How to Best Nest Regular Path Queries

Pierre Bourhis[1,2], Markus Krötzsch[3], and Sebastian Rudolph[3]

[1] CNRS LIFL University of Lille 1, France
[2] INRIA Lille Nord Europe, France
[3] TU Dresden, Germany

Abstract. Regular path queries (RPQs) define query patterns in terms of regular expressions and are therefore well-suited to query for paths over roles in DL. RPQs can be extended to 2-way RPQs (with converse), CRPQs (with conjunctions), or PRPQs (arbitrary positive Boolean combinations), all of which have been explored in DL research. Another natural extension of any query language is nesting, where query predicates can be defined in terms of subqueries. In this paper, we discuss several ways of introducing nesting to PRPQs, and show that they lead to increasingly expressive query languages: CN2RPQs, which were studied in the context of DLs recently; nested P2RPQs; and positive queries with transitive closure on binary predicates. The latter is one of the most expressive languages for which query answering can still be decided over DL knowledge bases. We present initial complexity results that show query answering to be non-elementary in the worst case, with an exponential increase for each level of nesting of the transitive closure operator.

1 Introduction

Regular path queries (RPQs) are an important query formalism that has influenced many practical query languages, including SPARQL and XPath, and that has also played a prominent role in DL research [4,5,2]. Indeed, the ability of RPQs to navigate along paths in directed graphs makes them a most natural candidate for querying DL knowledge bases.

It is therefore no surprise that the landscape of RPQ-based query languages has been expanding significantly in recent years. The use of regular expressions in query languages was first considered in the 1990's in the context of semi-structured databases [7], but related logical constructs like transitive closure have been studied much earlier [8]. Later extensions introduced inverse roles to obtain two-way regular path queries [3], the general use of conjunction and disjunction [5], and certain *test operators* that were inspired by XPath [10].

A closely related notion of *nested* RPQ*s* has also been studied for DLs very recently [1]. Generally speaking, query nesting is the process of using an n-ary subquery instead of an n-ary predicate symbol within a query, with the obvious semantics. However, the impact of this extension in terms of complexity and expressiveness may vary significantly depending on the query language under consideration and the exact form of nested queries. Nested RPQs employ a form of unary subqueries.

Surprisingly, it seems that the natural extension of conjunctive RPQs with *binary* subqueries has not been studied so far. Another closely related formalism is positive

first-order logic with transitive closure. How do these languages compare in terms of expressiveness? Can any of them be decided over DL knowledge bases? At which complexity? These questions seem to be largely unanswered today.

Our results shed some light on these issues. We study three forms of nested queries: CN2RPQ (nested test expressions), P2RPQ$^+$ (nested binary queries), and PFO+TC1 (transitive closure with one input and output variable) and show these query languages to form a hierarchy of increasing expressiveness. DL query answering is decidable even for the most expressive language PFO+TC1. We establish tight complexity bounds for query answering for various DLs, including $\mathcal{SHIQ}$, $\mathcal{SHOQ}$, and $\mathcal{SHOI}$, showing a multi-exponential behavior that depends on the nesting depth of the query.

2 DLs and Positive Regular Path Queries

Readers who are not familiar with DLs might wish to consult an introductory text first [9]. All DLs we consider are fragments of $\mathcal{SROIQ}$. We assume a fixed signature consisting of sets $\mathbf{N_I}$ of *individual names*, $\mathbf{N_C}$ of *concept names*, and $\mathbf{N_R}$ of *role names*. The set $\mathbf{R}$ of roles is $\{R, R^- \mid R \in \mathbf{N_R}\}$; the set $\mathbf{C}$ of concept expressions depends on the DL considered. In queries, we use variables from a countably infinite set $\mathbf{V}$.

Definition 1. *A* regular expression *over an alphabet Σ is a term constructed from elements of Σ, binary operators $\cdot$ (concatenation) and $|$ (alternative), and the unary operator * (Kleene star). A* two-way regular path query *(*2RPQ*) over a DL signature is of the form $E(s,t)$, with E a regular expression over $\mathbf{R} \cup \mathbf{C}$, and s and t terms in $\mathbf{V} \cup \mathbf{N_I}$.*[4]

A positive Boolean formula *is one that uses only the operators $\wedge$ and $\vee$. A* positive two-way regular path query *(*P2RPQ*) is an expression $\exists \boldsymbol{y}.\varphi[\boldsymbol{x}, \boldsymbol{y}]$, where $\varphi[\boldsymbol{x}, \boldsymbol{y}]$ is a positive Boolean formula over* 2RPQ*s. All variables in φ occur in $\boldsymbol{x} \cup \boldsymbol{y}$, with free variables $\boldsymbol{x}$ disjoint from bound variables $\boldsymbol{y}$. A query without free variables is* Boolean*.*

The semantics of queries for a DL interpretation $\mathcal{I}$ is defined as follows: Roles R describe binary relations $R^\mathcal{I}$, while concepts C describe *binary* relations $\{\langle\delta,\delta\rangle \mid \delta \in C^\mathcal{I}\}$. The semantics of regular expressions is defined inductively as follows: $(E_1 \cdot E_2)^\mathcal{I} := E_1^\mathcal{I} \circ E_2^\mathcal{I}$; $(E_1 \mid E_2)^\mathcal{I} := E_1^\mathcal{I} \cup E_2^\mathcal{I}$; $(E_1^*)^\mathcal{I} := \mathsf{id} \cup E_1^\mathcal{I} \cup (E_1^\mathcal{I} \circ E_1^\mathcal{I}) \cup \ldots$, where $\circ$ is relational composition and id denotes the identity relation on the domain of $\mathcal{I}$.

