non-containment easier to find. However, as Example 5.2 shows, we cannot pass freely from finite to possibly infinite graphs, as this may affect the answer.

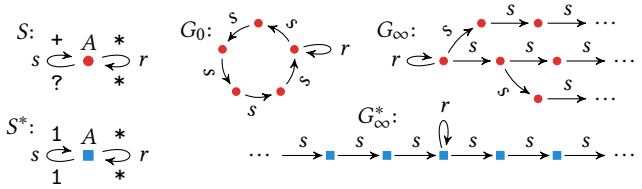

Figure 2: Query containment over finite and infinite graphs.

Example 5.2. Consider the schema S in Figure 2. Observe that S allows infinite graphs that are essentially infinite trees when restricted to s-edges, e.g. G_∞ in Figure 2. In fact, every infinite graph satisfying S that is connected when restricted to s-edges is an infinite tree. On the other hand, every non-empty finite graph that conforms to S is a collection of disjoint cycles when restricted to s-edges, e.g., G_0 in Figure 2. Clearly, the topology of finite and infinite graphs defined by the schema differs drastically.

Now, take the queries $P = \exists x.r(x,x)$, $Q = \exists x,y.(r \cdot s^+ \cdot r)(x,y)$, and observe that $P \subseteq_S Q$. However, the containment does not hold over infinite graphs: P is satisfied by G_∞ while Q is not. □

The reason why we cannot pass directly to infinite models is that finite graphs conforming to schema S may display certain additional common properties, detectable by queries, but not shared by infinite graphs conforming to S. The cycle *reversing technique* [18] allows to capture these properties in S^* such that

$$P \subseteq_S Q \qquad \textit{iff} \qquad P \subseteq^{\infty}_{S^*} Q$$

where by $\subseteq^{\infty}_{S^*}$ we mean containment over possibly infinite graphs conforming to S^*. However, as the following example shows, we cannot obtain S^* by analysing S alone.

Example 5.3. In Example 5.2 we saw that in a finite graph conforming to S, each node has exactly one incoming and one outgoing s-edge. We can use this observation to tighten the original schema S to the schema S^* (Figure 2). Alas, we still have $P \not\subseteq^{\infty}_{S^*} Q$ because there is an infinite graph G^*_∞ that satisfies P but not Q. □

Instead, we first reduce containment modulo schema to finite satisfiability, fusing the schema S and the query Q into a single Horn-$\mathcal{ALCIF}$ TBox, and then pass from finite to unrestricted satisfiability by applying cycle reversing to the resulting TBox. We follow closely the approach of Ibáñez-García et al. [36], relying crucially on some of their results.

Let $\mathcal{T}$ be a Horn-$\mathcal{ALCIF}$ TBox. A *finmod cycle* is a sequence

$$K_1, R_1, K_2, R_2, \ldots, K_{n-1}, R_{n-1}, K_n$$

where $R_1, \ldots, R_{n-1} \in \Sigma^{\pm}$ and $K_1, \ldots, K_n$ are conjunctions of concept names such that $K_n = K_1$ and

$$\mathcal{T} \models K_i \sqsubseteq \exists R_i.K_{i+1} \quad \text{and} \quad \mathcal{T} \models K_{i+1} \sqsubseteq \exists^{\leq 1} R_i^-.K_i$$

for $1 \leq i < n$. By *reversing* the finmod cycle we mean extending $\mathcal{T}$ with concept inclusions

$$K_{i+1} \sqsubseteq \exists R_i^-.K_i \quad \text{and} \quad K_i \sqsubseteq \exists^{\leq 1} R_i.K_{i+1}$$

for $1 \leq i < n$. The *completion* $\mathcal{T}^*$ of a TBox $\mathcal{T}$ is obtained from $\mathcal{T}$ by exhaustively reversing finmod cycles. The following key result is stated in [36] in terms of sets of ground facts (so-called ABoxes) rather than subgraphs, but our formulation is equivalent.

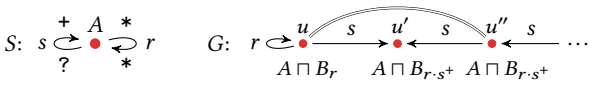

Figure 3: Cycle reversal argument.

Theorem 5.4 (Ibáñez-García et al., 2014)**.** *A Horn-$\mathcal{ALCIF}$ TBox $\mathcal{T}$ has a finite model containing a finite subgraph H iff its completion $\mathcal{T}^*$ has a possibly infinite model containing H.*

Example 5.5. Schema S from Example 5.2 is equivalent to TBox $\mathcal{T}_S$ that consists of

$$\top \sqsubseteq A, \quad A \sqsubseteq \exists s.A, \quad A \sqsubseteq \exists^{\leq 1} s^-.A\,.$$

Non-satisfaction of Q is captured by TBox $\mathcal{T}_{\neg Q}$ that consists of

$$\top \sqsubseteq \forall r.B_r\,, \quad B_r \sqsubseteq \forall s.B_{r\cdot s^+}\,, \quad B_{r\cdot s^+} \sqsubseteq \forall s.B_{r\cdot s^+}\,, \quad B_{r\cdot s^+} \sqsubseteq \forall r.\bot\,.$$

Let $\mathcal{T} = \mathcal{T}_S \cup \mathcal{T}_{\neg Q}$ and observe that $A \sqcap B_{r\cdot s^+}, s, A \sqcap B_{r\cdot s^+}$ is a finmod cycle in $\mathcal{T}$. By reversing it, we obtain

$$A \sqcap B_{r\cdot s^+} \sqsubseteq \exists s^-.A \sqcap B_{r\cdot s^+} \quad \text{and} \quad A \sqcap B_{r\cdot s^+} \sqsubseteq \exists^{\leq 1} s.A \sqcap B_{r\cdot s^+}\,.$$

Now, suppose that there exists a (finite or infinite) model G of $\mathcal{T}^*$ that satisfies P (see Figure 3). G must have a node u with $(u,u) \in r^G$. It follows already from $\mathcal{T}$ that $u \in (A \sqcap B_r)^G$ and that u has an s-successor $u' \in (A \sqcap B_{r\cdot s^+})^G$. The statement $A \sqcap B_{r\cdot s^+} \sqsubseteq \exists s^-.A \sqcap B_{r\cdot s^+}$ in $\mathcal{T}^*$ implies that u' has an s^--successor $u'' \in (A \sqcap B_{r\cdot s^+})^G$. As each node has at most one incoming s-edge, $u = u''$ and $u \in (B_{r\cdot s^+})^G$. But u has an outgoing r-edge, which contradicts the last concept inclusion in $\mathcal{T}_{\neg Q}$. Thus, P is not satisfied in $\mathcal{T}^*$. □

We are now ready to reduce containment modulo schema to unrestricted satisfiability modulo Horn-$\mathcal{ALCIF}$ TBox. Note that the guarantees on the resulting TBox in the statement below are sufficient to conclude Theorem 5.1 using Theorem 6.1.

Theorem 5.6. *Given a UC2RPQ P, an acyclic UC2RPQs Q, and a schema S, one can compute in* EXPTIME *a UC2RPQ $\widehat{P}$ of polynomial size and a Horn-$\mathcal{ALCIF}$ TBox $\mathcal{T}$ using linearly many additional concept names and polynomially many at-most constraints, such that $\widehat{P}$ is (unrestrictedly) satisfiable modulo $\mathcal{T}$ if and only if $P \subseteq_S Q$.*

Let us sketch the proof. Let $\mathcal{T}_S$ be the Horn-$\mathcal{ALCIF}$ TBox corresponding to S. Note that apart from the explicit restrictions captured in $\mathcal{T}_S$ the schema S also ensures that only graphs with *exactly* one label per node are considered. To ensure *at most* one label from Γ_S per node, we use the TBox $\widehat{\mathcal{T}}_S = \mathcal{T}_S \cup \{A \sqcap B \sqsubseteq \bot \mid A, B \in \Gamma_S, A \neq B\}$. The concept inclusion $\top \sqsubseteq \bigsqcup \Gamma_S$, expressing that each node has *at least* one label from Γ_S, is not Horn and cannot be used. Instead, we modify the query P. Assuming $\Gamma_S = \{A_1, A_2, \ldots, A_n\}$, we include $(A_1 + A_2 + \cdots + A_n)$ before and after each edge label used in an atom of P. Additionally, to ensure that P uses only labels allowed by S, we substitute in P each label not in $\Gamma_S \cup \Sigma^{\pm}_S$ by $\varnothing$. Letting $\widehat{P}$ be the resulting query, we have

$$P \subseteq_S Q \qquad \textit{iff} \qquad \widehat{P} \subseteq_{\widehat{\mathcal{T}}_S} Q$$

(see Lemma D.3). Because Q is acyclic, by adapting the rolling-up technique [33] one can compute in PTIME a Horn-$\mathcal{ALCIF}$ TBox $\mathcal{T}_{\neg Q}$ over an extended set of concept names $\Gamma_S \cup \Gamma_Q$ such that

$$\widehat{P} \subseteq_{\widehat{\mathcal{T}}_S} Q \qquad \textit{iff} \qquad \widehat{P} \text{ is finitely satisfiable modulo } \widehat{\mathcal{T}}_S \cup \mathcal{T}_{\neg Q}\,.$$

(see Lemma C.2). Since $\widehat{\mathcal{T}_S} \cup \mathcal{T}_{\neg Q}$ is a Horn-$\mathcal{ALCIF}$ TBox, we can consider its completion $\left(\widehat{\mathcal{T}_S} \cup \mathcal{T}_{\neg Q}\right)^*$. As UC2RPQs are witnessed by finite subgraphs whenever they are satisfied, we can infer from Theorem 5.4 that $\widehat{P}$ is finitely satisfiable modulo $\widehat{\mathcal{T}_S} \cup \mathcal{T}_{\neg Q}$ iff $\widehat{P}$ is satisfiable modulo $\left(\widehat{\mathcal{T}_S} \cup \mathcal{T}_{\neg Q}\right)^*$ (see Lemma D.4).

It remains to compute the completion. Reversing cycles does not introduce new concept names, but it may generate exponentially many concept inclusions. Identifying a finmod cycle involves deciding unrestricted entailment of Horn-$\mathcal{ALCIF}$ concept inclusions, which is decidable in EXPTIME [24]. However, since the input TBox might grow to an exponential size as more and more cycles are reversed, it is unlikely that the completion can be computed in EXPTIME for every Horn-$\mathcal{ALCIF}$ TBox. Our key insight is that $\widehat{\mathcal{T}_S} \cup \mathcal{T}_{\neg Q}$ enjoys a particular property, invariant under reversing cycles, that allows to keep the complexity under control.

A concept inclusion (CI) of the form $K \sqsubseteq \exists R.K'$ or $K \sqsubseteq \exists^{\leq 1} R.K'$ is *relevant* for a TBox $\mathcal{T}$ if the triple (K, R, K') is satisfiable modulo $\mathcal{T}$; that is, some model G of $\mathcal{T}$ contains nodes u and u' such that $u \in K^G$, $(u, u') \in R^G$, and $u' \in (K')^G$. We say that $\mathcal{T}$ is *S-driven* if for each relevant CI in $\mathcal{T}$ of the form $K \sqsubseteq \exists R.K'$ (resp. $K \sqsubseteq \exists^{\leq 1} R.K'$), $\mathcal{T}$ contains $A \sqsubseteq \exists R.A'$ (resp. $A \sqsubseteq \exists^{\leq 1} R.A'$) for some $A, A' \in \Gamma_S$ such that $A \in K$, $A' \in K'$; here and later we blur the distinction between conjunctions of concept names and sets of labels. Note that $\widehat{\mathcal{T}_S} \cup \mathcal{T}_{\neg Q}$ is trivially S-driven, as all its existential and at-most constraints are of the form $A \sqsubseteq \exists R.A'$ or $A \sqsubseteq \exists^{\leq 1} R.A'$.

Lemma 5.7. *Every S-driven TBox $\mathcal{T}$ can be reduced in polynomial time so that it contains at most $|\Sigma_S^{\pm}| \cdot |\Gamma_S|^2$ at-most constraints.*

From our results in Section 6 it follows that unrestricted entailment for a Horn-$\mathcal{ALCIF}$ TBox $\mathcal{T}$ with k concept names and ℓ at-most constraints can be solved in time $O\big(\mathrm{poly}(|\mathcal{T}|) \cdot 2^{\mathrm{poly}(k,\ell)}\big)$ (Corollary E.7). Hence, it would suffice to show that by reversing a finmod cycle in an S-driven TBox, we obtain another S-driven TBox. In fact, we prove something weaker, but sufficient to compute the completion in EXPTIME, and conclude that it is S-driven.

Let $K_1, R_1, \ldots, K_{n-1}, R_{n-1}, K_n$ be a finmod cycle in an S-driven Horn-$\mathcal{ALCIF}$ TBox $\mathcal{T}$. Reversing it will extend $\mathcal{T}$ with CIs

$$K_{i+1} \sqsubseteq \exists R_i^-.K_i \quad \text{and} \quad K_i \sqsubseteq \exists^{\leq 1} R_i.K_{i+1}$$

for $1 \leq i < n$. If all triples (K_i, R_i, K_{i+1}) are unsatisfiable wrt $\mathcal{T}$, then all CIs to be added are irrelevant for $\mathcal{T}$ and we are done. Suppose that some (K_i, R_i, K_{i+1}) is satisfiable. Then, in the model for (K_i, R_i, K_{i+1}) we can trace the finmod cycle forward, witnessing each triple. Hence, the whole cycle is satisfiable (all triples its are). Then, we can show that there are unique $A_1, A_2, \ldots, A_n \in \Gamma_S$ such that $A_i \in K_i$ for all $i \leq n$, and $A_1, R_1, \ldots, A_{n-1}, R_{n-1}, A_n$ is a finmod cycle in $\mathcal{T}$ (Lemma D.6). By reversing it, we can add to $\mathcal{T}$ CIs

$$A_{i+1} \sqsubseteq \exists R_i^-.A_i \quad \text{and} \quad A_i \sqsubseteq \exists^{\leq 1} R_i.A_{i+1}$$

for $1 \leq i < n$, which makes the resulting extension S-driven.

Based on the obtained invariant we can compute the completion $\left(\widehat{\mathcal{T}_S} \cup \mathcal{T}_{\neg Q}\right)^*$ in EXPTIME (Lemma D.7). By reducing $\left(\widehat{\mathcal{T}_S} \cup \mathcal{T}_{\neg Q}\right)^*$ as described above, we obtain the desired TBox.

Finite containment modulo general TBoxes. While the EXPTIME upper bound relies on the special shape of $\widehat{\mathcal{T}_S} \cup \mathcal{T}_{\neg Q}$, the method can be applied directly to general Horn-$\mathcal{ALCIF}$ TBoxes, at the cost of an exponential increase in complexity. Thus, we immediately get that finite containment of UC2RPQs in acyclic UC2RPQs modulo Horn-$\mathcal{ALCIF}$ TBoxes can be solved in 2EXPTIME. To the best of our knowledge this is the first result on finite containment of C2RPQs in the context of description logics. A related problem of finite entailment has been studied for various logics [25–27, 29]. While for conjunctive queries the solutions carry over to finite containment, for CRPQs these logics are too weak.

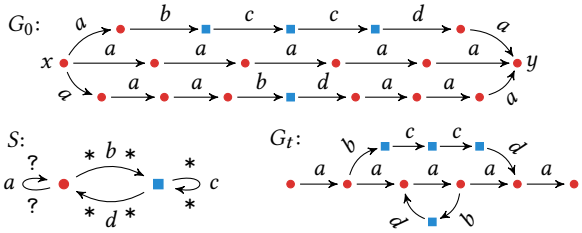

Figure 4: Simple witness for satisfiability.

6 SATISFIABILITY MODULO TBOX

The last missing piece is to solve the unrestricted satisfiability of C2RPQs modulo Horn-$\mathcal{ALCIF}$. Calvanese et al. show that the problem is in EXPTIME not only for Horn-$\mathcal{ALCIF}$, but even for $\mathcal{ALCIF}$ extended with additional features [14]. This result is not directly applicable, because our reduction produces a TBox of exponential size. The following theorem gives the more precise complexity bounds we need.

Theorem 6.1. *Unrestricted satisfiability of a C2RPQ p modulo an $\mathcal{ALCIF}$ TBox $\mathcal{T}$ using k concept names and ℓ at-most constraints can be decided in time $O\big(\mathrm{poly}(|\mathcal{T}|) \cdot 2^{\mathrm{poly}(|p|,k,\ell)}\big)$.*

Calvanese et al. solve the problem by first showing a simple model property and then providing an algorithm testing existence of simple models. We rely on the same simple model property, but design a new algorithm with the desired complexity bounds. Yet, rather than diving into the details of the algorithm, we devote most of this section to the simple model property. We do it to show a connection to an elegant graph-theoretical notion that allows to simplify the reasoning considerably, at least for $\mathcal{ALCIF}$. We begin by illustrating how simple models are obtained for queries satisfiable modulo schemas (rather than arbitrary TBoxes).