A variable assignment $\mathcal{Z}$ for $\mathcal{I}$ is a mapping $\mathbf{V} \to \Delta^\mathcal{I}$. For a variable x, we set $x^{\mathcal{I},\mathcal{Z}} := \mathcal{Z}(x)$; for an individual name $c \in \mathbf{N_I}$, we set $c^{\mathcal{I},\mathcal{Z}} := c^\mathcal{I}$. A 2RPQ $E(s,t)$ evaluates to true under $\mathcal{Z}$ and $\mathcal{I}$ if $\langle s^{\mathcal{I},\mathcal{Z}}, t^{\mathcal{I},\mathcal{Z}}\rangle \in E^\mathcal{I}$. A P2RPQ $\exists \boldsymbol{y}.\varphi[\boldsymbol{x}, \boldsymbol{y}]$ is *satisfied* by $\mathcal{Z}$ and $\mathcal{I}$ if there is a variable assignment $\mathcal{Z}'$ that agrees with $\mathcal{Z}$ on all variables other than possibly $\boldsymbol{y}$, such that φ evaluates to true under $\mathcal{I}$ and $\mathcal{Z}'$.

3 Three Ways of Nesting P2RPQs

Next, we consider three different approaches of extending P2RPQs with nesting. Since the query predicates used in P2RPQs are binary, the canonical way to define nested P2RPQs uses binary P2RPQ as subqueries.

[4] Some works use $C?$ instead of C [1]. We avoid this notation since ? is often used to denote the *optional* operator in regular expressions.

Definition 2. *A* 1-nested P2RPQ *is a* P2RPQ*. A* $(k+1)$-nested P2RPQ *is a* P2RPQ *with regular expressions that may use binary* k*-nested* P2RPQ*s in addition to DL roles and concepts. The query language of* k*-nested* P2RPQ*s is denoted* $\mathsf{P2RPQ}^k$*, and the query language of arbitrarily nested* P2RPQ*s is denoted* $\mathsf{P2RPQ}^+$*.*

Example 1. The following $\mathsf{P2RPQ}^2$ uses a subquery $\psi[x,y] = R(x,y) \wedge L(x,y)$ to match an arbitrarily long chain of parallel R and L relationships:

$$\varphi[u,v] = \psi[x,y]^*(u,v) = (R(x,y) \wedge L(x,y))[x,y]^*(u,v) \tag{1}$$

Note that the explicit $[x,y]$ is necessary, since the formula $R(x,y) \wedge L(x,y)$ alone could also define a binary relation where y represents the first argument and x represents the second. It follows that $\mathsf{P2RPQ}^+$ does not require inverse roles in regular expressions: instead of R^- we can always use a subquery $R(x,y)[y,x]$ in regular expressions. Likewise, the operators $\cdot$ and | in regular expressions can be replaced by the use of $\wedge$ and $\vee$, respectively, in subqueries. Therefore, the Kleene star is the only relevant feature in regular expressions of $\mathsf{P2RPQ}^+$s, and one could simplify the definition accordingly.

Bienvenu et al. recently considered another notion of nested 2RPQs, which uses *existential test operators* as known from the XML query language XPath [1]. Indeed, in the absence of conjunction and disjunction, nested 2RPQs would not be any different from 2RPQs when defined as above. Instead, existential test operators $\langle E \rangle$ merely check if there is a path matching the regular expression E starting at the current element.

Definition 3. *Nested regular expressions are regular expressions that are constructed using regular expression operators and an additional unary operator* $\langle \cdot \rangle$*. A* nested two-way regular path query *(*N2RPQ*) over a DL signature is of the form* $E(s,t)$*, with* E *a nested regular expression over* $\mathbf{R} \cup \mathbf{C}$*, and* s *and* t *terms in* $\mathbf{V} \cup \mathbf{N_I}$*. The semantics of nested regular expressions under a DL interpretation* $\mathcal{I}$ *is defined by setting* $\langle E \rangle^{\mathcal{I}} := \{\langle \delta, \delta \rangle \mid \langle \delta, \delta' \rangle \in E^{\mathcal{I}} \text{ for some } \delta'\}$*, and defining the semantics of the remaining operators as before. A* conjunctive N2RPQ *(*CN2RPQ*) is of the form* $\exists \boldsymbol{y}.\varphi[\boldsymbol{x}, \boldsymbol{y}]$*, where* φ *is a conjunction of* N2RPQ*s, with the obvious semantics.*

Example 2. The CN2RPQ $\varphi[x,y] = (R \cdot \langle L^* \cdot C \rangle)^*(x,y)$ matches an arbitrarily long R-chain in which each but the first element can reach an element in C by some L-chain.

Comparing CN2RPQs to $\mathsf{P2RPQ}^+$s, we observe that test expressions $\langle E \rangle$ can be replaced by an 2RPQ $(E \cdot E^-)(x,x)$, where E^- is the "inverse" of E. Since $(E \cdot E^-)(x,x)$ is not a binary query, we cannot nest it into regular expressions directly. Therefore, if $\langle E \rangle$ is part of a regular expression, we need to transform part of this expression into a positive query, replacing $\cdot$ and | by $\wedge$ and $\vee$, respectively. In particular, the expressiveness of $\vee$ is needed for capturing CN2RPQs in this way.

Example 3. The generic way of expressing the query in Example 2 as a $\mathsf{P2RPQ}^+$ is $\psi[u,v]^*(x,y)$ where $\psi = R(u,v) \wedge (L^* \cdot C \cdot (L^-)^*)(v,v)$. An alternative formulation replaces ψ by $\psi' = \exists w.R(u,v) \wedge (L^* \cdot C)(v,w)$.

Although $\mathsf{P2RPQ}^+$ subsumes the expressiveness of CN2RPQ, the change from 2RPQ to N2RPQ already leads to an exponential increase in worst-case query answering complexity over some DLs [1].

One might ask if the restriction to binary subqueries in $\mathsf{P2RPQ}^+$ is really necessary. While the Kleene star operates on binary relations, one could allow such relations to be "parametrised" by additional variables. This idea is closely related to the extension of first-order logic with *transitive closure* [8]. We therefore abandon * as our last remaining regular expression operator to replace it by a restricted form of transitive closure.