Example 6.2. Take the schema S in Figure 4 (its two types are represented with a blue square and a red circle), and consider the following satisfiable (cyclic) query

$$p(x, y) = (a \cdot b \cdot c^+ \cdot d \cdot a)(x, y) \wedge (a^*)(x, y) \wedge (a^* \cdot b \cdot d \cdot a^*)(x, y)\,.$$

Since p is satisfiable modulo S, we take any graph conforming to S where p is satisfied, and we choose any 3 paths witnessing each of the regular expressions of p. We construct the initial graph G_0 consisting of the 3 paths joined at their ends: it might look like the one in Figure 4. We observe that S requires every red circle node to have at most one outgoing a-edge and at most one incoming a-edge (to and from a red circle node). The initial graph G_0 violates this requirement and to enforce it we exhaustively merge offending nodes. The final graph G_t is a simple model of p modulo S. □

We formalise simple models using a graph-theoretic notion of sparsity proposed by Lee and Streinu [41]. We say that a connected graph G with n nodes and m edges is *c-sparse* if $m \leq n + c$. (In Lee and Streinu's terminology this corresponds to $(1, -c)$-sparsity.) Being c-sparse is preserved under adding and removing nodes of degree 1. By exhaustively removing nodes of degree 1 from a c-sparse graph G we arrive at single node or a connected c-sparse graph H in which all nodes have degree at least 2. Assuming $c \geq 1$, it is not hard to see that such a graph consists of at most $k = 2c$ distinguished nodes connected by at most $l = 3c$ simple paths disjoint modulo endpoints (see Lemma E.1). We call such a graph a *(k, l)-skeleton*, and we refer to the graph H above as the *skeleton of* G. Thus, a c-sparse graph consists of a $(2c, 3c)$-skeleton and a number of attached trees; by attaching a tree to a graph we mean taking their disjoint union and adding a single edge between the root of the tree and some node of the graph.

For the purpose of the simple model property we need to lift the notion of c-sparsity to infinite graphs. We call a (possibly infinite) graph *c-sparse* if it consists of a finite connected c-sparse graph with finitely many finitely branching trees attached.

Theorem 6.3. *A connected C2RPQ p is satisfiable in a possibly infinite model of an $\mathcal{ALCIF}$ TBox $\mathcal{T}$ iff p is satisfiable in a possibly infinite $|p|$-sparse model of $\mathcal{T}$.*

Proof. Let c be the difference between the number of atoms and the number of variables of p. Because p is connected, $c \geq -1$. By definition, p understood as a graph with variables as nodes and atoms as edges is c-sparse.

We write $H \to H'$ to indicate that there is a *homomorphism* from graph H to graph H'; that is, a function h mapping nodes of H to nodes of H' that preserves node labels and the existence of labelled edges between pairs of nodes. Let G be a (possibly infinite) model of p and $\mathcal{T}$. We construct a sequence of finite connected c-sparse graphs of strictly decreasing size

$$G_0 \to G_1 \to \cdots \to G_t \to G$$

such that $G_0 \models P$ and the homomorphism from G_t to G is injective over R-successors of every node, for each R.

To construct G_0 let us fix a match of p in G together with a (finite) witnessing path for each atom of p. We construct G_0 as follows. For each variable x of p we include a node v_x whose set of labels is identical to that of the image of x in G under the fixed match. Next, for each atom of p that connects variables x and y we add a simple path connecting x and y such that the sequence of edge labels and sets of node labels read off of this path is identical to that of the witnessing path of this atom in G. This graph can be seen as a specialization of p where each regular expression is replaced by a single concrete word, except that we include full sets of labels of nodes, as they are encountered in the witnessing path in G. It follows immediately that $G_0 \models p$ and that $G_0 \to G$. To see that G_0 is c-sparse one can eliminate the internal nodes of the connecting paths one by one, like in the proof of Lemma E.1, until a graph isomorphic to p remains.

We define the remaining graphs G_i inductively, maintaining an additional invariant $G_i \to G$. Suppose we already have G_i together with a homomorphism $h_i \colon G_i \to G$ for some $i \geq 0$. If h_i is injective over R-successors of each node of G_i, we are done. If not, there are two different R-successors u_1 and u_2 of a node v in G_i that are mapped to the same node u' in G. It follows that u_1 and u_2 have the same sets of labels types. We let G_{i+1} be the graph obtained from G_i by merging u_1 and u_2 into a single node u. We include an R'-edge between u and each R'-successor of u_1 or u_2. This decreases the number of nodes by one, and the number or edges by at least one. It follows that G_{i+1} is c-sparse and $G_i \to G_{i+1} \to G$.

Because the sizes of graphs G_i are strictly decreasing, at some point we will arrive at a graph G_t such that the homomorphism from G_t to G is injective over R-successors.

The graph G_t clearly satisfies p. It also satisfies all concept inclusions in $\mathcal{T}$ of the forms $K \sqsubseteq A_1 \sqcup A_2 \sqcup \cdots \sqcup A_n$, $K \sqsubseteq \bot$, $K \sqsubseteq \forall R.K'$, $K \sqsubseteq \nexists R.K'$, and $K \sqsubseteq \exists^{\leq 1} R.K'$, because h_i is injective over R-successors and $G \models \mathcal{T}$. On the other hand, G_t is not guaranteed to satisfy concept inclusions of the form $K \sqsubseteq \exists R.K'$ in $\mathcal{T}$. In order to fix it, we exhaustively (ad infinitum) perform the following: whenever a node v in G_t is missing an R-successor with some set of labels, we add it and map it to some such R-successor u' of the image of v in G (u' exists because $G \models \mathcal{T}$). As $c \leq |p|$, the resulting (typically infinite) graph $\widehat{G}$ is $|p|$-sparse, and it satisfies p and $\mathcal{T}$. □

The connectedness assumption in Theorem 6.3 is not restrictive, because a witnessing graph for p can be obtained by taking the disjoint union of witnesses for its connected components. Hence, it remains to decide for a given connected p if there exists a $|p|$-sparse graph G that satisfies p and $\mathcal{T}$. To get a finer control of the effect different parameters of the input have on the complexity, we side-step two-way alternating tree automata (2ATA) applied by Calvanese et al. and develop a more direct algorithm. Observe that if p is satisfied in a $|p|$-sparse graph G, then G contains a $(4|p|, 5|p|)$-skeleton H', extending the skeleton of G, such that all variables of p are mapped to distinguished nodes of H'. Thus, the algorithm can guess a $(4|p|, 5|p|)$-skeleton H' with each path represented by a single symbolic edge and check that it can be completed to a suitable graph G by materializing symbolic edges into paths and attaching finitely many finitely branching trees in such a way that G is a model of $\mathcal{T}$ and there is a match of p in G that maps variables of p to distinguished nodes of H'. This can be done within the required time bounds by means of a procedure that can be seen as a variant of type elimination or an emptiness test for an implicitly represented nondeterministic tree automaton (see Theorem E.3).

7 CONCLUSIONS AND FUTURE WORK

In this paper we have studied several static analysis problems for graph transformations defined with Datalog-like rules that use acyclic C2RPQs. When the source schema is given, we studied the *equivalence* problem of two given transformations, and the problem of *target schema elicitation* for a given transformation. If the output schema is also given, we have studied the problem of *type checking*. We have shown that the above problems can be reduced to containment of C2RPQs in acyclic UC2RPQs modulo schema, a problem that we have reduced to the unrestricted (finite or infinite) satisfiabilty of a C2RPQ modulo Horn-$\mathcal{ALCIF}$ TBox using cycle reversing. For the latter problem we have presented an algorithm

with sufficiently good complexity to accommodate the exponential blow-up introduced by cycle reversing, thus allowing to solve in EXPTIME all problems of interest.

We have also shown matching lower bounds by reducing query containment modulo schema to each of the static analysis problems. Because containment of CRPQs alone (with no schema) is known to be EXPSPACE-complete [13] and containment of RPQs modulo disjunctive dependencies is known to be undecidable [21], it is unlikely (or even impossible) that our results can be extended to transformations with C2RPQs or schemas with disjunctions.

However, a number of extension of the presented results can be envisioned as future work. First, we believe that our methods can be extended to more expressive classes of queries used in transformation rules. It is straightforward to extend our methods to two-way *nested regular expressions* (NREs) [49] and we also intend to investigate introducing *negation* in filter expressions of NREs. It is straightforward to encode data values in our graph model, for instance, by using dedicated node labels to designate *literal* nodes whose identifiers are their data values. Then, one can apply methods similar to type checking to verify that transformations are well-behaved, and in particular, do not attempt to construct literal nodes from non-literal ones. However, the full consequences of allowing literal values in definitions of transformation rules need to be thoroughly investigated. Finally, we have considered equivalence of transformations based on equality of results but one could also consider a variant based on isomorphism of results.

REFERENCES

[1] Serge Abiteboul and Richard Hull. 1987. IFO: A Formal Semantic Database Model. *ACM Trans. Database Syst.* 12, 4 (Nov. 1987), 525–565. https://doi.org/10.1145/32204.32205 Place: New York, NY, USA Publisher: Association for Computing Machinery.

[2] Serge Abiteboul and Paris C. Kanellakis. 1998. Object Identity as a Query Language Primitive. *J. ACM* 45, 5 (Sept. 1998), 798–842. https://doi.org/10.1145/290179.290182 Place: New York, NY, USA Publisher: Association for Computing Machinery.

[3] Renzo Angles. 2018. The Property Graph Database Model. In *Proceedings of the 12th Alberto Mendelzon International Workshop on Foundations of Data Management, Cali, Colombia, May 21-25, 2018 (CEUR Workshop Proceedings, Vol. 2100)*, Dan Olteanu and Barbara Poblete (Eds.). CEUR-WS.org. http://ceur-ws.org/Vol-2100/paper26.pdf

[4] Renzo Angles and Claudio Gutierrez. 2008. Survey of graph database models. *Comput. Surveys* 40, 1 (Feb. 2008), 1–39. https://doi.org/10.1145/1322432.1322433

[5] Marcelo Arenas, Jorge Pérez, Juan Reutter, and Cristian Riveros. 2013. The language of plain SO-tgds: Composition, inversion and structural properties. *J. Comput. System Sci.* 79 (2013). JCSS Foundations of Data Management.

[6] Franz Baader, Ian Horrocks, Carsten Lutz, and Ulrike Sattler. 2017. *An Introduction to Description Logic.* Cambridge University Press.

[7] Pablo Barceló, Jorge Pérez, and Juan L. Reutter. 2013. Schema mappings and data exchange for graph databases. In *Joint 2013 EDBT/ICDT Conferences, ICDT '13 Proceedings, Genoa, Italy, March 18-22, 2013.* ACM.

[8] Mikolaj Bojanczyk, Leszek Aleksander Kolodziejczyk, and Filip Murlak. 2013. Solutions in XML data exchange. *J. Comput. Syst. Sci.* 79 (2013).

[9] Iovka Boneva, Slawek Staworko, and Jose Lozano. 2020. Consistency and Certain Answers in Relational to RDF Data Exchange with Shape Constraints *(Communications in Computer and Information Science, Vol. 1259)*. Springer.

[10] Angela Bonifati, Peter Furniss, Alastair Green, Russ Harmer, Eugenia Oshurko, and Hannes Voigt. 2019. Schema Validation and Evolution for Graph Databases. In *Conceptual Modeling (ER)*, Alberto H. F. Laender, Barbara Pernici, Ee-Peng Lim, and José Palazzo M. de Oliveira (Eds.). Springer International Publishing, 448–456.

[11] Peter Buneman, Mary Fernandez, and Dan Suciu. 2000. UnQL: A Query Language and Algebra for Semistructured Data Based on Structural Recursion. *The VLDB Journal* 9 (2000).

[12] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y Vardi. 2011. Simplifying schema mappings. In *Proceedings of the 14th International Conference on Database Theory.*

[13] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. 2000. Containment of Conjunctive Regular Path Queries with Inverse. In *KR 2000, Principles of Knowledge Representation and Reasoning Proceedings of the Seventh International Conference, Breckenridge, Colorado, USA, April 11-15, 2000*, Anthony G. Cohn, Fausto Giunchiglia, and Bart Selman (Eds.). Morgan Kaufmann, 176–185.

[14] Diego Calvanese, Magdalena Ortiz, and Mantas Simkus. 2011. Containment of Regular Path Queries under Description Logic Constraints. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, Toby Walsh (Ed.). IJCAI/AAAI, 805–812. https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-141

[15] Peter P. Chen. 1975. The Entity-Relationship Model: Toward a Unified View of Data. In *Proceedings of the International Conference on Very Large Data Bases, September 22-24, 1975, Framingham, Massachusetts, USA*, Douglas S. Kerr (Ed.). ACM, 173. https://doi.org/10.1145/1282480.1282492

[16] Cristina Civili, Jose Mora, Riccardo Rosati, Marco Ruzzi, and Valerio Santarelli. 2016. Semantic Analysis of R2RML Mappings for Ontology-Based Data Access. In *Web Reasoning and Rule Systems*, Magdalena Ortiz and Stefan Schlobach (Eds.). Springer International Publishing, Cham, 25–38.

[17] J. Corman, J. L. Reutter, and O. Savkovic. 2018. Semantics and Validation of Recursive SHACL. In *International Semantic Web Conference (ISWC)*. 318–336.

[18] Stavros S. Cosmadakis, Paris C. Kanellakis, and Moshe Y. Vardi. 1990. Polynomial-Time Implication Problems for Unary Inclusion Dependencies. *J. ACM* 37, 1 (1990), 15–46.

[19] Bruno Courcelle. 1994. Monadic second-order definable graph transductions: a survey. *Theoretical Computer Science* 126 (1994).

[20] Souripriya Das, Seema Sundara, and Richard Cyganiak. 2011. R2RML: RDB to RDF Mapping Language. http://www.w3.org/TR/r2rml/.

[21] Alin Deutsch and Val Tannen. 2002. Optimization Properties for Classes of Conjunctive Regular Path Queries. In *Database Programming Languages*, Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Giorgio Ghelli, and Gösta Grahne (Eds.). Vol. 2397. Springer Berlin Heidelberg, Berlin, Heidelberg, 21–39. https://doi.org/10.1007/3-540-46093-4_2 Series Title: Lecture Notes in Computer Science.

[22] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. 2005. Data exchange: semantics and query answering. *Theor. Comput. Sci.* 336 (2005).

[23] Ingo Feinerer, Reinhard Pichler, Emanuel Sallinger, and Vadim Savenkov. 2015. On the undecidability of the equivalence of second-order tuple generating dependencies. *Information Systems* 48 (2015).

[24] Giuseppe De Giacomo and Maurizio Lenzerini. 1996. TBox and ABox Reasoning in Expressive Description Logics. In *KR*. Morgan Kaufmann, 316–327.

[25] Tomasz Gogacz, Víctor Gutiérrez-Basulto, Albert Gutowski, Yazmín Ibáñez-García, and Filip Murlak. 2020. On Finite Entailment of Non-Local Queries in Description Logics. In *KR*. 424–433.

[26] Tomasz Gogacz, Víctor Gutiérrez-Basulto, Yazmín Ibáñez-García, Jean Christoph Jung, and Filip Murlak. 2019. On Finite and Unrestricted Query Entailment beyond SQ with Number Restrictions on Transitive Roles. In *IJCAI*. ijcai.org, 1719–1725.

[27] Tomasz Gogacz, Yazmín Angélica Ibáñez-García, and Filip Murlak. 2018. Finite Query Answering in Expressive Description Logics with Transitive Roles. In *KR*. AAAI Press, 369–378.

[28] W3C: RDF Working Group. 2004. Resource Description Framework. https://www.w3.org/RDF/. Accessed: 2022, June 03.

[29] Victor Gutiérrez-Basulto, Albert Gutowski, Yazmin Ibáñez-Garcia, and Filip Murlak. 2022. Finite Entailment of UCRPQs over ALC Ontologies. https://doi.org/10.48550/ARXIV.2204.14261

[30] M. Gyssens, J. Paredaens, J. van den Bussche, and D. van Gucht. 1994. A graph-oriented object database model. *IEEE Transactions on Knowledge and Data Engineering* 6, 4 (1994), 572–586. https://doi.org/10.1109/69.298174

[31] Jan Hidders. 2003. Typing Graph-Manipulation Operations. In *ICDT (Lecture Notes in Computer Science, Vol. 2572)*. Springer, 391–406.

[32] HL7.org. 2019. FHIR Mapping Language. https://hl7.org/fhir/mapping-language.html. Accessed: 2022-05-25.

[33] Ian Horrocks and Sergio Tessaris. 2000. A Conjunctive Query Language for Description Logic Aboxes. In *AAAI/IAAI*. AAAI Press / The MIT Press, 399–404.

[34] Richard Hull and Masatoshi Yoshikawa. 1990. ILOG: Declarative Creation and Manipulation of Object Identifiers. In *16th International Conference on Very Large Data Bases, August 13-16, 1990, Brisbane, Queensland, Australia, Proceedings*, Dennis McLeod, Ron Sacks-Davis, and Hans-Jörg Schek (Eds.). Morgan Kaufmann, 455–468. http://www.vldb.org/conf/1990/P455.PDF

[35] Richard Hull and Masatoshi Yoshikawa. 1991. On the Equivalence of Database Restructurings Involving Object Identifiers (Extended Abstract). In *Proceedings of the Tenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (Denver, Colorado, USA) *(PODS '91)*. Association for Computing Machinery, New York, NY, USA, 328–340. https://doi.org/10.1145/113413.113443

[36] Yazmín Angélica Ibáñez-García, Carsten Lutz, and Thomas Schneider. 2014. Finite Model Reasoning in Horn Description Logics. In *KR*. AAAI Press, 288–297.
[37] Kazuhiro Inaba, Soichiro Hidaka, Zhenjiang Hu, Hiroyuki Kato, and Keisuke Nakano. 2011. Graph-Transformation Verification Using Monadic Second-Order Logic. In *Proceedings of the 13th International ACM SIGPLAN Symposium on Principles and Practices of Declarative Programming (PPDP '11)*. Association for Computing Machinery, New York, NY, USA, 17–28. https://doi.org/10.1145/2003476.2003482 event-place: Odense, Denmark.
[38] Michael Kifer and Georg Lausen. 1989. F-Logic: A Higher-Order Language for Reasoning about Objects, Inheritance, and Scheme. *SIGMOD Rec.* 18, 2 (June 1989), 134–146. https://doi.org/10.1145/66926.66939 Place: New York, NY, USA Publisher: Association for Computing Machinery.
[39] Phokion G. Kolaitis, Reinhard Pichler, Emanuel Sallinger, and Vadim Savenkov. 2020. On the Language of Nested Tuple Generating Dependencies. *ACM Trans. Database Syst.* 45, 2 (2020), 8:1–8:59.
[40] Gabriel M. Kuper and Moshe Y. Vardi. 1993. The Logical Data Model. *ACM Trans. Database Syst.* 18, 3 (Sept. 1993), 379–413. https://doi.org/10.1145/155271.155274
[41] Audrey Lee and Ileana Streinu. 2008. Pebble game algorithms and sparse graphs. *Discret. Math.* 308, 8 (2008), 1425–1437.
[42] M. Levene and A. Poulovassilis. 1990. The hypernode model and its associated query language. In *Proceedings of the 5th Jerusalem Conference on Information Technology, 1990. 'Next Decade in Information Technology'*. IEEE Comput. Soc, Jerusalem, Israel, 520–530. https://doi.org/10.1109/JCIT.1990.128324
[43] David Maier. 1986. A Logic for Objects. In *Proceedings of the Workshop on Foundations of Deductive Databases and Logic Programming*. Washington D.C., 6 – 26.
[44] Sebastian Maneth, Alexandru Berlea, Thomas Perst, and Helmut Seidl. 2005. XML type checking with macro tree transducers. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*.
[45] Wim Martens and Frank Neven. 2007. Frontiers of tractability for typechecking simple XML transformations. *J. Comput. System Sci.* 73 (2007).
[46] Wim Martens, Frank Neven, and Marc Gyssens. 2008. Typechecking top-down XML transformations: Fixed input or output schemas. *Information and Computation* 206, 7 (2008), 806–827. https://doi.org/10.1016/j.ic.2008.01.002
[47] Tova Milo, Dan Suciu, and Victor Vianu. 2003. Typechecking for XML transformers. *J. Comput. System Sci.* 66 (2003).
[48] J. Paredaens, P. Peelman, and L. Tanca. 1995. G-Log: a graph-based query language. *IEEE Transactions on Knowledge and Data Engineering* 7, 3 (June 1995), 436–453. https://doi.org/10.1109/69.390249
[49] Jorge Pérez, Marcelo Arenas, and Claudio Gutiérrez. 2010. nSPARQL: A navigational language for RDF. *J. Web Semant.* 8, 4 (2010), 255–270. https://doi.org/10.1016/j.websem.2010.01.002
[50] Eric Prud'hommeaux, Harold R. Solbrig, and Guoqian Jiang. 2017. ShEx, RDF and FHIR. In *Summit on Clinical Research Informatics, CRI 2017, San Francisco, CA, USA, March 27-30, 2017*. AMIA. http://knowledge.amia.org/amia-64484-cri2017-1.3520710/t003-1.3521532/t003-1.3521533/a081-1.3521573/a082-1.3521570
[51] Grzegorz Rozenberg (Ed.). 1997. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific.
[52] Juan F. Sequeda. 2013. On the Semantics of R2RML and its Relationship with the Direct Mapping. In *Proceedings of the ISWC 2013 Posters & Demonstrations Track, Sydney, Australia, October 23, 2013 (CEUR Workshop Proceedings, Vol. 1035)*, Eva Blomqvist and Tudor Groza (Eds.). CEUR-WS.org, 193–196.
[53] S. Staworko, I. Boneva, J. E. Labra Gayo, S. Hym, E. G. Prud'hommeaux, and H. Solbrig. 2015. Complexity and Expressiveness of ShEx for RDF. In *International Conference on Database Theory (ICDT)*. 195–211.

A DETAILS ON QUERIES

A *two-way regular expression* is an expression defined with the following grammar.

$$\varphi ::= \varnothing \mid \epsilon \mid A \mid R \mid \varphi \cdot \varphi \mid \varphi + \varphi \mid \varphi^*,$$

where $A \in \Gamma$ and $R \in \Sigma^{\pm}$. We define the semantics with the notion of witnessing paths that we formalize next. Given a graph G, a *path* from u_0 to u_k in G is a sequence $\pi = u_0 \cdot \ell_1 \cdot u_1 \cdot \ldots \cdot u_{k-1} \cdot \ell_k \cdot u_k$ such that $u_0, \ldots, u_k$ are nodes of G, $\ell_1, \ldots, \ell_k \in \Gamma \cup \Sigma^{\pm}$, and for every $i \in \{1, \ldots, k\}$ the following conditions are satisfied:

(1) if $\ell_i \in \Gamma$, then $u_{i-1} = u_i$ and $u_i \in \ell_i^G$,
(2) if $\ell_i \in \Sigma^{\pm}$, then $(u_{i-1}, u_i) \in \ell_i^G$.

The labeling of π is $\ell_1 \cdot \ldots \cdot \ell_n$. Given a two-way regular expression φ we define the corresponding binary relation on nodes of the graph: $(u, v) \in [\varphi]_G$ iff there is a path from node u to node v in G whose labeling is recognized by φ.

Now, a *conjunctive two-way regular path query* (C2RPQ) is a formula of the form

$$q(\bar{x}) = \exists \bar{y}. \varphi_1(z_1, z_1') \wedge \ldots \wedge \varphi_k(z_k, z_k'),$$

where for every $i \in \{1, \ldots, k\}$ the formula φ_i is a two-way regular expression and $\bar{x} = \{z_1, z_1', \ldots, z_k, z_k'\} \setminus \bar{y}$. A C2RPQ is *Boolean* if all of its variables are existentially quantified.

Evaluating a C2RPQ $q(\bar{x})$ over a graph G yields a set $[q(\bar{x})]^G$ of tuples over $\bar{x}$ i.e., functions that assign nodes of G to elements of $\bar{x}$. Formally, $t \in [q(\bar{x})]^G$ iff there is a tuple t' over $\bar{y}$ such that the two tuples combined $t'' = t \cup t'$ satisfy all atoms i.e., $(t''(z_i), t''(z_i')) \in [\varphi_i]_G$ for every $i \in \{1, \ldots, k\}$. When the query is Boolean, then it may have only a single answer, the *empty tuple* () i.e., the unique function with the empty domain. If indeed $() \in [q]^G$ we say that q is satisfied in G and denote it by $G \models q$; otherwise, when $[q]^G = \emptyset$, we say that q is not satisfied in G and we write $G \not\models q$.

For defining transformations we employ the subclass of acyclic C2RPQs. Formally, for a query q we construct its *query multigraph* whose nodes are variables and for every atom $\varphi(x, y)$ we add an edge (x, y) unless the atom is of the form $A(x, x)$, $\epsilon(x, x)$, or $\varnothing(x, x)$. q is *acyclic* if its query multigraph is acyclic.

Finally, the semantics of *unions of conjunctive two-way regular path queries* (UC2RPQs), represented as sets of C2RPQs, is defined simply as:

$$[\{Q_1(\bar{x}), \ldots, Q_k(\bar{x})\}]^G = [Q_1(\bar{x})]^G \cup \ldots \cup [Q_k(\bar{x})]^G.$$

A UC2RPQ is *acyclic* if all of its components are acyclic. A *Boolean* UC2RPQ consists of Boolean C2RPQs.

B PROOFS FOR TRANSFORMATIONS

We begin by introducing elements of useful terminology. Given any finite subsets $\Gamma_0 \subseteq \Gamma$ and $\Sigma_0 \subseteq \Sigma$, we say that a schema S is over Γ_0 and Σ_0 if $\Gamma_S = \Gamma_0$ and $\Sigma_S = \Sigma_0$. Analogously, we say that a $\mathcal{ALCIF}$ TBox $\mathcal{T}$ is over Γ_0 and Σ_0 if all base concept names and base rule names used in $\mathcal{T}$ are from Γ_0 and Σ_0 respectively. Also, we say that a graph G is over Γ_0 and Σ_0 if G does not use any node or edge label outside of Γ_0 and Σ_0, and we extend this notion to families of graphs in the canonical fashion: $\mathcal{G}$ is a family of graphs over Γ_0 and Σ_0 if every graph in G is over Γ_0 and Σ_0. Finally, a transformation T is over Γ_0 and Σ_0 if all rules in T use in their heads node and edge labels in Γ_0 and Σ_0 respectively.

However, for a transformations we shall need to identify tighter sets of node and edge labels when the input schema is known. As such, a transformation rule $\rho \leftarrow q(\bar{x})$ is *productive* modulo a schema S if $q(\bar{x}) \not\subseteq_S \varnothing$. A transformation T is *trimmed* modulo S if 1) every rule in T is productive modulo S, 2) for every $A \in \Gamma_T$ there is an A-node rule in T, and 3) for every $r \in \Sigma_T$ there is a r-edge rule in T. Naturally, checking that a transformation is trimmed can be Turing-reduced in polynomial time to testing query containment modulo schema. Moreover, for a given schema S we can trim a given transformation T by removing all unproductive rules and removing from Γ_T and Σ_T any symbols that are not present in the head of any of the remaining rules.

Next, an $\mathcal{L}_0$ TBox over Γ_0 and Σ_0 is a set of statements of the forms

$$A \sqsubseteq \exists R.B, \qquad A \sqsubseteq \nexists R.B, \qquad A \sqsubseteq \exists^{\leq 1} R.B,$$

where $A, B \in \Gamma_0$ and $R \in \Sigma_0^{\pm}$. $\mathcal{T}$ is *coherent* iff 1) $\mathcal{T}$ does not contains two contradictory rules $A \sqsubseteq \exists R.B$ and $A \sqsubseteq \nexists R.B$ for any $A, B \in \Gamma$ and $R \in \Sigma^{\pm}$, and 2) $\mathcal{T}$ contains $A \sqsubseteq \exists^{\leq 1} R.B$ whenever it contains $A \sqsubseteq \nexists R.B$. Now, for a given schema S the corresponding $\mathcal{L}_0$ TBox $\mathcal{T}_S$ (over Γ_S and Σ_S) is defined as follows.

$$\begin{aligned}\mathcal{T}_S = &\{A \sqsubseteq \exists R.B \mid A, B \in \Gamma_S,\ R \in \Sigma_S^{\pm},\ \delta_S(A,R,B) \in \{1, +\}\}\\ &\cup \{A \sqsubseteq \exists^{\leq 1} R.B \mid A, B \in \Gamma_S,\ R \in \Sigma_S^{\pm},\ \delta_S(A,R,B) \in \{1, ?, 0\}\}\\ &\cup \{A \sqsubseteq \nexists R.B \mid A, B \in \Gamma_S,\ R \in \Sigma_S^{\pm},\ \delta_S(A,R,B) = 0\}.\end{aligned}$$

It is easy to see that there is one-to-one correspondence between schemas and coherent TBoxes. More precisely, given $\Gamma_0 \subseteq \Gamma$ and $\Sigma_0 \subseteq \Sigma$, for any schema S over Γ_0 and Σ_0, $\mathcal{T}_S$ is a coherent TBox over Γ_S and Σ_S, and for any coherent TBox $\mathcal{T}$ over Γ_0 and Σ_0 there is a unique schema S over Γ_0 and Σ_0 such that $\mathcal{T}_S = \mathcal{T}$. Naturally, $\mathcal{T}_S$ also captures the semantics of the cardinality constraints of S.

Proposition B.1. *For any schema S and for any graph G, G conforms to S if and only if $G \models \mathcal{T}_S$, $G \models \top \sqsubseteq \bigsqcup \Gamma_S$, and $G \models A \sqcap B \sqsubseteq \bot$ for any $A, B \in \Gamma_S$.*

Proof. Straightforward since the $\mathcal{ALCIF}$ formulas are translations of the conditions of conformance of a graph to a schema. □

We use the above result to reduce type checking to testing entailment of simple $\mathcal{ALCIF}$ statements. Recall that that for a schema S and a transformation T we define the entailment relation $(T, S) \models K \sqsubseteq K'$ as $T(G) \models K \sqsubseteq K'$ for every $G \in L(S)$.

Lemma B.2. *Given two schemas S and S' and a transformation T, $\{T(G) \mid G \in L(S)\} \subseteq L(S')$ if and only if $(T, S) \models \top \sqsubseteq \bigsqcup \Gamma_T$ and $(T, S) \models \mathcal{T}_{S'}$.*

Proof. Immediate consequence of Proposition B.1 and the fact that transformations must use a single dedicated node constructor for each node label. This ensures that $(T, S) \models A \sqcap B \sqsubseteq \bot$ holds for any $A, B \in \Gamma_{S'}$. □

Later we prove how to reduce entailment of statements to query containment. Before, we address the problem of schema elicitation by observing that the correspondence between schemas and their $\mathcal{L}_0$ TBoxes is tighter. We first need to establish two auxiliary results. The first one characterizes the containment of schemas, which is expressed as an extension of a syntactic containment relation $\preccurlyeq$ on the symbols used to specify participation constraints. More precisely, we define $\preccurlyeq$ as the transitive and reflexive closure of the following assertions: $0 \preccurlyeq ?$, $1 \preccurlyeq ?$, $? \preccurlyeq +$, and $+ \preccurlyeq *$.

Proposition B.3. *Take finite $\Gamma_0 \subseteq \Gamma$ and $\Sigma_0 \subseteq \Sigma$. Given two schemas S_1 and S_2 over Γ_0 and Σ_0, $L(S_1) \subseteq L(S_2)$ if and only if*

$$\delta_{S_1}(A, R, B) \preccurlyeq \delta_{S_2}(A, R, B)$$

for every $A, B \in \Gamma_0$ and $R \in \Sigma_0^{\pm}$.

Proof. For the *if* part, we take any G that conforms to S_1 and we note first that every node of G has exactly one label in Γ_0. Also, for any $A, B, \in \Gamma_0$ and any $R \in \Sigma_0^{\pm}$ we observe that

$$\delta_{S_1}(A, R, B) \preccurlyeq \delta_{S_2}(A, R, B)$$

implies that any A-node in G whose number of R-successors with label B satisfies the participation constraint $\delta_{S_1}(A, R, B)$ will also satisfy $\delta_{S_2}(A, R, B)$. □

Next, we establish correspondence between $\mathcal{L}_0$ theories of sets of graphs and their containment-minimal schemas.

Proposition B.4. *Take finite $\Gamma_0 \subseteq \Gamma$ and $\Sigma_0 \subseteq \Sigma$ and take any nonempty family $\mathcal{G}$ of graphs over Γ_0 and Σ_0 such that $\mathcal{G} \models \top \sqsubseteq \bigsqcup \Gamma_0$ and $\mathcal{G} \models A \sqcap B \sqsubseteq \bot$ for all $A, B \in \Gamma_0$. Let $\mathcal{T}$ be the set of all $\mathcal{L}_0$ statements over Γ_0 and Σ_0 that hold in every graph in $\mathcal{G}$. Then, $\mathcal{T}$ corresponds to the containment minimal schema S over Γ_0 and Σ_0 such that $\mathcal{G} \subseteq L(S)$.*

Proof. We first argue that $\mathcal{T}$ is coherent. Indeed, should $\mathcal{T}$ contain two contradictory statements $A \sqsubseteq \exists R.B$ and $A \sqsubseteq \nexists R.B$, then no graph in $\mathcal{G}$ could satisfy $\mathcal{T}$ and we know that $\mathcal{G}$ is nonempty. Consequently, $\mathcal{T}$ corresponds to a schema that we denote $S^{\circ} = (\Gamma_0, \Sigma_0, \delta_{S^{\circ}})$. Naturally, $\mathcal{G} \subseteq L(S^{\circ})$ because $\mathcal{G} \models \top \sqsubseteq \bigsqcup \Gamma_0$ and $\mathcal{G} \models A \sqcap B \sqsubseteq \bot$.

Now, take any schema S over Γ_0 and Σ_0 such that $\mathcal{G} \subseteq L(S)$. We show that $L(S^{\circ}) \subseteq L(S)$ with a proof by contradiction. Suppose $L(S^{\circ}) \not\subseteq L(S)$. By Proposition B.3, there are $A, B \in \Gamma_0$ and $R \in \Sigma_0^{\pm}$ such that $\delta_{S^{\circ}}(A, R, B) \not\preccurlyeq \delta_S(A, R, B)$. This means that $\mathcal{T}_S$ contains an (A, R, B)-constraint ϕ that $\mathcal{T}_{S^{\circ}}$ does not (by (A, R, B)-constraints we mean $A \sqsubseteq \exists R.B$, $A \sqsubseteq \exists^{\leq 1} R.B$, and $A \sqsubseteq \nexists R.B$). Since $\phi \notin \mathcal{T}_{S^{\circ}}$ there is a graph $G \in \mathcal{G}$ such that $H \not\models \phi$, and consequently, G does not conform to S. Thus $\mathcal{G} \not\subseteq L(S)$, a contradiction. □

We obtain the following result allowing to solve the problem of schema elicitation problem.