Definition 4. *Consider lists of variables $\boldsymbol{x}$ and $\boldsymbol{y}$ (not necessarily disjoint) of equal length ℓ, and a formula φ where all variables of $\boldsymbol{x}$ and $\boldsymbol{y}$ occur as free variables. The* transitive closure *of φ from $\boldsymbol{x}$ to $\boldsymbol{y}$ is the formula $\mathsf{TC}_{\boldsymbol{x},\boldsymbol{y}}.\varphi$. A first-order interpretation $\mathcal{I}$ and a variable assignment $\mathcal{Z}$* satisfy *$\mathsf{TC}_{\boldsymbol{x},\boldsymbol{y}}.\varphi$, written $\mathcal{I}, \mathcal{Z} \models \mathsf{TC}_{\boldsymbol{x},\boldsymbol{y}}.\varphi$, if either $\mathcal{Z}(\boldsymbol{x}) = \mathcal{Z}(\boldsymbol{y})$ or there are variable assignments $\mathcal{Z}_1, \dots, \mathcal{Z}_n$, such that*

- *$\mathcal{Z}_i(z) = \mathcal{Z}(z)$ for all $z \notin \boldsymbol{x} \cup \boldsymbol{y}$ and $i \in \{1, \dots, n\}$,*
- *$\mathcal{I}, \mathcal{Z}_i \models \varphi$ for all $i \in \{1, \dots, n\}$,*
- *$\mathcal{Z}_1(\boldsymbol{x}) = \mathcal{Z}(\boldsymbol{x})$,*
- *$\mathcal{Z}_i(\boldsymbol{y}) = \mathcal{Z}_{i+1}(\boldsymbol{x})$ for all $i \in \{1, \dots, n-1\}$,*
- *$\mathcal{Z}_n(\boldsymbol{y}) = \mathcal{Z}(\boldsymbol{y})$.*

Throughout this paper, we restrict to first-order formulae over a DL signature. Positive first-order logic with transitive closure *(*$\mathsf{PFO{+}TC}$*) is first-order logic with operators $\exists$, $\wedge$, $\vee$, and* TC. Positive first-order logic with unary transitive closure *(*$\mathsf{PFO{+}TC1}$*) is the fragment of* $\mathsf{PFO{+}TC}$ *where the variable lists in transitive closure are of length $\ell = 1$. We use* $\mathsf{PFO{+}TC1}^k$ *for the fragment of* $\mathsf{PFO{+}TC1}$ *where* TC *is nested at most k times.*

It is easy to see that $\mathsf{PFO{+}TC1}$ can express $\mathsf{C2RPQ}^+$ queries. Moreover, queries of this type are still decidable for many DLs (see Section 6). In contrast, $\mathsf{PFO{+}TC}$ queries are undecidable for most DLs. This can be shown by reduction from the Post Correspondence Problem, using a transitive closure with two input and two output variables.

Example 4. The following $\mathsf{PFO{+}TC1}$ matches an arbitrarily long R-chain in which each element other than the first can reach the same element (represented by z) by an L-chain:

$$\varphi[x, y] = \exists z.\mathsf{TC}_{x,y}.(R(x, y) \wedge \mathsf{TC}_{y,z}.L(y, z)) \tag{2}$$

We are using a simplified notation here. For example, the subquery $\mathsf{TC}_{y,z}.L(y, z)$ in (2) would more accurately be expressed as $(\mathsf{TC}_{u,v}.L(u, v)[u, v])(y, z)$. The translation between both versions is not ambiguous, so we prefer the more compact one.

It should be noted that Definition 4 is strictly more general than the one that has traditionally been considered when studying transitive closure [8], since we allow formulae in the scope of TC to contain free variables that are not used in the transitive closure. Example 4 illustrates a case where this actually adds expressiveness, and we exploit this in Theorem 3 below.

4 Expressiveness of Nested Path Queries

From the observations in the previous section, we can already order the query languages by their relative expressiveness: $\mathsf{2RPQ} \subseteq \mathsf{CN2RPQ} \subseteq \mathsf{P2RPQ}^+ \subseteq \mathsf{PFO{+}TC1} \subseteq$

PFO+TC. In this section, we ask which of these inclusions are strict. This is not obvious, since nesting does not generally increase expressive power, even if it is not trivially expressible in a query language. For example, unions of conjunctive queries (UCQs) can express nested UCQs, but possibly at the cost of an exponential increase in size. Such effects may also lead to an increase in worst-case query or combined complexity without any corresponding increase in expressiveness.

Formally, we compare the expressiveness of Boolean queries, i.e., queries without free variables. Any such query characterizes a set of interpretations that satisfy the query. A query language A is (strictly) more expressive than a query language B if A can characterize (strictly) more sets of models than B. It is easy to extend this definition to non-Boolean queries, but since we already showed the non-strict inclusions by direct translations, we do not need to consider this complication.

Hence our main tool to study expressiveness are general properties of the models that match a query. Since we consider positive queries, the matches of which are preserved under homomorphisms of interpretations, we may focus on subsets of models.

Definition 5. *Consider interpretations $\mathcal{I} = \langle \cdot^{\mathcal{I}}, \Delta^{\mathcal{I}} \rangle$ and $\mathcal{J} = \langle \cdot^{\mathcal{J}}, \Delta^{\mathcal{J}} \rangle$. A mapping $\eta : \Delta^{\mathcal{I}} \to \Delta^{\mathcal{J}}$ is a* homomorphism *if for all $c \in \mathbf{N_I}$, $\eta(c^{\mathcal{I}}) = c^{\mathcal{J}}$; for all $A \in \mathbf{N_C}$, $\delta \in A^{\mathcal{I}}$ implies $\eta(\delta) \in A^{\mathcal{J}}$; for all $R \in \mathbf{N_R}$, $\langle \delta, \delta' \rangle \in R^{\mathcal{I}}$ implies $\langle \eta(\delta), \eta(\delta') \rangle \in R^{\mathcal{J}}$.*

A set of interpretations $\mathscr{I}$ is a covering *of a Boolean query φ if, for all models $\mathcal{I}$, $\mathcal{I} \models \varphi$ iff there is a homomorphism from an interpretation $\mathcal{J} \in \mathscr{I}$ to $\mathcal{I}$.*

Intuitively speaking, a covering of a query represents every situation in which the query can match. There are situations where a minimal covering (i.e., a covering not properly containing another covering) does not exist. For example, for any $\ell \geq 0$, the query $\exists x.R^*(x, x)$ is covered by the set of interpretations that describe R-loops of length greater than ℓ, none of which is minimal. The example illustrates that coverings nevertheless can reveal interesting structural properties of query matches. We use this to prove the next theorem.

Bienvenu et al. already note that CN2RPQs can be rewritten as C2RPQ by encoding test expressions in TBox axioms using intersection, existentials, and inverse roles [1]. For DLs that lack existentials or inverses, however, test expressions increase expressiveness, which this does not follow from known complexity results.

Theorem 1. *For DLs that lack either existential quantifiers or inverse roles,* CN2RPQ *is strictly more expressive than* C2RPQ.

Proof. The *in-degree* of an element in an interpretation is the number of other elements that have a binary relation towards it. Every Boolean C2RPQ with n variables has a covering of interpretations that each contain at most n elements of in-degree greater than 2. Indeed, every RPQ has a covering of linear interpretations (with all elements of degree at most 2), and a conjunctive query over such structures introduces only at most n "joint points" of higher degree. These observations hold since concepts of the form $\exists R^-.C$ cannot occur in the query.

On the other hand, the CN2RPQ in Example 2 (considered as a Boolean query) only admits coverings with an unbounded number of elements with degree 3. Hence, no Boolean C2RPQ can characterize this set of models. □

CN2RPQs in turn are limited by the fact that test expressions can only describe tree-like structures, but not loops. This is not a restriction when using them in XML query languages, which operate on trees anyway, but it limits there expressiveness on arbitrary (finite) structures, as described by DL ABoxes. To formalize this, we count the number of distinct paths between two elements in a model.

Theorem 2. P2RPQ$^+$ *is strictly more expressive than* CN2RPQ.

Proof. Consider an interpretation $\mathcal{I}$. A *path* from δ to δ' in $\mathcal{I}$ is a finite sequence $\delta_1 R_1 \delta_2 \dots \delta_{n-1} R_{n-1} \delta_n$, such that $R_i \in \mathbf{N_R}$, $\delta_1 = \delta$, $\delta_n = \delta'$, and $\langle \delta_i, \delta_{i+1} \rangle \in R_i^{\mathcal{I}}$ for all $i \in \{1, \dots, n-1\}$. A path is *simple* if the are no i, j with $\delta_i = \delta_j$, $R_i = R_j$, and $\delta_{i+1} = \delta_{j+1}$.

Every Boolean CN2RPQ with n variables has a covering of finite interpretations such that, for every δ, δ', the number of paths from δ to δ' is bounded by a constant. Indeed, it is easy to see that every 2RPQ admits a covering of interpretations with at most one path between any two elements. The number of possible paths in a CN2RPQ therefore is in direct correspondence with the number of atom-paths between variables of the query, which is clearly bounded.

On the other hand, the P2RPQ$^+$ in Example 1 (considered as a Boolean query) only admits coverings that contain structures of the form $R(\delta_1, \delta_2), L(\delta_1, \delta_2), \dots, R(\delta_{n-1}, \delta_n)$, $L(\delta_{n-1}, \delta_n)$ for arbitrarily large n. Such a structure admits 2^n distinct paths from δ_1 to δ_n. Hence, no Boolean CN2RPQ can characterize this set of models. □

Nevertheless, P2RPQ$^+$ is still weaker than PFO+TC1. The structural property we use in this case is the maximal degree of nodes in covering interpretations.

Theorem 3. PFO+TC1 *is strictly more expressive than* P2RPQ$^+$.

Proof. For every Boolean P2RPQ$^+$, there is a constant d, for which the query admits a covering of interpretations that each contain only elements of degree at most d. Indeed, the nesting of queries can lead to an arbitrary number of elements of degree up to d, but it cannot increase the maximum degree.

On the other hand, the PFO+TC1 in Example 4 (considered as a Boolean query) only admits coverings with elements of unbounded degree (represented by the variable z). Hence, no Boolean P2RPQ$^+$ can characterize this set of models. □

5 Hardness of PFO+TC1 Query Answering

We now show the following lower complexity bound for query answering in the DL $\mathcal{S}$:

Theorem 4. *Deciding entailment of* PFO+TC1^k *queries over* $\mathcal{S}$ *knowledge bases is hard for* $(k+2)$ExpTime.

To show this, we provide direct encodings of Alternating Turing Machines (ATMs) with a fixed space bound [6], where we assume without loss of generality that every universal ATM configuration leads to exactly two successor configurations. Moreover, when considering ATMs with limited space, we assume that it is a legal transition to move to the left/right at the left/right end of the tape: this will result in the ATM head