Lemma B.5. *Take a schema S and a transformation T that is trimmed modulo S and such that $(T, S) \models \top \sqsubseteq \bigsqcup \Gamma_T$. Let $\mathcal{T}$ be the set of all $\mathcal{L}_0$ statements over Γ_T and Σ_T that are satisfied by every graph in the family $\{T(G) \mid G \in L(S)\}$. Then, $\mathcal{T}$ corresponds to the containment minimal schema over Γ_T and Σ_S that contains $\{T(G) \mid G \in L(S)\}$.*

Proof. The proof follows immediately from Proposition B.4 except for the case when T is empty. Then, however, Γ_T and Σ_T are empty too and so is $\mathcal{T}$. However, the schema that corresponds to $\mathcal{T}$ is also empty and it recognizes only empty graphs. As such it is the containment minimal schema over Γ_T and Σ_T that contains $\{T(G) \mid G \in L(S)\} \subseteq \{\emptyset\}$. □

To move to reducing entailment of statements to query containment we repeat the definitions of the relevant queries but in this version we clearly indicate the transformation in question. More precisely, For a transformation T, $A, B \in \Gamma_T$, and $r \in \Sigma_T$ we define:

$$\begin{aligned}Q_A^T(\bar{x}) &= \big\{q(\bar{x}) \;\big|\; A\big(f_A(\bar{x})\big) \leftarrow q(\bar{x}) \in T\big\},\\ Q_{A,r,B}^T(\bar{x}, \bar{y}) &= \big\{q(\bar{x}, \bar{y}) \;\big|\; r\big(f_A(\bar{x}), f_B(\bar{y})\big) \leftarrow q(\bar{x}, \bar{y}) \in T\big\},\\ Q_{A,r^-,B}^T(\bar{x}, \bar{y}) &= \big\{q(\bar{y}, \bar{x}) \;\big|\; r\big(f_B(\bar{y}), f_A(\bar{x})\big) \leftarrow q(\bar{y}, \bar{x}) \in T\big\}.\end{aligned}$$

Now, we prove that the entailment of $\top \sqsubseteq \bigsqcup \Gamma_T$ is reduced to query containment.

Lemma B.6. *Given a schema S and a transformation T, $(T, S) \models \top \sqsubseteq \bigsqcup \Gamma_T$ if and only if $\exists \bar{y}. Q_{A,R,B}^T(\bar{x}, \bar{y}) \subseteq_S Q_A^T(\bar{x})$ for every $A, B \in \Gamma_T$ and $R \in \Sigma_T^{\pm}$.*

Proof. For the *if* direction, we take any graph $G \in L(S)$ and any element in $u \in dom(T(G))$. This element has been introduced by node rule or by an edge rule, but only the latter is of concern. Thus, assume that $u = f(t)$ has been generated by the rule $R(f_A(\bar{x}), f_B(\bar{y})) \leftarrow Q(\bar{x}, \bar{y})$ with the valuation $\bar{x} = t$ and $\bar{y} = t'$. Since $(t, t') \in Q^T_{A,R,B}(G)$ and $\exists \bar{y}.Q^T_{A,R,B}(\bar{x}, \bar{y}) \subseteq_S Q^T_A(\bar{x})$, $t \in Q^T_A(G)$, and therefore, there is a node rule $A(f_A(\bar{x})) \leftarrow Q'(\bar{x})$ such that $t \in Q'(G)$. Consequently, $u \in A^{T(G)}$.

For the *only if* direction, we take any $G \in L(S)$ and any answer $(t, t') \in Q^T_{A,R,B}(\bar{x}, \bar{y})$ which implies that $(t, t') \in q(\bar{x}, \bar{y})$ for some rule $R(f_A(\bar{x}), f_B(\bar{y})) \leftarrow q(\bar{x}, \bar{y})$. Consequently, $T(G)$ contains the fact $R(f_A(t), f_B(t'))$. Since $T(G)$ satisfies the statement $\top \sqsubseteq \bigsqcup \Gamma_T$ and nodes constructed with f_A can only be part of node label assertions with A, $T(G) \models f_A(t)$. Therefore, there must be a rule $A(f_A(\bar{x}) \leftarrow q'(\bar{x})$ that generated the fact $f_A(t)$ with the valuation $\bar{x} = t$. Consequently, $t \in [Q_A]^{T(G)}$. □

Lemma B.7. *Take a schema S and a transformation T, such that $\Gamma_T \subseteq \Gamma_S$, $\Sigma_S \subseteq \Sigma_S$, and $(T, S) \models \top \sqsubseteq \bigsqcup \Gamma_T$. For any $A, B \in \Gamma_T$ and any $R \in \Sigma^{\pm}_T$ we have that*

$$(T,S) \models A \sqsubseteq \exists R.B \quad \textit{iff} \quad Q_A(\bar{x}) \subseteq_S Q^T_{A,R,B}(\bar{x}),$$
$$(T,S) \models A \sqsubseteq \nexists R.B \quad \textit{iff} \quad \exists \bar{y}.Q_A(\bar{x}) \wedge Q^T_{A,R,B}(\bar{x}, \bar{y}) \subseteq_S \varnothing,$$
$$(T,S) \models A \sqsubseteq \exists^{\leq 1} R.B \quad \textit{iff}$$
$$\exists \bar{x}.Q^T_A(\bar{x}) \wedge Q^T_{A,R,B}(\bar{x}, \bar{y}) \wedge Q^T_{A,R,B}(\bar{x}, \bar{z}) \subseteq_S \textstyle\bigwedge_i [\epsilon(y_i, z_i)].$$

Proof. We prove each of the 3 claims separately.

(1) For the *if* part, we fix a graph $G \in L(S)$ and take any node $u = f_A(t)$ with label A in $T(G)$. Thus, there is a node rule $A(f_A(\bar{x})) \leftarrow q(\bar{x})$ such that $t \in [q(\bar{x})]^G$ and consequently, $t \in [Q^T_A(\bar{x})]^G$. Since $Q^T_A(\bar{x}) \subseteq_S Q^T_{A,R,B}(\bar{x}), t \in [Q^T_{A,R,B}(\bar{x})]^G$ and there exists rule $R(f_A(\bar{x}), f_B(\bar{y})) \leftarrow q'(\bar{x}, \bar{y})$ such that $(t, t') \in [q'(\bar{x}, \bar{y})]^G$. Consequently, $T(G)$ contains the edge $R(f_A(t), f_B(t'))$. Because $T(G)$ satisfies $\top \sqsubseteq \bigsqcup \Gamma_T$, there is also a rule $B(f_B(\bar{y})) \leftarrow q''(\bar{y})$ such that $t' \in [q''(\bar{y})]^G$, and hence the node $f_B(t')$ has label B in G.
For the *only if* part, we fix a graph $G \in L(S)$ and take any $t \in [Q^T_A]^G$, which means that there is a node rule $A(f_A(\bar{x})) \leftarrow q(\bar{x})$ with $t \in [q(\bar{x})]^G$. Consequently, $A(f_A(t))$ belongs to $T(G)$. Since $\mathcal{M}_0(G) \models A \sqsubseteq \exists R.B$, G has an edge $R(f_A(t), v)$ and the node v has label B. This edge must be generated by an edge rule $R(f_A(\bar{x}), f_B(\bar{y})) \leftarrow q'(\bar{x}, \bar{y})$. Consequently, t belongs to the answers to $\exists \bar{y}.q'(\bar{x}, \bar{y})]$ which is contained in $Q_{A,R,B}(\bar{x})$ modulo S.

(2) The proof of this statement is by contradiction and it uses arguments that are analogous to those used in the proof of the above claim and we only outline it. We take a graph $G \in L(S)$ such that in $T(G)$ there is a node $f_A(t)$ with label A and an R-edge to a node with with label B. This happens if and only if the intersection of $Q_A(\bar{x})$ and $\exists \bar{y}.Q_{A,R,B}(\bar{x}, \bar{y})$ is non-empty.

(3) Similarly, the proof is by contradiction but uses argument analogous to those in the proof of the first claim and we only outline it. We take a graph $G \in L(S)$ such that $T(G)$ has an A-node $f_A(t)$ which has R-edges to two different B-nodes $f_B(t'_1)$ and $f_B(t'_2)$. This is possible if and only if the query $\exists \bar{x}.Q^T_A(\bar{x}) \wedge Q^T_{A,R,B}(\bar{x}, \bar{y})$ returns both t'_1 and t'_2, and consequently, $\exists \bar{x}.Q^T_A(\bar{x}) \wedge Q^T_{A,R,B}(\bar{x}, \bar{y}) \wedge Q^T_{A,R,B}(\bar{x}, \bar{z}) \subseteq_S \bigwedge_i \epsilon(y_i, z_i)$ returns (t'_1, t'_2). Because node constructors are invective, $t'_1 \neq t'_2$, and therefore, (t'_1, t'_2) cannot be answer to $\bigwedge_i \epsilon(y_i, z_i)$. □

For testing equivalence of two transformations we observer that since a transformation is equivalent to its trimmed version, two transformations T_1 and T_2 are equivalent modulo S if and only if they trimmed versions $trim_S(T_1)$ and $trim_S(T_2)$ are equivalent modulo S. In the following lemma, $Q_1 \equiv_S Q_2$ is short for $Q_1 \subseteq_S Q_2$ and $Q_2 \subseteq_S Q_1$.

Lemma B.8. *Take a schema S and two transformations T_1 and T_2 that are both trimmed modulo S. We have that $T_1 \equiv_S T_2$ if and only if the following conditions are satisfied:*

(1) $\Gamma_{T_1} = \Gamma_{T_2}$ *and* $\Sigma_{T_1} = \Sigma_{T_2}$,
(2) $Q^{T_1}_A(\bar{x}) \equiv_S Q^{T_2}_A(\bar{x})$ *for every* $A \in \Gamma_{T_1}$,
(3) $Q^{T_1}_{A,R,B}(\bar{x}, \bar{y}) \equiv_S Q^{T_2}_{A,R,B}(\bar{x}, \bar{y})$ *for every* $A, B \in \Gamma_{T_1}$, $R \in \Sigma_{T_1}$.

Proof. The *if* part is trivial. We prove the *only if* part by proving the contraposition: we show that if one of the conditions (1), (2), and (3) is not satisfied, then $T_1 \not\equiv_S T_2$.

If (1) is not satisfied, then one of the transformations has at least one rule ρ that generate a node or an edge with a label that is not employed by the other transformations. Since both transformations are trimmed, there exists an input graph G such that the rule ρ produces objects on the output. But then $T_1(G) \neq T_2(G)$.

If (2) is not satisfied, then there is an input graph G such that one of the transformations generates a node that the other does not. Hence, $T_1(G) \neq T_2(G)$.

If (3) is not satisfied, then analogously, there is an input graph G such that one of the transformations generates an edge that the other does not. Hence, $T_1(G) \neq T_2(G)$. □

C ROLLING UP QUERIES

We next show how to reduce the non-satisfaction of a acyclic UC2RPQ Q to the satisfaction of a Horn-$\mathcal{ALCIF}$ TBox $\mathcal{T}_{\neg Q}$. The TBox Q is basically a recursive program that defines a collection of sets (monadic relations) of nodes. We illustrate the this construction with the following example.

Example C.1. We take the following Boolean query.

$$Q_0 = \exists x_0, x_1, x_2, x_3.\, (a \cdot b^* \cdot c)(x_2, x_1) \wedge (A)(x_3, x_1) \wedge (a^-)(x_1, x_0).$$

We construct a TBox that essentially simulates automata for the regular expressions, which are presented in Figure 5.

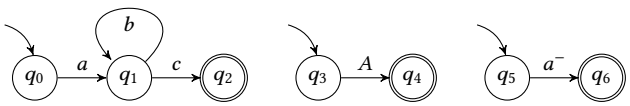

Figure 5: Automata for regular expressions of Q.

The TBox $\mathcal{T}_{\neg Q_0}$ consists of the following constraints.

$$\top \sqsubseteq q_0, \quad q_0 \sqsubseteq \forall a.q_1, \quad q_1 \sqsubseteq \forall b.q_1, \quad q_1 \sqsubseteq \forall c.q_2,$$
$$\top \sqsubseteq q_3, \quad q_3 \sqcap A \sqsubseteq q_4, \quad q_2 \sqcap q_4 \sqsubseteq q_5, \quad q_5 \sqsubseteq \forall a^-.q_6, \quad q_6 \sqsubseteq \bot.$$

□

$\mathcal{T}_{\neg Q}$ introduces a set fresh node labels Γ_Q and the satisfaction $\mathcal{T}_{\neg Q}$ is defined in terms of the existence of valuations of symbols in Γ_Q. More precisely, given a graph G over Γ_0 and Σ_0 and a TBox $\mathcal{T}$ over $\Gamma_0 \cup \Gamma_1$ and Σ_0, we say that G *satisfies* $\mathcal{T}$ if and only if there is an interpretation $\cdot^U : \Gamma_1 \to \mathcal{P}(dom(G))$ of symbols in Γ_1 such that $G \cup U \models \mathcal{T}$.

Lemma C.2. *Given a Boolean acyclic UC2RPQs Q, one can compute in polynomial time a Horn-$\mathcal{ALCIF}$ TBox $\mathcal{T}_{\neg Q}$ and a reserved set of concept names Γ_Q such that for every G that does not use labels in Γ_Q, $G \not\models Q$ if and only if G satisfies $\mathcal{T}_{\neg Q}$.*

Proof. We prove the lemma for queries that are Boolean C2RPQs that are acyclic and connected. The claim extends to unions of Boolean acyclic C2RPQs in a straightforward fashion: it suffices to take the union of the desired TBoxes of all connected components of the union. Consequently, the query can be seen as a tree and we assume that it is defined with the following grammar:

$$Q ::= \varphi(Q, \ldots, Q),$$

where φ is a two-way regular expression over Σ and Γ. For instance, the query from Example C.1 is represented as $Q_0 = a^-(A, a \cdot b^* \cdot c)$. We express the semantics of such defined queries as the set of all nodes that satisfy it.

$$[\varphi(Q_1, \ldots, Q_k)]^G = \{u \in dom(G) \mid \exists v.\, (v,u) \in [\varphi]_G,\, v \in \textstyle\bigcap_i [Q_i]^G\}.$$

Naturally, a graph G satisfies Q iff $[Q]^G \neq \emptyset$.

Now, fix an acyclic Boolean C2RPQ Q and let Φ be the set of all two-way regular expressions used in Q. For any $\varphi \in \Phi$ by $N_\varphi = (K_\varphi, I_\varphi, \delta_\varphi, F_\varphi)$ we denote an ϵ-free NDA over the alphabet $\Sigma \cup \Gamma$ that recognizes φ, where K_φ is a finite set of states, $I_\varphi \subseteq K_\varphi$ is the set of initial states, $F_\varphi \subseteq K_\varphi$ is the set of final states, and $\delta_\varphi \subseteq K_\varphi \times (\Sigma \cup \Gamma) \times K_\varphi$ is the transition table. We assume that the size of N_φ is polynomial in the size of the expression φ (such automaton can be obtained for instance with the standard Glushkov technique). We also assume that the sets of states are pair-wise disjoint.

The set of additional node labels consists of the states of automata: $\Gamma_Q = \bigcup_\varphi K_\varphi$. The constructed TBox consists of two subsets of rules: $\mathcal{T}_{\neg Q} = \mathcal{T}_1 \cup \mathcal{T}_0$. The set $\mathcal{T}_1$ encodes transitions of the automata that simulate their execution.

(1) For every φ and every $(q, R, q') \in \delta_\varphi$ such that $R \in \Gamma^\pm$, $\mathcal{T}_1$ contains $q \sqsubseteq \forall R.q'$;
(2) For every φ and every $(q, A, q') \in \delta_\varphi$ such that $A \in \Sigma$, $\mathcal{T}_1$ contains $q \sqcap A \sqsubseteq q'$;
(3) For every node φ of Q with children $\varphi_1, \ldots, \varphi_k$, every $q \in I_\varphi$, $\mathcal{T}_1$ contains $\bigsqcap\{q' \mid q' \in F_{\varphi_i},\, 1 \leq i \leq k\} \sqsubseteq q$. Note that when φ is a leaf of Q, then $\mathcal{T}_1$ contains $\top \sqsubseteq q$ for every $q \in I_\varphi$.

The set $\mathcal{T}_0$ contains denial rules that ensure lack of valid run.

(4) For every $q \in F_\varphi$ of the root φ of Q, $\mathcal{T}_0$ contains $q \sqsubseteq \bot$;

Now, we fix a graph G whose node labels do not use any symbol in Γ_Q. We first argue that there is a unique minimal interpretation $U_0 : \Gamma_Q \to \mathcal{P}(dom(G))$ such that $G \cup U_0 \models \mathcal{T}_1$. Indeed, since the rules are Horn-like, an intersection of two models of $\mathcal{T}_1$ is also a model of $\mathcal{T}_1$.

Next, we prove the main claim with an inductive argument which requires defining subqueries of Q. For $\varphi \in \Phi$ and $q \in K_\varphi$ by Q_q we denote the query $\psi(Q_1, \ldots, Q_k)$, where $Q_1, \ldots, Q_k$ are children of φ in Q and ψ is the two-way regular expression corresponding to the automaton $M_{\varphi,q} = (K_\varphi, I_q, \delta_\varphi, \{q\})$ (essentially, we make q the only final state). We claim that for any $\varphi \in \Phi$, any $q \in K_\varphi$, and any $u \in N_G$ we have

$$u \in [Q_q]^G \quad \textit{iff} \quad u \in q^{U_0}.$$

In essence, the unary predicate q identifies all nodes at which the subquery Q_q is satisfied. We prove the above claim with double induction: firstly over the height of the subquery $Q_q = \psi(Q_1, \ldots, Q_k)$, and secondly, over the length of the witnessing path for $(v,u) \in [\psi]^G$ such that $v \in \bigcap_i [Q_i]^G$.