Table 1. TBox for the knowledge base in Proposition 1

$$
\begin{aligned}
\mathsf{Run} &\sqsubseteq \exists \mathsf{firstConf}.\mathsf{Conf} \\
\mathsf{Conf} &\sqsubseteq \bigsqcup_{q\in Q} \mathsf{State}_q \sqcap \exists \mathsf{firstCell}.\mathsf{Cell} \\
\mathsf{Cell} &\sqsubseteq \bigsqcap_{i=1}^{\ell} (\mathsf{Bit0}_i \sqcup \mathsf{Bit1}_i) \sqcap \bigsqcup_{\sigma\in\Sigma} \mathsf{Symbol}_\sigma \sqcap (\mathsf{L} \sqcup \mathsf{H} \sqcup \mathsf{R}) \sqcap (\mathsf{LastCell} \sqcup \exists \mathsf{nextCell}.\mathsf{Cell}) \\
\mathsf{Conf} &\sqsubseteq \mathsf{LastConf} \sqcup \bigsqcup_{q\in Q_\exists}^{\delta=\langle q,\sigma,q',\sigma',d\rangle} \exists \mathsf{nextConf}_\delta.\mathsf{Conf} \sqcup \\
&\qquad \bigsqcup_{q\in Q_\forall,\delta_1\neq\delta_2}^{\delta_1=\langle q,\sigma,q',\sigma',d\rangle} (\exists \mathsf{nextConf}_{\delta_1}.\mathsf{Conf} \sqcap \exists \mathsf{nextConf}_{\delta_2}.\mathsf{Conf})
\end{aligned}
$$

staying at the same tape cell. This allows an ATM to detect the limits of a tape. One application for this is to implement an ATM that, when run on a tape with an arbitrary space bound s, will count from 0 to 2^s and then halt.

We can also use this to transform any given ATM $\mathcal{M}$ into an ATM $\mathcal{M}'$ that performs the computation of $\mathcal{M}$ while counting the required steps in space s. The counter can be written on the tape using an extended alphabet that encodes a symbol of $\mathcal{M}$ and a counter digit in each symbol. $\mathcal{M}'$ performs one transition of $\mathcal{M}$, stores the current tape symbol, marks the current tape position, increments the tape counter, returns to the original tape position, restores its original content, and enters the next state of $\mathcal{M}$. Using this technique, we can construct an ATM that enters an existential state without legal transitions (rejecting state) as soon as more than k^s steps are executed (for any $k \geq 2$). Since this construction is polynomial in the size of $\mathcal{M}$ and k, we can assume without loss of generality that all of our space-bounded ATMs halt, i.e., do not admit infinite runs. In this case, every valid run is accepting, which greatly simplifies our encoding.

Before looking at the general case, we illustrate our approach by reducing the acceptance of an ExpSpace ATM to $\mathsf{PFO{+}TC1}^1$ query answering over $\mathcal{S}$ knowledge bases.

Proposition 1. *For any ATM $\mathcal{M}$, there is an $\mathcal{S}$ knowledge base* KB *and a* $\mathsf{PFO{+}TC1}^0$ *$Q[x]$, such that $\mathcal{M}$ accepts the empty input in exponential space iff* $\mathrm{KB} \not\models \exists x.Q[x]$.

KB consists of the TBox shown in Table 1 and the RBox axioms $\mathsf{firstCell} \sqsubseteq \mathsf{cellConf}$, $\mathsf{nextCell} \sqsubseteq \mathsf{cellConf}$, and $\mathsf{Trans}(\mathsf{cellConf})$. Typical models of KB are tree structures that resemble the runs of an ATM, with the concepts Conf and Cell representing configurations and tape cells, respectively. Concepts L, H, and R specify, for each cell, if the head is to its left, on top of it, or to its right. Cells are marked by addresses of ℓ bits.

These intuitions may not be followed by all models. The query Q is used to filter erroneous encodings. First, we define queries that detect when the address has not been incremented correctly from one cell to the next. Together with suitable starting end ending conditions, this already yields queries that ensure that all tapes have exactly 2^ℓ cells. We then define a query $\mathit{SameCell}[x,y] \coloneqq \bigwedge_{i=1}^{\ell} (\mathsf{Bit0}_i(x) \wedge \mathsf{Bit0}_i(y)) \vee (\mathsf{Bit1}_i(x) \wedge \mathsf{Bit1}_i(y))$ that relates tape cells with the same address. This can be used to express other conditions for correct ATM runs, starting with the requirement that L, H, and R are used consistently. Transitions are verified using queries

$$
\begin{aligned}
&\mathsf{nextConf}_\delta(y,y') \wedge \mathsf{State}_q(y) \wedge \mathsf{cellConf}(z,y) \wedge \mathsf{H}(z) \wedge \mathsf{Symbol}_\sigma(z) \wedge \\
&\qquad\qquad \mathsf{State}_{q'}(y') \wedge \mathsf{cellConf}(z',y') \wedge \mathsf{Symbol}_{\sigma'}(z') \wedge \mathit{SameCell}(z',z)
\end{aligned}
$$

for every transition $\delta = \langle q_1, \sigma_1, q_2, \sigma_2, d\rangle$ where $q_1 \neq q$, $\sigma_1 \neq \sigma$, $q_2 \neq q'$, or $\sigma_2 \neq \sigma'$.

Proposition 1 merely shows that answering positive Boolean queries over $\mathcal{S}$ is hard for 2ExpTime. For higher complexities, we use nesting to encode "higher-level" ATMs with longer tapes. The general setup is as in Proposition 1, but the tape of an ATM on level $i+1$ is obtained as the sequence of configurations of a deterministic TM of level i.

As explained above, there is a deterministic TM $\mathcal{M}_{\text{count}}$ that, when run on a tape of space s, will count from 0 to 2^s in binary and then halt. $\mathcal{M}_{\text{count}}$ can be small (constant size). The computation will necessarily take $s' > 2^s$ steps to complete, so the accepting runs form a chain of s' configurations. Thus, using the construction of Proposition 1, we obtain a knowledge base KB_0 and a query Q_0 such that, for every interpretation $\mathcal{I}$ with $\mathcal{I} \models \text{KB}_0$ and $\mathcal{I} \not\models \exists x.Q_0(x)$, every $\epsilon \in \mathsf{Run}^{\mathcal{I}}$ is the start of such a chain that is exactly of length s'.