If we let $I_\varphi = \{q_1, \ldots, q_k\}$, then Q is equivalent to the union of $Q_{q_1} \cup \ldots \cup Q_{q_k}$. Consequently, Q is satisfied at a node $u \in N_G$ iff $u \in q_i{}^{U_0}$ for some $i \in \{1, \ldots, k\}$. As such, Q is not satisfied at any node of G if and only if $U_0 \models q_i \sqsubseteq \bot$ for every $i \in \{1, \ldots, k\}$ i.e., $U_0 \models \mathcal{T}_0$. We finish the proof by observing that if the minimal model U_0 does not satisfy $\mathcal{T}_0$, then none of supersets of U_0 does. □

D PROOFS FOR CONTAINMENT

Lemma D.1. *Given a schema S, a UC2RPQ $P(\bar{x})$, and an acyclic UC2RPQ $Q(\bar{x})$, one can compute in polynomial time a schema S°, a Boolean UC2RPQ P°, and a Boolean acyclic UC2RPQ Q° such that $P(\bar{x}) \subseteq_S Q(\bar{x})$ iff $P^\circ \subseteq_{S^\circ} Q^\circ$.*

Proof. Let $\bar{x} = (x_1, x_2, \ldots, x_n)$ and let $\Gamma_S = \{A_1, \ldots, A_k\}$. We take a fresh node labels $X_1, \ldots, X_n \notin \Sigma_S$ and fresh edge labels $r_1, r_2, \ldots, r_n \notin \Sigma_S$. The schema S° is obtained from S as follows:

$$\Gamma_{S^\circ} = \Gamma_S \cup \{A_0\},$$
$$\Sigma_{S^\circ} = \Sigma_S \cup \{r_1, \ldots, r_n\},$$
$$\delta_{S^\circ}(A,R,B) = \begin{cases} \delta_S(A,R,B) & \text{if } A, B \in \Gamma_S \text{ and } R \in \Sigma_S^\pm, \\ ? & \text{if } A = X_i,\, R \in \{r_i, r_i^-\}, \text{ and } B \in \Gamma_S, \\ 0 & \text{otherwise.} \end{cases}$$

Now, the queries P° and Q° are obtained from $P(\bar{x})$ and $Q(\bar{x})$ by quantifying existentially $x_1, x_2, \ldots, x_n$ and also adding atoms $\exists y.(X_i \cdot r_i)(y, x_i)$ for every $i \in \{1, \ldots, n\}$. It is routine to check that $P(\bar{x}) \subseteq_S Q(\bar{x})$ if and only if $P^\circ \subseteq_{S^\circ} Q^\circ$. There are two key facts. Firstly, 2RPQs in P and Q do not use labels $r_1, r_2, \ldots, r_n$ (nor wildcards) and consequently cannot traverse edges with such labels. Secondly, the schema S° ensures that the original regular expression can be witnessed only by paths that begin and end in nodes with labels in Σ_S only. □

Corollary D.2. *Given a schema S, two unary acyclic 2RPQs $p(x)$ and $q(x)$, one can compute in polynomial time a schema S° and Boolean 2RPQs p° and q° such that $p(x) \subseteq_S q(x)$ iff $p^\circ \subseteq_{S^\circ} p^\circ$.*

Proof. The construction of S° is as in Lemma D.1 and the construction of Boolean RPQs depends on the form of the unary RPQ: 1) if $p(x_1) = \exists x_2.\varphi(x_1, x_2)$, then $p^\circ = r_1 \cdot \varphi$ and 2) if $p(x_1) = \exists x_2.\varphi(y, x)$, then $p^\circ = \varphi \cdot r_1^-$; q° is constructed in the same way. □

Lemma D.3. *$P \subseteq_S Q$ iff $\widehat{P} \subseteq_{\widehat{\mathcal{T}_S}} Q$.*

Proof. Each finite graph falsifying the left-hand side condition falsifies the right-hand side condition as well. For the converse, let G be a finite graph falsifying the right-hand side condition. Without loss of generality we can assume that only labels from $\Gamma_S \cup \Sigma_S$

are used in G. Let G' be obtained by dropping all nodes without a label, as well as edges incident with these nodes. Because all concept inclusions in $\widehat{\mathcal{T}_S}$ that require a witnessing neighbour specify the label of this neighbour, they are not affected by this modification. Other concept inclusions are always preserved when passing to a subgraph. It follows that G' conforms to S. The RPQs in $\widehat{P}$ can only traverse nodes with a label from Γ_S, so $\widehat{P}$ is still satisfied in G'. Then, P is satisfied as well. Q is not satisfied in G', because G' is a subgraph of G. □

Lemma D.4. *$\widehat{P}$ is finitely satisfiable modulo $\widehat{\mathcal{T}_S} \cup \mathcal{T}_{\neg Q}$ iff $\widehat{P}$ is satisfiable modulo $(\widehat{\mathcal{T}_S} \cup \mathcal{T}_{\neg Q})^*$.*

Proof. Suppose that $\widehat{P}$ is satisfied in a finite model G of $\widehat{\mathcal{T}_S} \cup \mathcal{T}_{\neg Q}$. By Theorem 5.4, there is a (possibly infinite) model of $(\widehat{\mathcal{T}_S} \cup \mathcal{T}_{\neg Q})^*$ containing G as a subgraph. This model obviously satisfies $\widehat{P}$.

Conversely, suppose that there is a possibly infinite graph G satisfying $\widehat{P}$ and $\left(\widehat{\mathcal{T}_S} \cup \mathcal{T}_{\neg Q}\right)^*$. Let p be the disjunct of $\widehat{P}$ that is satisfied in G. Let H be the image of p in G, including a finite witnessing path for each RPQ. Note that H is finite. By Theorem 5.4, there is a finite model of $\widehat{\mathcal{T}_S} \cup \mathcal{T}_{\neg Q}$ containing H as a substructure. This models satisfies $\widehat{P}$ as well. □

Lemma D.5. *Every S-driven TBox $\mathcal{T}$ can be reduced in polynomial time so that it contains at most $|\Sigma_S^{\pm}| \cdot |\Gamma_S|^2$ at-most constraints.*

Proof. To achieve this, for each such CI of the form $K \sqsubseteq \exists^{\leq 1} R.K'$ in $\mathcal{T}$ we do one of the following.

- If $\mathcal{T}$ contains $A \sqsubseteq \exists^{\leq 1} R.A'$ for some $A, A' \in \Gamma_S$ such that $A \in K$ and $A' \in K'$, then simply remove $K \sqsubseteq \exists^{\leq 1} R.K'$ from $\mathcal{T}$. This is correct because $A \sqsubseteq \exists^{\leq 1} R.A' \models K \sqsubseteq \exists^{\leq 1} R.K'$.
- Otherwise, because $\mathcal{T}$ is S-driven, it follows that the triple (K, R, K') is not satisfiable modulo $\mathcal{T}$. That is, $\mathcal{T} \models K \sqsubseteq \nexists R.K'$. Since $K \sqsubseteq \nexists R.K' \models K \sqsubseteq \exists^{\leq 1} R.K'$, we can safely replace $K \sqsubseteq \exists^{\leq 1} R.K'$ with $K \sqsubseteq \nexists R.K'$ in $\mathcal{T}$.

The resulting TBox $\mathcal{T}'$ is equivalent to $\mathcal{T}$ and it only contains at-most constraints involving single concept names from Γ_S. The number of those is clearly bounded by $|\Sigma_S^{\pm}| \cdot |\Gamma_S|^2$. □

Lemma D.6. *Let $\mathcal{T}$ be an S-driven Horn-$\mathcal{ALCIF}$ TBox that was obtained from $\widehat{\mathcal{T}_S} \cup \mathcal{T}_{\neg Q}$ by reversing some finmod cycles. For every satisfiable finmod cycle*

$$K_1, R_1, \ldots, K_{n-1}, R_{n-1}, K_n$$

in $\mathcal{T}$ there exist unique $A_1, A_2, \ldots, A_n \in \Gamma_S$ such that $A_i \in K_i$ for all $i \leq n$, and

$$A_1, R_1, \ldots, A_{n-1}, R_{n-1}, A_n$$

is a finmod cycle in $\mathcal{T}$

Proof. Since all triples in $K_1, R_1, \ldots, K_{n-1}, R_{n-1}, K_n$ are satisfiable, all CIs $K_i \sqsubseteq \exists R_i.K_{i+1}$ and $K_{i+1} \sqsubseteq \exists^{\leq 1} R_i^-.K_i$ are relevant for $\mathcal{T}$. We cannot simply apply the fact that $\mathcal{T}$ is S-driven, because these CIs need not belong to $\mathcal{T}$: they are only entailed by $\mathcal{T}$. The proof will proceed in several steps.

The first step is to see that each K_i contains a label from Γ_S. Towards contradiction, suppose it does not. We construct a graph witnessing that $\mathcal{T}$ does not entail $K_i \sqsubseteq \exists R_i.K_{i+1}$, which is a contradiction. Let T_i be the tree-shaped graph obtained by unravelling some model of $\mathcal{T}$ witnessing that (K_i, R_i, K_{i+1}) is satisfiable, from a node u satisfying K_i. Clearly, T_i is also a model of $\mathcal{T}$, its root u satisfies K_i and has an R_i-successor u' satisfying K_{i+1}. We construct G as the graph with a single node u_0 whose labels are copied from the root u of T_i but with any letter from Γ_S dropped. To see that $G \not\models K_i \sqsubseteq \exists R_i.K_{i+1}$, note that as $u \in (K_i)^{T_i}$ and K_i contains no labels from Γ_S, also $u_0 \in (K_i)^G$; but clearly u_0 has no R_i-successors at all. Let us check that $G \models \mathcal{T}$.

- New CIs of the form $K \sqsubseteq A$ are not introduced by reversing cycles, so it suffices to look at ones from $\widehat{\mathcal{T}_S} \cup \mathcal{T}_{\neg Q}$. There, such CIs are only present in $\mathcal{T}_{\neg Q}$ and always satisfy $A \notin \Gamma_S$ (see the proof of Lemma C.2). Hence, as they were satisfied in T_i and G was obtained by dropping labels from Γ_S, they still hold in G.
- CIs of the form $K \sqsubseteq \bot$ in $\mathcal{T}$ were satisfied in T_i and they cannot be violated by dropping labels (recall that K does not use negation).
- All CIs of the forms $K \sqsubseteq \forall R.K'$, $K \sqsubseteq \nexists R.K'$, and $K \sqsubseteq \exists^{\leq 1} R.K$ are trivially satisfied in G.
- Consider a CI of the form $K \sqsubseteq \exists R.K'$ from $\mathcal{T}$. Suppose that $u_0 \in K^G$. Then also $u \in K^{T_i}$. This means that the CI was "fired" in T_i, which implies that (K, R, K') is satisfiable modulo $\mathcal{T}$ and $K \sqsubseteq \exists R.K'$ is relevant for $\mathcal{T}$. As $\mathcal{T}$ is S-driven, it follows in particular that K contains a label from Γ_S. But this contradicts the fact that $u_0 \in K^G$. Hence, $K \sqsubseteq \exists R.K'$ is trivially satisfied in G.

Thus we have shown that $G \models \mathcal{T}$. This concludes the first step.

Now, as all K_i contain a label from Γ_S and all triples (K_i, R_i, K_{i+1}) are satisfiable modulo $\mathcal{T}$, it follows that for each i there exists exactly one label $A_i \in \Gamma_S$ such that $A_i \in K_i$. It remains to show that $A_i \sqsubseteq \exists R_i.A_{i+1}$ and $A_{i+1} \sqsubseteq \exists^{\leq 1} R_i^-.A_i$.

Let us begin with $A_i \sqsubseteq \exists R_i.A_{i+1}$. Consider graph G obtained from T_i (same as above) by removing all subtrees rooted at R_i-successors of the root that satisfy K_{i+1}. Clearly, $G \not\models K_i \sqsubseteq \exists R_i.K_{i+1}$. As $\mathcal{T} \models K_i \sqsubseteq \exists R_i.K_{i+1}$, it follows that $G \not\models \mathcal{T}$. Then, some CI of the form $K \sqsubseteq \exists R.K'$ from $\mathcal{T}$ is violated in G, because CIs of other forms are preserved when passing to a subgraph. In particular, it must be the case that the root of G satisfies K. But then also the root of T_i satisfies K and since $T_i \models \mathcal{T}$, the root u of T_i has an R-successor u' that satisfies K'. This means that $K \sqsubseteq \exists R.K'$ is relevant for $\mathcal{T}$. Because $\mathcal{T}$ is S-driven, it must contain $A \sqsubseteq \exists R.A'$ for some $A, A' \in \Gamma_S$ such that $A \in K$, $A' \in K'$. As the root of G satisfies both K and K_i, and we know that $A \in K$ and $A_i \in K_i$ and that labels from Γ_S are exclusive, it follows that $A = A_i$. We claim that also $R = R_i$ and $A' = A_{i+1}$. If $R \neq R_i$, then u' is not an R_i-successor of the root in T_i, and it has not been removed in G. That would imply that G actually does satisfy $K \sqsubseteq \exists R.K'$. Since we know this is not the case, we conclude that $R = R_i$. Similarly, suppose that $A' \neq A_{i+1}$. Because u' satisfies K' and $A' \in K'$, it must have label A'. But then u' cannot have label A_{i+1}, which means it cannot satisfy K_{i+1}, and has not been removed in G. This yields a contradiction just like before and we can conclude that $A' = A_{i+1}$. Wrapping up, we have seen that $A \sqsubseteq \exists R.A'$ belongs to $\mathcal{T}$ and that

$A = A_i$, $R = R_i$, and $A' = A_{i+1}$. This means that $A_i \sqsubseteq \exists R_i.A_{i+1}$ belongs to $\mathcal{T}$.

Finally, let us see that $A_{i+1} \sqsubseteq \exists^{\leq 1} R_i^-.A_i$ belongs to $\mathcal{T}$. Consider the model T_i but reorganize it so that the root u satisfies K_{i+1} and has an R_i^--successor u' satisfying K_i. Let G be the graph obtained from T_i by duplicating the whole subtree rooted at u', and adding an R_i^--edge from u to the root u'' of the copy. Clearly $G \not\models K_{i+1} \sqsubseteq \exists^{\leq 1} R_i^-.K_i$ and since $\mathcal{T} \models K_{i+1} \sqsubseteq \exists^{\leq 1} R_i^-.K_i$, we conclude that $G \not\models \mathcal{T}$. It follows immediately that G violates some CI of the form $K \sqsubseteq \exists^{\leq 1} R.K'$ from $\mathcal{T}$, as CIs of other forms are not affected by the modification turning T_i to G. Similarly, it must hold that $R = R_i^-$, and that u satisfies K and u' and u'' satisfy K'. It follows that $K \sqsubseteq \exists^{\leq 1} R.K'$ is relevant, $A_{i+1} \in K$, $A_i \in K'$, and $A_{i+1} \sqsubseteq \exists^{\leq 1} R_i^-.A_i$ belongs to $\mathcal{T}$. □

Lemma D.7. *For $\mathcal{T} = \widehat{\mathcal{T}_S} \cup \mathcal{T}_{\neg Q}$, the completion $\mathcal{T}^*$ can be computed in* EXPTIME.

Proof. Construct a graph $G_{\mathcal{T}}$ over all possible intersections K of concept names used in $\mathcal{T}$, including an edge with label $R \in \Sigma^{\pm}$ from K to K' iff

$$\mathcal{T} \models K \sqsubseteq \exists R.K' \quad \text{and} \quad \mathcal{T} \models K' \sqsubseteq \exists^{\leq 1} R^-.K\,.$$

$G_{\mathcal{T}}$ has exponential size and can be constructed in EXPTIME, because CI entailment by Horn-$\mathcal{ALCIF}$ TBoxes can be tested in exponential time [24]. Repeat the following until the graph stops changing. Pick an R-edge from K to K' such that there is no R^--edge from K' to K. Check if there exists a path from K' to K in $G_{\mathcal{T}}$. If so, the identified path combined with the R-edge from K to K' constitutes a finmod cycle

$$K_1, R_1, \ldots, K_{n-1}, R_{n-1}, K_n$$

in $\mathcal{T}$. Add to $G_{\mathcal{T}}$ an R_i^--edge from K_{i+1} to K_i for all $i < n$ and extend $\mathcal{T}$ with the corresponding concept inclusions. Note that this includes an R^--edge from K' to K and concept inclusions

$$K' \sqsubseteq \exists R^-.K \quad \text{and} \quad \mathcal{T} \models K \sqsubseteq \exists^{\leq 1} R.K'\,.$$

Moreover, if there are unique $A_1, A_2, \ldots, A_n \in \Gamma_S$ such that $A_i \in K_i$ for $i \leq n$, check if

$$A_1, R_1, \ldots, A_{n-1}, R_{n-1}, A_n$$

is a cycle in $G_{\mathcal{T}}$. If so, add to G an R_i^- edge from A_{i+1} to A_i, and the corresponding CIs to $\mathcal{T}$. By Lemma D.6, this ensures that the extended $\mathcal{T}$ is S-driven. We can now reduce it and recompute $G_{\mathcal{T}}$ based on the updated $\mathcal{T}$. Using the complexity bounds for CI entailment given in Corollary E.7, we conclude that this can be done in EXPTIME. Note that we are indeed relying on the more precise complexity bounds here, because at later iterations of the cycle reversing procedure the TBox might well contain exponentially many concept inclusions. However, it has still only the original concept names and, after reducing, only a polynomial number of at-most restrictions. □

E PROOFS FOR SATISFIABILITY

E.1 Introductory lemmas

We begin by showing the two lemmas mentioned in the body of the paper.