We can therefore axiomatize an ATM $\mathcal{M}'$ that runs in space $s' > 2^s \in O(2^{(2^n)})$ by using the axioms in Table 1, replacing Cell (of Table 1) by Run (of KB_0), firstCell by firstConf, nextCell by nextConf, etc. and using fresh vocabulary symbols instead of firstConf, nextConf, etc. RBox axioms are created in the same way following the earlier construction. Let KB' denote the resulting knowledge base.

Together with $Q_0[x]$, KB' describes models that resemble runs of $\mathcal{M}'$, one tapes of a fixed length, but possibly violating some other conditions. It remains to define additional queries to check for invalid runs. The key for doing this is a $\mathit{SameCell}'[x, y]$ query that relates the cells of the same address across tapes. This is difficult now since the tapes are very large. For brevity, we use an auxiliary query $\mathit{SameContent}'[v, w]$ to check if two tape cells have the exact same annotations for $\mathcal{M}'$ (clearly, this is just a simple positive Boolean query). We now can define $\mathit{SameCell}'[x, y]$ as follows:

$$\begin{aligned}&\exists v'.\mathsf{firstCell}'(x, v) \wedge \mathsf{firstCell}'(y, w) \wedge \mathit{SameContent}'(v, w) \wedge \\ &\quad \mathsf{TC}_{v,v'}.\exists u, u'.(\mathit{SameCell}(v, u) \wedge \mathsf{nextCell}'(v, v') \wedge \mathsf{nextCell}'(u, u') \wedge \\ &\qquad\qquad \mathit{SameContent}'(v', u') \wedge \mathsf{cellConf}(u, y) \qquad\qquad) \wedge \mathsf{LastCell}'(v')\end{aligned}$$

Note how this query nests the *SameCell* query of the previous level, thus leading to an increase in nesting depth. The queries required to check for invalid runs of $\mathcal{M}'$ are now expressed as before, using $\mathit{SameCell}'$ where required.

This construction can be repeated, leading to an exponential increase in tape length for every additional nesting level. Note that the subquery *SameCell* occurs exactly once in $\mathit{SameContent}'$, making sure that there is only a linear increase in query size during nesting. Using Proposition 1 as an induction base, we thus obtain Theorem 4.

6 Deciding Entailment for PFO+TC1

In this section, we outline how to answer PFO+TC1 queries over knowledge bases in DLs with certain model-theoretic properties. This generalizes known results for P2RPQs [5], and we build on and extend the according line of argumentation.

We consider DLs exhibiting a *quasi-forest hom-cover property* (QFHC), which states that, for any model $\mathcal{I}$ of some knowledge base KB, there exists a homomorphism from $\mathcal{I}$ into some model $\mathcal{I}'$ of KB that has a specific shape, called quasi-forest. The most expressive DLs known to have this property are $\mathcal{ZIQ}$, $\mathcal{ZOQ}$, and $\mathcal{ZOI}$ [5].

Definition 6 ([5]). *A model $\mathcal{I}$ of a knowledge base* KB *is a* quasi-forest model *if:*

- *the domain $\Delta^{\mathcal{I}}$ of $\mathcal{I}$ is a forest with bounded branching k, i.e., a prefix-closed subset of $Roots \times \{1, \ldots, k\}^*$ for some finite set Roots*
- *$Roots = \{a^{\mathcal{I}} \mid$ a individual name in* KB$\}$,
- *for all $\delta, \delta' \in \Delta^{\mathcal{I}}$ satisfying $\langle \delta, \delta' \rangle \in R^{\mathcal{I}}$ for some role name $R \in \mathbf{N_R}$, either (i) $\{\delta, \delta'\} \cap Roots \neq \emptyset$, or (ii) $\delta = \delta'$, or (iii) δ is a child of δ', or (iv) δ' is a child of δ.*

The following property is then an easy consequence of the fact that the set of models of each PFO+TC formula is closed under homomorphisms.

Property 1. Let KB be a knowledge base in any DL satisfying the QFHC property and let q be a PFO+TC formula. KB does not entail q iff there exists a quasi-forest model $\mathcal{I}$ of KB not satisfying q.

Finally, we recall from [5] that for each KB there is an encoding tree of quasi-forest interpretations of branching degree $\leq k$ over the vocabulary of KB into infinite trees of fixed rank k over some finite alphabet Λ in a way that allows one to define a *one-way nondeterministic parity tree automaton* (1NTA) $\mathcal{A}_{\text{KB}}$ of size double exponential in KB recognizing exactly the trees $t = \text{tree}(\mathcal{I})$ that encode quasi-forest models $\mathcal{I}$ of KB. As tree is injective, we will write $\text{tree}^{-1}(t)$ to refer to $\mathcal{I}$, presuming its existence.

We now describe how to build, for a PFO+TC1 formula using as unary (binary) predicates only concept names (roles) occurring in KB, a 1NTA $\mathcal{A}_{\text{KB},q}$ that recognizes all trees corresponding to quasi-forest interpretations into which q has a match.

First, we show how to build the tree-automaton for conjunctive queries, then for positive queries, and then for a query of the form $\mathsf{TC}_{x,y}.\varphi(x, y, z)$ for a positive query φ. Finally, we generalize the construction by induction on the nesting of TC operators.

Let q be a conjunctive query with a set of free variables X. In a first step, we want to recognize trees representing quasi-forest interpretations $\mathcal{I}$ that have been enriched by the information stemming from a variable assignment $\mathcal{Z} : X \to \Delta^{\mathcal{I}}$. To this end, let Λ_q be the finite alphabet equal to $\Lambda \times 2^X$.

Let t be a tree over Λ_q. Intuitively, if a node n of t is labeled by $\langle \alpha, S \rangle$ with $S \subseteq X$, then all variables from S are mapped to n by $\mathcal{Z}$. Because each node of some quasi-forest interpretation $\mathcal{I}$ corresponds to exactly one node of $\text{tree}(\mathcal{I})$, there shall not exist two nodes in t labeled with two tags both containing the same variable x. In other words, the S-labels of all tree nodes have to be pairwise disjoint. Moreover, every variable has to occur in some label. Given a tree t over Λ_q satisfying these conditions, we let $\mathcal{Z}_t$ denote the variable assignment mapping every $x \in X$ to the individual $\delta \in \Delta^{\mathcal{I}}$ for which the label of the tree node corresponding to δ contains x. We let $\Pi_\Lambda(t)$ denote the tree obtained by projecting the labels of t to Λ.

Property 2. Consider KB and a conjunctive query q with a set of free variables X. There exists a 1NTA $\mathcal{A}_{\text{KB},q}$ such that t is accepted by $\mathcal{A}_{\text{KB},q}$ iff $\mathcal{Z}_t$ exists and q is satisfied by $\text{tree}^{-1}\Pi_\Lambda(t)$ and $\mathcal{Z}_t$. The size of the states of $\mathcal{A}_{\text{KB},q}$ is exponential in q. The length of the parity condition of $\mathcal{A}_{\text{KB},q}$ is bounded by a constant.

This is a well-known classical construction presented in previous papers (e.g. [5]). We can lift this result to arbitrary positive queries without further blow-up:

Corollary 1. *Consider* KB *and a positive query q with a set of free variables X. There exists a 1NTA $\mathcal{A}_{\mathrm{KB},q}$ such that t is accepted by $\mathcal{A}_{\mathrm{KB},q}$ iff $\mathcal{Z}_t$ exists and q is satisfied by $\mathrm{tree}^{-1}\Pi_\Lambda(t)$ and $\mathcal{Z}_t$. The size of the set of states $\mathcal{A}_{\mathrm{KB},q}$ is exponential in q. The length of the parity condition of $\mathcal{A}_{\mathrm{KB},q}$ is bounded by a constant.*

Every positive query q can be decomposed into a union of conjunctive queries $\bigvee_{i\leq n} q_i$ such that each q_i is polynomial in q and n is exponential in q. For each q_i, we build a 1NTA $\mathcal{A}_{q_i}$ using Property 2. The 1NTA associated with $\bigvee_{i\leq k} q_i$ is equal to $\cup_{i\leq k}\mathcal{A}_{q_i}$. The size of $\cup_{i\leq k}\mathcal{A}_{q_i}$ is the sum of the size of each q_i. Since n is exponential in q and each $\mathcal{A}_{q_i}$ is exponential in q_i (i.e., in q) $\mathcal{A}_{\bigvee_{i\leq n} q_i}$ is exponential in q.

Next, toward the construction of an automaton recognizing the transitive closure of some query, we need as an intermediate building block an automaton running on a tree, where $\mathcal{Z}$ is given via labeling (as above) for *all but one* variable. The node n associated to the "missing variable" is pinpointed by letting the automaton start at n (instead of letting it start at the root, as usual). In order for this automaton to access nodes above n, it has to be a *two-way alternating tree automaton* (2ATA).

Property 3. Let KB be a knowledge base and let q be a query with a set of free variables X. Let q represented by some 1NTA $\mathcal{A}_{\mathrm{KB},q}$ as described above. Let $x \in X$ and let $X' = X \setminus \{x\}$. Then there exists a 2ATA $\mathcal{A}_{\mathrm{KB},q,x}$ such that $\mathcal{A}_{\mathrm{KB},q,x}$ accepts a $\Lambda \times 2^{X'}$-labeled tree t when starting from n iff $\mathcal{Z}_t$ exists and q is satisfied by $\mathrm{tree}^{-1}\Pi_\Lambda(t)$ and $\mathcal{Z}_t \cup \{x \mapsto \mathrm{tree}^{-1}(n)\}$. The size of $\mathcal{A}_{\mathrm{KB},q,x}$ is linear in the size of $\mathcal{A}_{\mathrm{KB},q}$. The length of the parity condition of $\mathcal{A}_{\mathrm{KB},q,x}$ is bounded by a constant.

We can construct $\mathcal{A}_{\mathrm{KB},q,x}$ by modifying the 1NTA $\mathcal{A}_{\mathrm{KB},q}$. If t is accepted by $\mathcal{A}_{\mathrm{KB},q}$, there exists exactly one node labeled with a tag containing x. If we start from this node and guess the state $\mathfrak{q}$ that $\mathcal{A}_{\mathrm{KB},q}$ has in an accepting run on t, we need to verify that (i) the subtree t_n rooted at n admits a partial run of $\mathcal{A}_{\mathrm{KB},q}$ starting in state $\mathfrak{q}$ and that (ii) the tree t without the subtree t_n admits a run of $A_{\mathrm{KB},q}$ such that n is associated to $\mathfrak{q}$. We can check (i) with a 1NTA and (ii) with a 2APA starting from n.

Finally, we build the 2ATA for a formula $\mathsf{TC}_{x,y}.\varphi(x,y,z)$.

Property 4. Let $q = \mathsf{TC}_{x,y}.\varphi(x,y,z)$ be a query where φ is represented by some 2ATA $\mathcal{A}_{\mathrm{KB},\varphi,x}$ as described above. Then there exists a 2ATA $\mathcal{A}_{\mathrm{KB},q}$ such that $\mathcal{A}_{\mathrm{KB},q}$ accepts a $\Lambda \times 2^X$-labeled tree t iff $\mathcal{Z}_t$ exists and q is satisfied by $\mathrm{tree}^{-1}\Pi_\Lambda(t)$. The size of the set of states of $\mathcal{A}_{\mathrm{KB},q}$ is linear in the size of $\mathcal{A}_{\mathrm{KB},\varphi,x}$. The length of the parity condition of $\mathcal{A}_{\mathrm{KB},q}$ is bounded by a constant.

$\mathcal{A}_{\mathrm{KB},q,x}$ can be obtained from $\mathcal{A}_{\mathrm{KB},\varphi,x}$ by simple modifications: $\mathcal{A}_{\mathrm{KB},q}$ traverses the tree top-down until reaching the node with the label containing x and then starts to run $A_{\mathrm{KB},\varphi,x}$; whenever $\mathcal{A}_{\mathrm{KB},\varphi,x}$ requires to read a label containing y, we allow it to return to an initial state instead.