Lemma E.1. *For $c \geq 1$, if a finite connected c-sparse graph has only nodes of degree at least 2, then it is $(2c, 3c)$-skeleton.*

Proof. Let G be a finite connected c-sparse graph without nodes of degree 0 or 1. We claim that G consists of at most $2c$ nodes connected by at most $3c$ paths disjoint modulo endpoints. If G is empty, we are done. Otherwise, we eliminate vertices of degree 2 that are incident with two different edges by merging these edges into a single edge. This process results in a c-sparse multigraph G_0, whose edges represent simple paths in G. This graph is either a single node with a loop or all its nodes have degree at least 3. In the first case it follows that G is a single cycle, and thus a $(1,1)$-skeleton. In the second case, assuming that G_0 has n nodes and m edges, we have $3n/2 \leq m \leq n + c$. It follows that $c > 0$, $n \leq 2c$, $m \leq 3c$. □

Lemma E.2. *If p is satisfied in a $|p|$-sparse graph G, then G contains a $(4|p|, 5|p|)$-skeleton H, extending the skeleton of G, such that all variables of p are mapped to distinguished nodes of H and G can be obtained by attaching finitely many finitely branching trees to H.*

Proof. The skeleton H_0 of G is a $(2|p|, 3|p|)$-skeleton. Consider a match of p in G. Some variables of p might well be matched to nodes on the paths connecting the distinguished nodes of H_0 or in the attached trees. We define H as follows. First, we add to H as distinguished nodes all images of variables of p that lie on the paths connecting distinguished nodes of H_0. Next, for each attached tree T that contains an image of a variable of p, we add to H as distinguished nodes all the images of variables of p that belong to T together with all their least common ancestors in T, as well as the node of H to which the root of T is connected. All ancestors (in T) of these nodes are added to H as ordinary nodes. The skeleton H thus obtained has the required properties. □

E.2 The main result

The goal of this section is to prove the following theorem.

Theorem E.3. *Given a C2RPQ p and an $\mathcal{ALCIF}$ TBox $\mathcal{T}$ using k concept names and ℓ at-most constraints, one can decide in time $O\big(\text{poly}(|\mathcal{T}|) \cdot 2^{\text{poly}(|p|,k,\ell)}\big)$ if there exists a $|p|$-sparse graph that satisfies p and $\mathcal{T}$.*

The proof of Theorem E.3 is not very hard, but it combines several components and requires developing some machinery. Let us begin with a road map.

Relying on Lemma E.2, we guess a $(4|p|, 5|p|)$-skeleton H. The distinguished nodes of H are represented explicitly, together with all their labels, but each of the connecting paths is represented by a single *symbolic edge*. Note that there might be multiple symbolic edges between the same pair of distinguished nodes, representing different paths. We need to check that H can be completed to a graph G by materializing the symbolic edges into paths and attaching finitely many finitely branching trees in such a way that G is a model of $\mathcal{T}$ and there is a match of p in G that maps variables of p to distinguished nodes of H.

To achieve this, we guess an annotation of H that summarizes how the witnessing paths of p can traverse the parts of G missing from H, and which witnesses of distinguished nodes required by $\mathcal{T}$ these parts provide (Section E.3). We then check if these promises

of the annotation are sufficient to guarantee that p and $\mathcal{T}$ are satisfied (Section E.4). Finally, we verify that the promises of the annotation can be fulfilled: we check if we can attach trees to the distinguished nodes and expand the symbolic edges into finite paths with attached trees in a way that matches the promises of the annotation and respects the TBox $\mathcal{T}$ (Section E.5).

E.3 Annotated skeleta

Let Γ_p, Σ_p, $\Gamma_\mathcal{T}$, $\Sigma_\mathcal{T}$ be the sets of edge and node labels used in p and $\mathcal{T}$, respectively. In what follows we only consider graphs and skeleta using only node labels from $\Gamma_p \cup \Gamma_\mathcal{T}$ and edge labels from $\Sigma_p \cup \Sigma_\mathcal{T}$.

Let Φ be the set of two-way regular expressions used in p. For each $\varphi \in \Phi$ we fix an equivalent linear size non-deterministic automaton $\mathcal{A}_\varphi$ over the alphabet $\Gamma_p \cup \Sigma_p^\pm$ with states K_φ, initial states $I_\varphi \subseteq K_\varphi$, and final states $F_\varphi \subseteq K_\varphi$. We assume that all K_φ are pairwise disjoint and let $\delta = \bigcup_{\varphi\in\Phi} \delta_\varphi$.

An *annotation* of skeleton H is given by the following functions.

- β_{src} and β_{tgt} record information about the source and target of the paths represented by each symbolic edge: they both map each symbolic edge e to $\left(\Sigma_p \cup \Sigma_\mathcal{T}\right)^\pm \times 2^{\Gamma_p \cup \Gamma_\mathcal{T}}$.
- δ_{node} records how the witnessing paths for p may loop in the subtrees attached to the distinguished nodes. Thus, δ_{node} maps every distinguished node to a subset of $\bigcup_{\varphi\in\Phi} K_\varphi \times K_\varphi$.
- δ_{edge} records how the witnessing paths for p progress along paths (and the trees attached to them) represented by the symbolic edges in the skeleton. Thus, δ_{edge} maps every edge e to a subset of $\bigcup_{\varphi\in\Phi} K_\varphi \times K_\varphi \times \{\circlearrowright, \circlearrowleft, \leftarrow, \rightarrow\}$. If e is an edge from u to v, then $(s, s', \rightarrow) \in \delta_{\text{edge}}(e)$ indicates that some path enters (the part of the model summarized by) the edge e from u in state s, and exits at node v in state s'. Similarly, $(s, s', \circlearrowleft) \in \delta_{\text{edge}}(e)$ indicates a loop: some path enters e from v in state s, and exits at the same node v in state s', etc.

E.4 Verifying annotated skeleta

An annotation of H is *sufficient for TBox* $\mathcal{T}$ if the witnesses recorded by β_{src} and β_{tgt} respect $\mathcal{T}$; that is, for each distinguished node u of H the graph G_u defined below satisfies the TBox $\mathcal{T}_0$ obtained from $\mathcal{T}$ by dropping all concept inclusions of the form $A \sqsubseteq \exists R.B$. To construct G_u we begin from u with labels inherited from H, and then for each symbolic edge e incident with u we add an R-successor v_e of u with label set Λ, where $(R, \Lambda) = \beta_{\text{src}}(e)$ if u is the source of e and $(R, \Lambda) = \beta_{\text{tgt}}(e)$ if u is the target of e.

An annotation is *sufficient for C2RPQ* p if there exists a function η mapping variables of p to distinguished nodes of H such that for each atom $\varphi(x, y)$ of p, there exists a finite witnessing sequence $s_0 u_0 s_1 u_1 \dots s_k u_k$ of states and distinguished nodes of H satisfying the following conditions.

- The witnessing sequence begins and ends correctly; that is, $s_0 \in I_\varphi$, $s_k \in F_\varphi$, $u_0 = \eta(x)$, $u_k = \eta(y)$.
- Each transition step along a symbolic edge (or subtree attached to a distinguished node) updates the state as expected: for each $i < k$ one of the following holds:
  - $(s_i, s_{i+1}, \rightarrow) \in \delta_{\text{edge}}(e)$ for some edge e from u_i to u_{i+1};
  - $(s_i, s_{i+1}, \leftarrow) \in \delta_{\text{edge}}(e)$ for some edge e from u_{i+1} to u_i;
  - $(s_i, s_{i+1}, \circlearrowright) \in \delta_{\text{edge}}(e)$ for some edge e from u_i to some u, and $u_i = u_{i+1}$;
  - $(s_i, s_{i+1}, \circlearrowleft) \in \delta_{\text{edge}}(e)$ for some edge e from some u to u_i, and $u_i = u_{i+1}$;
  - $(s_i, s_{i+1}) \in \delta_{\text{node}}(u_i)$ and $u_i = u_{i+1}$.

We point out that the witnessing sequence may traverse a symbolic edge multiple times. In consequence, each tuple in $\delta_{\text{edge}}(e)$ must be "realised" by the single path represented by e (and the attached trees).

Proposition E.4. *One can decide if a given annotated skeleton is sufficient for p and $\mathcal{T}$ in* PTIME.

Proof. To check that the annotated skeleton is sufficient for $\mathcal{T}$ it is enough to examine the graphs G_u for each distinguished node u of the skeleton.

Checking that the annotated skeleton is sufficient for p amounts to guessing the function η and for each atom $\varphi(x, y)$ running a reachability test in the product graph whose nodes combine distinguished nodes of the skeleton with states from K_φ, where edges are defined according to the symbolic edges in the skeleton and the triples from δ_{edge}. In the reachability test we check if there exists a path beginning in $\{\eta(x)\} \times I_\varphi$ and ending in $\{\eta(y)\} \times F_\varphi$. □

E.5 Implementing annotated skeleta

Consider an annotated skeleton $\mathcal{H} = \left(H, \beta_{\text{src}}, \beta_{\text{tgt}}, \delta_{\text{edge}}, \delta_{\text{node}}\right)$. We say that a graph G *implements* $\mathcal{H}$ if G is obtained from H by replacing each symbolic edge e with a path π_e connecting the endpoints of e and by attaching finitely many finitely branching trees in a way consistent with the annotations, in the following sense.

- For each symbolic edge e from u to u', the subgraph G_e of G that consists of π_e and all trees attached to the internal nodes of π_e is correctly summarized in the annotations:
  - for each $(s, s', d) \in \delta_{\text{edge}}(e)$ with $s, s' \in K_\varphi$ there is a path in G_e with endpoints (u, u) if $d = \circlearrowright$, (u, u') if $d = \rightarrow$, (u', u') if $d = \circlearrowleft$, and (u', u) if $d = \leftarrow$, on which $\mathcal{A}_\varphi$ moves from state s to state s';
  - if $\beta_{\text{src}}(e) = (R_1, \Lambda_1)$ and $\beta_{\text{src}}(e) = (R_2, \Lambda_2)$, then the first edge of π_e is an R-edge, the last edge of π_e is an R_2^--edge, the second node on π_e has the labels set Λ_1, and the penultimate node on π_e has label set Λ_2.
- For each distinguished node u, the trees attached to u are summarized correctly in the annotations: for each $(s, s') \in \delta_{\text{node}}(u)$ with $s, s' \in K_\varphi$ there is a tree $T_u^{s,s'}$ attached to u and a path that starts and ends in u and otherwise only visits nodes of $T_u^{s,s'}$, on which $\mathcal{A}_\varphi$ moves from state s to state s'.
- G is a model of $\mathcal{T}$.

Note that all the missing pieces of the graph are essentially trees (finitely branching, but typically infinite). Indeed, each $T_u^{s,s'}$ simply is a tree, but also G_e can be viewed as a tree: its root is the source of e, the root has exactly one child, the path π_e constitutes a special finite branch ending in the target of e which is a leaf in this tree. Importantly, each $(s, s') \in \delta_{\text{node}}(u)$ is witnessed by a finite subgraph of $T_u^{s,s'}$, and each triple $(s, s', d) \in \delta_{\text{edge}}(e)$ is witnessed by a finite subgraph of G_e. The algorithm to check if there exist such

$T_u^{s,s'}$ and G_e can be seen as an emptiness test for tree automaton, or as a variant of type elimination.

We first define *types*, which can also be viewed as states of a tree automaton. We assign to each node of the tree a type that records the following information:

- a subset of $\Gamma_p \cup \Gamma_{\mathcal{T}}$, representing the labels of the current node;
- an element of $\Sigma_p^{\pm} \cup \Sigma_{\mathcal{T}}^{\pm}$ and a subset of $\Gamma_p \cup \Gamma_{\mathcal{T}}$, representing the label on the edge to the parent and the parent's label set;
- with ℓ the number of at-most restrictions in $\mathcal{T}$, a list of $t \leq \ell+1$ elements of $\Sigma_p^{\pm} \cup \Sigma_{\mathcal{T}}^{\pm}$ and subsets of $\Gamma_p \cup \Gamma_{\mathcal{T}}$, representing labels on the edges to t children of the current node and the children's label sets;
- a Boolean flag indicating whether the current node belongs to the special path (not used for $T_u^{s,s'}$ at all);
- a subset of $\bigcup_{\varphi \in \Phi} K_\varphi \times K_\varphi \times \{\curvearrowright, \curvearrowleft, \downarrow, \uparrow\}$ recording the progress on witnessing δ_{edge} or δ_{node}:
  - $(s, s', \curvearrowright)$ indicates that from state s in the current node we can navigate the current subtree and return to the current node in state s',
  - $(s, s', \curvearrowleft)$ indicates that from state s in the current node we can navigate outside of the current subtree and return to the current node in state s',
  - $(s, s', \downarrow)$ indicates that from state s in the current node, we can reach the target node of e in state s',
  - $(s, s', \uparrow)$ indicates that from state s in target node of e we can reach the current node in state s'.

  Actually, all four kinds of triples are required along the special path, but in the remaining nodes we only need the triples of the form $(s, s', \curvearrowright)$.

By a *pre-type* we shall understand a type with the boolean flag and the progress information dropped; that is, a tuple

$$(\Lambda, R', \Lambda', R_1, \Lambda_1, \dots, R_t, \Lambda_t)$$

with $\Lambda, \Lambda', \Lambda_1, \dots, \Lambda_t \subseteq \Gamma_p \cup \Gamma_{\mathcal{T}}$, and $R', R_1, \dots, R_t \in \Sigma_p^{\pm} \cup \Sigma_{\mathcal{T}}^{\pm}$, and $0 \leq t \leq \ell + 1$. In what follows we blur the distinction between conjunctions K of concept names and sets Λ of labels, as usual, and write $K \subseteq \Lambda$.

A pre-type $(\Lambda, R', \Lambda', R_1, \Lambda_1, \dots, R_t, \Lambda_t)$ is *compatible* with $\mathcal{T}$ iff there exists a graph G such that

- there are pairwise different nodes $u, u', u_1, \dots, u_t$ with label sets $\Lambda, \Lambda', \Lambda_1, \dots, \Lambda_t$;
- there is an R'-edge from u to u' and an R_i-edge from u to u_i for all $i \leq t$, and no other edges are incident with u';
- for each $K \sqsubseteq \exists^{\leq 1} R.K'$ in $\mathcal{T}$ with $K \subseteq \Lambda$, every R-successor of u that satisfies K' belongs to $\{u', u_1, \dots, u_t\}$; and
- G satisfies $\mathcal{T}$ except that CIs of the form $K \sqsubseteq \exists R.K'$ are not required to be satisfied for u'.

Note that unlike in the notion of satisfiability used in Appendix D, the witnessing nodes cannot have additional labels, not listed in $\Lambda, \Lambda', \Lambda_1, \dots, \Lambda_t$.

Lemma E.5. *Given $\mathcal{T}$ and p one can compute the set of pre-types compatible with $\mathcal{T}$ within the time bound stated in Theorem E.3*

Proof. Each pre-type $(\Lambda, R', \Lambda', R_1, \Lambda_1, \dots, R_t, \Lambda_t)$ can be interpreted as a star-shaped graph consisting of nodes $u, u', u_1, \dots, u_n$ with label sets $\Lambda, \Lambda', \Lambda_1, \dots, \Lambda_t$ such that u' is an R'-successor of u, u_i is an R_i-successor of u for all $i \leq t$, and there are no other edges. Thus we can speak of a pre-type satisfying a concept inclusion, etc.

We say a pre-type $(\Lambda, R', \Lambda', R_1, \Lambda_1, \dots, R_t, \Lambda_t)$ is *repeatable* if there is no at-most restriction $K \sqsubseteq \exists^{\leq 1} R.K'$ in $\mathcal{T}$ such that $K \subseteq \Lambda'$, $R = (R')^-$, and $K' \subseteq \Lambda$.

A pre-type $(\Lambda, R', \Lambda', R_1, \Lambda_1, \dots, R_t, \Lambda_t)$ is said to be *compatible with $\mathcal{T}$ modulo* a set Θ of pre-types if

- Θ contains a pre-type $(\Lambda_i, R_i^-, \Lambda, \dots)$ for each $i \leq t$;
- the pre-type satisfies all CIs in $\mathcal{T}$ not of the form $K \sqsubseteq \exists R.K'$;
- for each concept inclusion $K \sqsubseteq \exists R.K'$ in $\mathcal{T}$ with $K \subseteq \Lambda$, at least one of the following holds:
  - $R = R'$ and $K' \subseteq \Lambda'$, or
  - $R = R_i$ and $K' \subseteq \Lambda_i$ for some $1 \leq i \leq t$, or
  - $R = R_0$ and $K' \subseteq \Lambda_0$ for some repeatable $(\Lambda_0, R_0^-, \Lambda, \dots)$ from Θ.

Now, to compute the set of pre-types compatible with $\mathcal{T}$, we start with the set $\Theta = \Theta_0$ of all pre-types, and exhaustively remove those pre-types that are not compatible with $\mathcal{T}$ modulo Θ. This algorithm terminates after at most

$$|\Theta_0| = \sum_{t=0}^{\ell+1} \left|\Sigma_p^{\pm} \cup \Sigma_{\mathcal{T}}^{\pm}\right|^{t+1} \cdot \left(2^{|\Gamma_p \cup \Gamma_{\mathcal{T}}|}\right)^{t+2}$$

iterations. Each iteration takes time polynomial in $|\Theta|$ and $|\mathcal{T}|$.

The result is the maximum set Θ of pre-types such that each pre-type from Θ is compatible with $\mathcal{T}$ modulo Θ. Each pre-type compatible with $\mathcal{T}$ will belong to this set, because the graph witnessing the triple can be used to argue that the triple will not be removed at any iteration. Conversely, each triple from Θ is compatible with $\mathcal{T}$, because one can construct a witnessing tree-shaped graph top-down, using the witnesses justifying the presence of pre-types in Θ in the last iteration of the algorithm. □

Lemma E.6. *The existence of a graph implementing a given annotated skeleton is decidable within the time bound from Theorem E.3.*

Proof. We call a type $(\Lambda, R', \Lambda', R_1, \Lambda_1, \dots, R_t, \Lambda_t, b, \Delta)$ *compatible with $\mathcal{T}$* if the underlying pre-type $(\Lambda, R', \Lambda', R_1, \Lambda_1, \dots, R_t, \Lambda_t)$ is compatible with $\mathcal{T}$. *Repeatable types* are defined analogously, based on the underlying pre-types. Clearly, Lemma E.5 suffices to precompute the set of types compatible with $\mathcal{T}$. Our task is to check if from these types one can construct the witnessing G_e and $T_u^{s,s'}$. We will build them bottom-up, guaranteeing that each promise related to p is fulfilled in a finite fragment.

A type $(\Lambda, R', \Lambda', R_1, \Lambda_1, \dots, R_t, \Lambda_t, b, \Delta)$ is *compatible with p modulo* a set Θ of types if there exists types $(\Lambda_i, R_i^-, \Lambda, \dots, b_i, \Delta_i) \in \Theta$ for $1 \leq i \leq t$ such that

- if $b = 0$, then $b_i = 0$ for all $1 \leq i \leq t$, else $t \geq 1$, $b_1 = 1$, and $b_i = 0$ for all $1 < i \leq t$;
- for each $(s, s', \curvearrowright) \in \Delta$,
  - $(s, A, s') \in \delta$ for some $A \in \Lambda$, or
  - $(s, R_i, s_1) \in \delta$, $(s_1, s_2, \curvearrowright) \in \Delta_i^*$, and $(s_2, R_i^-, s') \in \delta$ for some s_1, s_2 and $1 \leq i \leq t$, or
  - $(s, R_0, s_1) \in \delta$, $(s_1, s_2, \curvearrowright) \in \Delta_0^*$, and $(s_2, R_0^-, s') \in \delta$ for some s_1, s_2 and repeatable $(\Lambda_0, R_0^-, \Lambda, \dots, 0, \Delta_0) \in \Theta$,

where Δ_i^* is the set of all $(s, s', \curvearrowright)$ such that there are states $s = s_1, s_2, \ldots, s_m = s'$ with $(s_j, s_{j+1}, \curvearrowright) \in \Delta_i$ for all $j < m$;
- if $b = 1$, then for each $(s, s', \uparrow) \in \Delta$, there are s_1, s_2 such that $(s, s_1, \uparrow) \in \Delta_1$, $(s_1, s_2, \curvearrowright) \in \Delta_1^*$, and $(s_2, R_1^-, s') \in \delta$;
- if $b = 1$, then for each $(s, s', \downarrow) \in \Delta$, there are s_1, s_2 such that $(s, R_1, s_1) \in \delta$, $(s_1, s_2, \curvearrowright) \in \Delta_1^*$, and $(s_2, s', \downarrow) \in \Delta_1$;
- if $b = 1$, then for each $(s, s', \curvearrowright) \in \Delta_1$, there are $s_1, \ldots, s_m$ such that $(s, R_1^-, s_1), (s_m, R_1, s') \in \delta$ and for all $j < m$, either $(s_j, s_{j+1}, \curvearrowright) \in \Delta_2^* \cup \cdots \cup \Delta_m^*$, or $(s_j, s_{j+1}, \curvearrowright) \in \Delta$, or $(s_j, A, S_{j+1}) \in \delta$ for some $A \in \Lambda$.

Let us first see how to decide the existence of G_e for a given symbolic edge e. The algorithm begins with the set Θ of all "initial types", which are

- types $(\Lambda, R', \Lambda', b, \Delta)$ such that
  - Λ is the label set of the target of e,
  - $(R', \Lambda') = \beta_{\mathrm{src}}(e)$,
  - $b = 1$,
  - Δ consists of all $(s, s', \curvearrowright)$ such that $(s, s', \overset{\curvearrowright}{\in}) \in \delta_{\mathrm{edge}}(e)$, as well as all $(s, s, \uparrow)$ and $(s, s, \downarrow)$;
- types $(\Lambda, R', \Lambda', \ldots, b, \Delta)$ compatible with $\mathcal{T}$ such that
  - $b = 0$,
  - $\Delta = \emptyset$.

Then, we exhaustively extend Θ with types that are compatible with $\mathcal{T}$ and compatible with p modulo Θ. When no more types can be added, the graph G_e exists iff Θ contains a type $(\Lambda, R', \Lambda', \ldots, b, \Delta)$ such that

- Λ is the label set of the source of the symbolic edge e;
- $\big((R')^-, \Lambda\big) = \beta_{\mathrm{src}}(e)$;
- $b = 1$;
- Δ contains no triples of the form $(s, s', \curvearrowright)$;
- for each $(s, s', \curvearrowright) \in \delta_{\mathrm{edge}}(e)$ there are states s_1, s_2 such that $\big(s, (R')^-, s_1\big) \in \delta$, $(s_1, s_2, \curvearrowright) \in \Delta^*$, and $\big(s_2, (R')^-, s'\big) \in \delta$;
- for each $(s, s', \downarrow) \in \delta_{\mathrm{edge}}(e)$ there are states s_1, s_2 such that $\big(s, (R')^-, s_1\big) \in \delta$, $(s_1, s_2, \curvearrowright) \in \Delta^*$, and $(s_2, s', \downarrow) \in \Delta$;
- for each $(s, s', \uparrow) \in \delta_{\mathrm{edge}}(e)$ there are states s_1, s_2 such that $(s, s_1, \uparrow) \in \Delta$, $(s_1, s_2, \curvearrowright) \in \Delta^*$, and $(s_2, R', s') \in \delta$.

This number of iterations of the algorithm is bounded by the number of all types,

$$\sum_{t=0}^{\ell+1} \left|\Sigma_p^{\pm} \cup \Sigma_{\mathcal{T}}^{\pm}\right|^{t+1} \cdot \left(2^{|\Gamma_p \cup \Gamma_{\mathcal{T}}|}\right)^{t+2} \cdot 2 \cdot \left(2^{\left|\bigcup_{\varphi \in \Phi} K_\varphi \times K_\varphi \times \{\curvearrowright, \curvearrowright, \downarrow, \uparrow\}\right|}\right)^t.$$

Each iteration takes time polynomial in $|\Theta|^\ell$ and $|\mathcal{T}|$. The promised complexity bounds follow.

Deciding the existence of the witnessing trees for a node u of the annotated skeleton is very similar. We can reuse the set Θ computed for any symbolic edge e. The only delicate issue is that we need to account for $\beta_{\mathrm{src}}(e')$ for all edges e' outgoing from u and $\beta_{\mathrm{tgt}}(e'')$ for all edges e'' incoming to u. Essentially, we check if there exists a type $(\Lambda, R_1, \Lambda_1, \ldots, R_t, b, \Delta)$ – note the missing R' and Λ' – with $b = 0$ and $t \le \ell + \deg(u)$, compatible with $\mathcal{T}$ and compatible with p modulo Θ, except that for $i = 1, 2, \ldots, \deg(u)$, the components R_i, Λ_i must be as specified by $\beta_{\mathrm{src}}(e')$ and $\beta_{\mathrm{tgt}}(e'')$ for outgoing e' and incoming e'', and their corresponding types must be $(\Lambda_i, R_i^-, \Lambda, 0, \emptyset)$, not required to belong to Θ. This can be done in time polynomial in $|\Theta|^\ell$, $\mathcal{T}$, and $\mathcal{H}$. □

Corollary E.7. *Unrestricted entailment of concept inclusions by an $\mathcal{ALCIF}$ TBox $\mathcal{T}$ using k concept names and ℓ at-most constraints can be decided in time $O\big(\mathrm{poly}(|\mathcal{T}|) \cdot 2^{\mathrm{poly}(k,\ell)}\big)$.*

Proof. The result holds in full generality, but we only sketch the arguments for the two kinds of concept inclusions we need to compute the completion. For existential constraints, note that

$$\mathcal{T} \models A_1 \sqcap \cdots \sqcap A_n \sqsubseteq \exists R.K'$$

iff the query

$$\exists x.(A_1 \cdot \ldots \cdot A_n \cdot B)(x, x)$$

is unsatisfiable modulo the TBox

$$\mathcal{T} \cup \left\{K' \sqsubseteq \forall R^-.B',\ B \sqcap B' \sqsubseteq \bot\right\},$$

where B and B' are fresh concept names. For at-most constraints,

$$\mathcal{T} \models A_1 \sqcap \cdots \sqcap A_n \sqsubseteq \exists^{\le 1} R.A_1' \sqcap \cdots \sqcap A_m'$$

iff the query

$$\exists x, y, z.(A_1 \cdot \ldots \cdot A_n)(x, x) \wedge (R \cdot A_1' \cdot \ldots \cdot A_m' \cdot B)(x, y) \wedge \wedge (R \cdot A_1' \cdot \ldots \cdot A_m' \cdot B')(x, z)$$

is unsatisfiable modulo the TBox

$$\mathcal{T} \sqcup \left\{B \sqcap B' \sqsubseteq \bot\right\}$$

where B and B' are fresh concept names. □

F PROOF OF HARDNESS

Theorem F.1. *Testing containment of Boolean 2RPQs modulo schema is* EXPTIME*-hard.*

We present a reduction of the acceptance problem of an alternating Turing machine with a polynomial bound on space. We begin by defining a special variant of alternating Turing machines. We also present a number of conceptual tools used in the reduction.

Alternating Turing machines. We consider a variant of alternating Turing machine with the following particularities:

- there is a single distinguished initial state that the machine never reenters;
- there are two special states q_{yes} and q_{no} that are final (no transition allowed to follow)[1];
- the transition table has exactly two transitions for any non-final state and any symbol;
- there exists 3 special symbols: □ for empty tape space, ▷ for left tape boundary, and ◁ for right tape boundary; we only assume that the input word does not use those symbols and the transition table handles the boundary symbols appropriately.

It's relatively easy to see that any alternating Turing machine with polynomially bounded space can converted to the variant above.

Formally, an *alternating Turing machine* (ATM) is a tuple $M = (A, K, q_0, \delta_1, \delta_2)$, where A is a finite alphabet, K is a finite set of states with two distinguished final states q_{yes} and q_{no} and partitioned into three pair-wise disjoint subsets $K = K_\forall \cup K_\exists \cup \{q_{yes}, q_{no}\}$, $q_0 \in K$ is a distinguished initial state, and $\delta_i : (K \setminus \{q_{yes}, q_{no}\}) \times$

[1] The state q_{no} is not necessary for the purposes of our reduction but we include it for the sake of completeness of this variant of ATM

$A \to (K \setminus \{q_0\}) \times A \times \{-1, +1\}$ are two transition tables such that $\delta_i(q, x) = (q', y, d)$ satisfies the following two conditions:

(1) if $x = \triangleright$, then $y = \triangleright$ and $d = +1$ and
(2) if $x = \triangleleft$, then $y = \triangleleft$ and $d = -1$.

A *configuration* of M is a string of the form $\triangleright \cdot w \cdot q \cdot v \cdot \triangleleft$, where $q \in K$ and $w, v \in \Sigma^*$ Applying a transition $(q', z, d) \in K \times A \times \{-1, +1\}$ to the configuration $\triangleright \cdot w \cdot x \cdot q \cdot y \cdot v \cdot \triangleleft$ yields:

(1) $\triangleright \cdot w \cdot q' \cdot x \cdot z \cdot v \cdot \triangleleft$ if $d = -1$
(2) $\triangleright \cdot w \cdot x \cdot z \cdot q' \cdot v \cdot \triangleleft$ if $d = +1$

We consider ATMs with polynomially bounded space, a class of Turing machines that defines the class ASPACE known to coincide with EXPTIME. Recall that a binary tree is a finite prefix-closed subset $T \subseteq \{1, 2\}^*$ and a labeled-tree is a function λ that assigns a label to every element (node) of a tree.

Given an ATM M and a polynomial $poly(n)$, a *run* of M w.r.t. $poly$ on an input $w \in (\Sigma \setminus \{\triangleright, \triangleleft, \square\})^*$ is a binary tree λ whose nodes are labeled with configurations of M such that:

(1) the root node is labeled with $\lambda(\varepsilon) = \triangleright \cdot q_0 \cdot w \cdot \square^{poly(|w|)-|w|} \cdot \triangleleft$
(2) for non-leaf node $n \in dom(\lambda)$ let $\lambda(n) = \triangleright \cdot w \cdot q \cdot x \cdot v \cdot \triangleleft$; for every $i \in \{1, 2\}$ if n has a child $n \cdot i$, then the configuration $\lambda(n \cdot i)$ is obtained by applying the transition $\delta_i(q, x)$ to the configuration $\lambda(n)$. Also, if $q \in K_\forall$, then n has both children $n \cdot 1$ and $n \cdot 2$ and if $q \in K_\exists$, then n has precisely one child,
(3) for every leaf node $n \in dom(\lambda)$ the configuration $\lambda(n)$ uses a final state q_{yes} or q_{no}.

A run is *accepting* if and only if all its leaves use the state q_{yes}. The ATM M (with space bound $poly$) accepts a word w, in symbols $M(w) = yes$ if and only if there is an accepting run of M w.r.t. $poly$ on w.

Reduction outline. We present a reduction of the problem of word acceptance by an ATM with polynomial bound on space to the complement of the problem of containment of Boolean 2RPQs in the presence of schema. We point out that the class of ASPACE-complete problems is closed under complement, and consequently, this reduction proves that the query containment problem is EXPTIME-hard.

More precisely, for an ATM M, whose space is bounded by $poly(n)$, and an input word w we construct a schema S and two Boolean 2RPQs p and q such that

$$M(w) = yes \quad iff \quad p \not\subseteq_S q \quad iff \quad \exists G \in L(S).\, G \models p \wedge G \not\models q\,.$$

In the sequel, we refer to p as the positive query and to q as the negative query. Naturally, we present a reduction that is polynomial i.e., the combined size of p, q, and S is bounded by polynomial in the size of M and w.

The reduction constructs a schema S and queries p and q for which the counter-example of $p \subseteq_S q$ represents an accepting run of M on w. Before we present the reduction in detail, we introduce 3 conceptual devices that we use in the reduction: nesting queries, encoding disjunction, and enforcing tree structure.

Nesting queries. The reduction employs a relatively large and complex queries and throughout the reduction we employ *nesting of regular path queries* that is expanded as follows:

$$p[q] = p \cdot q \cdot q^-$$

with the inverse operator being extended to regular path queries in the standard fashion.

$$\varnothing^- = \varnothing\,, \qquad \epsilon^- = \epsilon\,, \qquad A^- = A\,,$$
$$(\varphi_1 \cdot \varphi_2)^- = \varphi_2^- \cdot \varphi_1^-\,, \quad (\varphi_1 + \varphi_2)^- = \varphi_1^- + \varphi_2^-\,, \quad (\varphi^*)^- = (\varphi^-)^*\,.$$

We point out that, in general, this definition is not equivalent to the standard meaning of nesting of regular expressions but in our reduction nested queries are evaluated at nodes for which the schema ensures the intended meaning.

Encoding disjunction. The first conceptual device allows us to express disjunction in schemas, which we illustrate on the following example. Take two node labels A and B and suppose we wish to require A-nodes to have either one outgoing a-edge or one outgoing b-edge to a node with label B. The schema formalism allows us to make the following restriction.

$$A \to a : B^?, b : B^?\,.$$

Alone, it is insufficient as it allows nodes that do not fulfill the disjunctive requirement: a A-node that has no outgoing edge or has both outgoing edges. We remove those cases with the help of a positive and a negative query. Namely, we define

$$p = A[(a + b)] \qquad \text{and} \qquad q = A[a][b]$$

and we observe that in a graph that conforms to the above schema any node with label A that satisfies p and does not satisfy q has precisely one outgoing edge.

Enforcing tree structure. In our reduction we aim at constructing a tree-shaped counter examples and we use the positive query to diligently enforce disjunction in every node. In essence, the positive query will traverse the counter-example and impose satisfaction of a relevant query in every node. We present this device on an example where we define rooted binary trees. The general shape of the tree follows the schema in Figure 6.

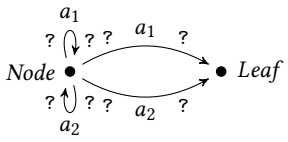

Figure 6: Example schema for modeling trees.

Naturally, the schema alone is insufficient to capture the right structure of the tree. Consequently, additional requirements are imposed with the help of the following negative Boolean query

$$q = Node[a_1 \cdot Node][a_1 \cdot Leaf] + Node[a_2 \cdot Node][a_2 \cdot Leaf] + [a_1^-][a_2^-]$$

that ensures that an inner node does not have two outgoing edges with the same label and that no node has two incoming edges. We point out that when q is not satisfied at a node, schema ensures that it has at most one incoming edge. To enforce the correct tree structure we define the following unary query

$$p_{Tree}(x) = \big(((Node[a_1][a_2] \cdot a_1)^* \cdot Leaf \cdot (a_2^-)^* \cdot a_1^- \cdot a_2)^* \cdot Leaf \cdot (a_2^-)^*\big)(x, x)\,.$$

The key observation here is that a_1^- is always followed by a_2 and the query can move up the tree only after a leaf has been reached. This ensures a proper traversal of the structure, with every node satisfying the pattern $Node[a_1][a_2]$. Consequently, for any connected graph G that conforms to the above schema, satisfies p, and does not satisfy q, G is a binary tree.

The input of the reduction. We fix an ATM $M = (A, K, q_0, \delta_1, \delta_2)$ whose space is bounded by $poly(n)$ and we fix an input word $w \in (A \setminus \{\triangleright, \triangleleft, \square\})^*$. We let $n = |w|$, $m = poly(|w|)$, and assume that $A = \{a_1, \ldots, a_k\}$ and that $K = \{q_0, q_1, \ldots, q_\ell\}$. Throughout the description of the reduction, unless we say otherwise, we use a, b to range over symbols in A, we use q, p to range over states in K, and we use i, j to range over tape positions $\{1, \ldots, m\}$.

The schema. We construct a schema S whose signature is

$$\Sigma_S = \{Config, Pos, Symb, St\},$$
$$\Gamma_S = \{\forall_1, \forall_2, \exists_1, \exists_2, pos_1, \ldots, pos_m\} \cup \{a_1, \ldots, a_k\} \cup \{q_0, \ldots, q_\ell\}.$$

In essence, *Config*-nodes represent configurations and *Pos*-nodes represent tape cells. The edges labeled with $\{\forall_1, \forall_2, \exists_1, \exists_2\}$ are transition edges that connect configurations. The schema S is presented in Figure 7. We introduce macros that illustrate the intended

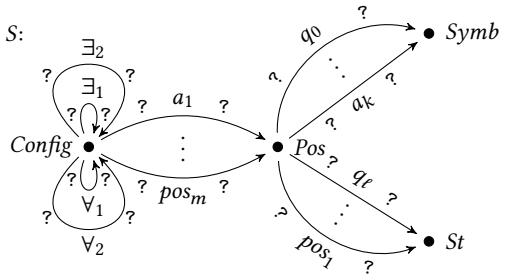

Figure 7: Schema for the reduction.

meaning of the remaining edge labels. The first macro checks that the symbol at position i on the tape is a.

$$Symbol_{i,a} = Config\left[pos_i \cdot a\right].$$

The next one checks that the configuration is a given state q with the head at a given position i.

$$State_{i,q} = Config\left[pos_i \cdot q\right].$$

Finally, we also introduce a macro that asserts the state of a configuration without any constraint on the position of the head.

$$State_q = Config\Big[\bigplus_i pos_i \cdot q\Big].$$

And analogously, a macro that asserts heads position only

$$Head_i = Config\Big[\bigplus_q pos_i \cdot q\Big].$$

The negative query. We define a number of queries that detect violations of good structure of a run; their union will be used as the negative query. First, we identify configurations that has two different symbols at a position of the tape.

$$q_{TwoSymbols} = Config\left[\bigplus_i \bigplus_{a\neq b} Symbol_{i,a} \cdot Symbol_{i,b}\right].$$

Similarly, we identify configurations with two different heads.

$$q_{TwoHeads} = Config\left[\bigplus_{i\neq j \vee p\neq q} State_{i,q} \cdot State_{j,p}\right].$$

Next, we identify configurations with outgoing transition edges that do not fit their state.

$$q_{BadTransitionEdges} = Config\begin{bmatrix} \bigplus_{q\in K_\forall} State_q[\exists_1 + \exists_2] + \\ \bigplus_{q\in K_\exists} State_q[\forall_1 + \forall_2] + \\ State_{q_{yes}}[\forall_1 + \forall_2 + \exists_1 + \exists_2] + \\ State_{q_{no}}[\forall_1 + \forall_2 + \exists_1 + \exists_2] \end{bmatrix}.$$

Additionally, identify configurations with existential states that have both existential outgoing edges (the definition of a run requires precisely one).

$$q_{TwoExistentialEdges} = \bigplus_{q\in K_\exists} State_q[\exists_1][\exists_2].$$

The initial configuration, which is the only configuration with state q_0, should be the root of the run and as such it should not have any incoming transition edges.

$$q_{BadTreeRoot} = State_{q_0}[\exists_1^- + \exists_2^- + \forall_1^- + \forall_2^-].$$

To make sure that the run is a tree, no configuration should have two incoming transitions (note that the schema forbids more than one incoming edge with the same label).

$$q_{BadTreeNode} = Config\begin{bmatrix} [\exists_1^-][\exists_2^-] + [\exists_1^-][\forall_1^-] + [\exists_1^-][\forall_2^-] + \\ [\exists_2^-][\forall_1^-] + [\exists_2^-][\forall_2^-] + [\forall_1^-][\forall_2^-] \end{bmatrix}.$$

Similar requirements apply to tape: we do not allow tape positions that are used by two different configurations.

$$q_{BadTape} = \bigplus_{i\neq j} Pos[pos_i^-][pos_j^-] + \bigplus_{p\neq q} St[p^-][q^-] + \bigplus_{a\neq b} Symb[a^-][b^-].$$

Finally, we construct the union of the above queries.

$$q_M = q_{TwoSymbols} + q_{TwoHeads} + q_{BadTransitionEdges} + q_{TwoExistentialEdges} + q_{BadTreeRoot} + q_{BadTreeNode} + q_{BadTape}.$$

The positive query. We first construct a query that ensures that a configuration is valid and then we design a path query that traverses the tree and ensures that each of its configurations is valid. A valid configuration satisfies the following queries. It has a head at some position.

$$p_{Head} = Config\left[\bigplus_i Head_i\right].$$

Every position has a symbol.

$$p_{Tape} = Config\left[\bigplus_a Symbol_{1,a}\right] \ldots \left[\bigplus_a Symbol_{m,a}\right].$$

The configuration has the required outgoing transitions and only final states are accepted in leaves.

$$p_{Transition} = Config\begin{bmatrix} \bigplus_{q\in K_\forall} State_q[\forall_1][\forall_2] + \\ \bigplus_{q\in K_\exists} State_q[\exists_1 + \exists_2] + \\ State_{q_{yes}} + State_{q_{no}} \end{bmatrix}.$$

The positive query ensuring that transitions are executed properly is more difficult to define and we decompose it into several macros. First, we define a macro $Move_{i,q,a}$ that verifies that that a configuration in state q at position i with symbol $a \in \Sigma$ has the right

children configurations. We define this macro depending on the type of state:

(1) For $q \in \{q_{yes}, q_{no}\}$ no children are necessary (the negative query $q_{BadTransitionsEdges}$ forbids any)

$$Move_{i,q,a} = State_q \cdot Symbol_{i,a} .$$

(2) For $q \in K_\exists$ we check that one of the transitions is implemented (the negative query $q_{TwoExistentialEdges}$ forbids more than one). We let $\delta_1(q,a) = (q_1, b_1, d_1)$ and $\delta_2(q,a) = (q_2, b_2, d_2)$.

$$\begin{aligned} Move_{i,q,a} = {} & [State_{i,q} \cdot Symbol_{i,a} \cdot \exists_1 \cdot State_{i+d_1,q_1} \cdot Symbol_{i,b_1}] \\ & + [State_{i,q} \cdot Symbol_{i,a} \cdot \exists_2 \cdot State_{i+d_2,q_2} \cdot Symbol_{i,b_2}] . \end{aligned}$$

(3) For $q \in K_\forall$ both transitions must be implemented. Again we let $\delta_1(q,a) = (q_1, b_1, d_1)$ and $\delta_2(q,a) = (q_2, b_2, d_2)$.

$$\begin{aligned} Move_{i,q,a} = {} & [State_{i,q} \cdot Symbol_{i,a} \cdot \forall_1 \cdot State_{i+d_1,q_1} \cdot Symbol_{i,b_1}] \\ & \cdot [State_{i,q} \cdot Symbol_{i,a} \cdot \forall_2 \cdot State_{i+d_2,q_2} \cdot Symbol_{i,b_2}] . \end{aligned}$$

Now, a transition is executed correctly if the following positive query holds at a configuration node.

$$p_{Execution} = Config \left[\textstyle\sum_{i,q,a} Move_{i,q,a} \right] .$$

To handle the tape we need to make sure that 1) the tape of the initial configuration contains precisely the input word and 2) that symbols at the positions without head are copied correctly. For the first, we define the following macro.

$$InitTape = Symbol_{1,w_1} \cdot \ldots Symbol_{n,w_n} \cdot Symbol_{n+1,\square} \cdot \ldots \cdot Symbol_{m,\square} .$$

The next macro verifies that the symbol at a position i of the tape is a correct copy of its preceding configuration.

$$PosCopy_i = \left[\textstyle\sum_a Symbol_{i,a} (\exists_1 + \exists_2 + \forall_1 + \forall_2)^- Symbol_{i,a} \right] .$$

Naturally, when the head in the preceding configuration is at position i, then we must only verify that symbols at positions other than i are copied.

$$\begin{aligned} TapeCopy = \textstyle\sum_i \big(& [(\exists_1 + \exists_2 + \forall_1 + \forall_2)^- Head_i] \cdot \\ & PosCopy_1 \cdot \ldots \cdot PosCopy_{i-1} \cdot \\ & PosCopy_{i+1} \cdot \ldots \cdot PosCopy_m \big) . \end{aligned}$$

Finally, the query that verifies the correctness of the tape follows.

$$p_{TapeCopy} = Config \left[State_{1,q_0} \cdot InitTape + TapeCopy \right] .$$

Now, we take the conjunction of the queries that verify local correctness of a configuration.

$$p_{Config} = p_{Head} \cdot p_{Tape} \cdot p_{Transition} \cdot p_{Execution} \cdot p_{TapeCopy} .$$

Additionally, we define a configuration that is a leaf (accepting)

$$p_{Accept} = p_{Config} \cdot State_{q_{yes}} .$$

And, the initial configuration

$$p_{Start} = p_{Config} \cdot State_{q_0} .$$

Finally, we define the positive query, based on the ideas of enforcing tree structure in p_{Tree}. It traverses the counter-example and ensures that it contains only good configurations.

$$\begin{aligned} p_{M,w} = {} & p_{Start} \cdot \\ & \big((p_{Config} \cdot (\forall_1 + \exists_1 + \exists_2))^* \cdot p_{Accept} \cdot (\exists_1^- + \exists_2^- + \forall_2^-)^* \cdot \forall_1^- \cdot \forall_2 \big)^* \cdot \\ & (p_{Config} \cdot (\forall_1 + \exists_1 + \exists_2))^* \cdot p_{Accept} \cdot (\exists_1^- + \exists_2^- + \forall_2^-)^* \cdot p_{Start} . \end{aligned}$$

Before stating the main proof we present in Figure 8 a conceptual automaton that corresponds to the above Boolean 2RPQ. In the

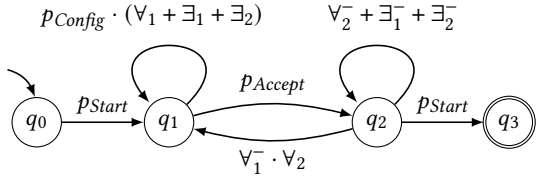

Figure 8: Conceptual automaton of the positive query $p_{M,w}$.

proof below, we refer to $p_{i,j}$ as the query defined with the above automaton whose initial state is q_i and final state is q_j. The main claim follows.

Claim. *$p_{M,w} \not\subseteq_S q_M$ if and only if $M(w) = yes$.*

Proof. For the *if* direction, we take the accepting run λ and construct the corresponding graph G as follows. The nodes and their labels are as follows.

$$\begin{aligned} Config^G &= \{c_n \mid n \in dom(\lambda)\}, \\ Pos^G &= \{t_{n,i} \mid n \in dom(\lambda), 1 \le i \le M\}, \\ St^G &= \{s_n \mid n \in dom(\lambda)\}, \\ Symb^G &= \{e_{n,i} \mid n \in dom(\lambda), 1 \le i \le M\} . \end{aligned}$$

The edges of G are:

(1) $(c_n, pos_i, t_{n,i})$ for every $n \in dom(\lambda)$ and $i \in \{1, \ldots, M\}$,
(2) $(t_{n,i}, q, s_n)$ for every $n \in dom(\lambda)$ where q is the state of configuration $\lambda(n)$;
(3) $(t_{n,i}, a, e_{i,n})$ for every $n \in dom(\lambda)$ and $i \in \{1, \ldots, M\}$ where a is the symbol at position i of the tape of configuration $\lambda(n)$;
(4) $(c_n, \forall_1, c_{n\cdot 1})$ and $(c_n, \forall_2, c_{n\cdot 2})$ for every $n \in dom(\lambda)$ such that the configuration $\lambda(n)$ is at state $q \in K_\forall$;
(5) $(c_n, \exists_j, c_{n\cdot j})$ for every $n \in dom(\lambda)$ such that the configuration $\lambda(n)$ is at state $q \in K_\exists$ and n has a child $n \cdot j$ in λ for some $j \in \{1, 2\}$.

It is easy to show that G satisfies the schema S, does not satisfy q, all *Config*-nodes satisfy p_{Config}, the root node satisfies p_{Start} and every leaf node satisfies p_{Accept}.

With a simple induction, on the height of a node $n \in dom(\lambda)$, we prove that for any $n \in dom(\lambda)$ the node c_n satisfies the query $p_{1,2}$. This shows that the root node c_ε satisfies the query $p_{0,3} = p$.

For the *only if* direction, we take any G that satisfies S, satisfies p, and does not satisfy q. W.l.o.g. we can assume that G is connected; otherwise we take any connected component that satisfies p. We show that G is a tree encoding an accepting run of M on w. Note that q is a Boolean RPQ, and thus a single two-way regular expression. Thus, in the sequel we analyze its witnessing paths in G but p should not be confused with a binary query; a Boolean RPQ ask the existence of a witnessing path without the need to report its endings.

Take any pair of nodes u_0 and v_0 such that there is a path from u_0 to v_0 that witnesses q (which is a regular expression). Since G

does not have a node with two incoming edges ($q_{BadTreeNode}$ and $q_{BadTape}$ are not satisfied at any node), u_0 and v_0 are the same node. Consequently there is a path from u_0 to u_0 that witnesses $p_{1,2}$ and we show with an induction on the length of the path from u_0 to any reachable *Config*-node v that there is a path form v to v that witnesses $p_{1,2}$, and consequently, v satisfies p_{Config}. This implies that G has the form of a tree, all of its *Config*-nodes satisfy p_{Config} and all its leaves satisfy p_{Accept}. Moreover, we can construct an accepting run λ from G that shows that $M(w) = yes$. □

Finally, we observe that the sizes of S, p, and q are polynomial in the size of M and w, which proves the main claim.
The hardness of containment in the presence of schema implies hardness of the static analysis problems we study.

Lemma F.2. *Type checking, equivalence, and schema elicitation are* EXPTIME*-hard.*

Proof. We reduce the containment of unary 2RPQs in the presence of schema to the problems of interest. Note that by Theorem F.1 and Corollary D.2, containment of unary acyclic 2RPQs is EXPTIME-hard. We take any schema S and two unary 2RPQs $p(x)$ and $q(x)$. In all reductions S is the input schema and we assume a single unary constructor $\mathcal{F} = \{f_A\}$.

We begin by showing that testing $(T, S) \models \bigcap \Gamma_T$ is EXPTIME-hard. We take the transformation T defined with the following rules.

$$A(f_A(x)) \leftarrow q(x) \quad \text{and} \quad a(f_A(x), f_A(x)) \leftarrow p(x)\,.$$

We observe that $(T, S) \models \bigcap \Gamma_T$ if and only if $p(x) \subseteq_S q(x)$.

For equivalence, we define the following two transformations.

$$T_1 : A(f_A(x)) \leftarrow q(x)\,.$$
$$T_2 : A(f_A(x)) \leftarrow q(x)\,, \quad A(f_A(x)) \leftarrow p(x)\,.$$

We observe that $T_1 \equiv_S T_2$ if and only if $p(x) \subseteq_S q(x)$.

For type checking we define the following transformation and output schema

$$T : A(f_A(x)) \leftarrow p(x)\,, \qquad A(f_A(x)) \leftarrow q(x)\,,$$
$$a(f_A(x), f_A(x)) \leftarrow q(x)\,.$$
$$S' : A \to a : A^1\,.$$

We observe that that $T(S) \subseteq S'$ if and only if $p(x) \subseteq_S q(x)$.

To prove that schema elicitation is also EXPTIME-hard, we take the previous transformation T, the input schema S, and show that $p(x) \subseteq_S q(x)$ if and only if the $\subseteq$-minimal schema that captures the output graphs is precisely S'. We observe that deciding equivalence of two schemas is easily accomplished in polynomial time and therefore any algorithm for schema elicitation must require exponential time. □