The construction for general nested PFO+TC1s is done by induction over the nesting of the TC operators. We thereby assume w.l.o.g. all considered PFO+TC1s to be in *nested prenex form* (NPF), that is, existential quantifiers can only occur in front of the whole formula or directly after $\mathsf{TC}_{x,y}$. Every PFO+TC1 query can be rewritten into an equivalent NPF query of linear size. This may require some renaming of variables.

Lemma 1. *Let* KB *be given. Let q be a* PFO+TC1^n *formula in NPF with a set of free variables X. Then there exists a 1NTA $\mathcal{A}_{\mathrm{KB},q}$ such that t is accepted by $\mathcal{A}_{\mathrm{KB},q}$ iff $\mathcal{Z}_t$ exists and q is satisfied by* $\mathrm{tree}^{-1}\Pi_A(t)$ *and $\mathcal{Z}_t$. The size of $\mathcal{A}_{\mathrm{KB},q}$ is $(n+1)$-exponential in q and the parity condition has a fixed length.*

Proof (Sketch). The proof is by induction on the nesting depth. The base case (depth 0) is stated in Corollary 1. For the induction step, consider a PFO+TC1^n formula $\psi = \exists z.\psi'$ where ψ' is a positive Boolean expression over $\mathsf{TC}_{x_1,y_1}.\psi'_1, \ldots \mathsf{TC}_{x_k,y_k}.\psi'_k$ with $\psi'_1, \ldots, \psi'_k$ in PFO+TC1^{n-1}. By induction hypothesis, we find 1NTAs $\mathcal{A}_{\mathrm{KB},\psi'_1}, \ldots, \mathcal{A}_{\mathrm{KB},\psi'_k}$ of n-exponential size. We use Property 3 and 4 to get 2ATAs $\mathcal{A}_{\mathrm{KB},\mathsf{TC}_{x_1,y_1}.\psi'_1}, \ldots, \mathcal{A}_{\mathrm{KB},\mathsf{TC}_{x_k,y_k}.\psi'_k}$ of still n-exponential size. Exploiting that for 2ATAs, intersection and union automata have only linear size, we arrive at a 2ATA $\mathcal{A}_{\mathrm{KB},\psi'}$ of n-exponential size. Now, we obtain the 1NTA $\mathcal{A}_{\mathrm{KB},\psi}$ by turning $\mathcal{A}_{\mathrm{KB},\psi'}$ into a 1NTA (exponential) and then projecting away all variable labels of z (polynomial). Thus $\mathcal{A}_{\mathrm{KB},\psi}$ is of $(n+1)$-exponential size. □

Theorem 5. *Let* KB *be a $\mathcal{ZOIQ}$ knowledge base with the QFHC property. Let q be a Boolean* PFO+TC1^n *formula. Then deciding if* KB *entails q is in* $(n+2)$ExpTime.

Proof (Sketch). Following [5], we can build a 1NTA $\mathcal{A}_{\mathrm{KB}}$ that recognizes tree representations of quasi-forest models of KB of branching degree at most k. It suffices to consider models of such branching degrees as the models of q are closed under homomorphism. $\mathcal{A}_{\mathrm{KB}}$ is double exponential in KB. By Lemma 1, we can find an 1NTA $\mathcal{A}_{\mathrm{KB},q}$ recognizing the tree representation of quasi-forest interpretations satisfying q. By taking the complement of $\mathcal{A}_{\mathrm{KB},q}$, $\mathcal{A}_{\mathrm{KB},\neg q}$, this 1NTA recognizes the tree representation of quasi-forest interpretations not satisfying q. This 1NTA has a size $(n+2)$-exponential in KB and q. Finally, the intersection of the tree languages accepted by $\mathcal{A}_{\mathrm{KB}}$ and $\mathcal{A}_{\mathrm{KB},\neg q}$ is empty iff KB entails q. Hence, we just have to check the emptiness of the product automaton of $\mathcal{A}_{\mathrm{KB}}$ and $\mathcal{A}_{\mathrm{KB},\neg q}$. This is polynomial w.r.t. to the two automata and exponential w.r.t. the parity conditions. As the latter are fixed, we have shown our claim. □

7 Conclusion

This work is a starting point for a systematic study of a range of highly expressive regular path query languages that are obtained by introducing various forms of nesting. Our results open up a field of decidable DL query languages that require further study.

The complexity bounds we obtain are very high: non-elementary for the most general language PFO+TC1, and 3ExpTime even for a single level of nesting in PFO+TC1^1. In contrast, CN2RPQ query answering is in 2ExpTime irrespective of the nesting level, and is known to become simpler when restricting to weaker DLs [1]. There is a host of open research questions regarding the space in between these two results.

Moreover, the high complexity of PFO+TC1 does not mean that this is the most expressive query language that is still decidable for DLs. Indeed, PFO+TC1 is properly contained in *nested monadically defined queries*, proposed recently as a combination of monadic Datalog and regular path queries [11], and all of our complexity arguments can be extended to this case as well.

Acknowledgement This work was supported by the DFG in project DIAMOND (Emmy Noether grant KR 4381/1-1).

References

1. Bienvenu, M., Calvanese, D., Ortiz, M., Šimkus, M.: Nested regular path queries in description logics. In: Proc. 14thInt. Conf. on Principles of Knowledge Representation and Reasoning (KR'14). AAAI Press (2014), to appear
2. Bienvenu, M., Ortiz, M., Šimkus, M.: Conjunctive regular path queries in lightweight description logics. In: Rossi, F. (ed.) Proc. 23rd Int. Joint Conf. on Artificial Intelligence (IJCAI'13). pp. 761–767. AAAI Press/IJCAI (2013)
3. Calvanese, D., De Giacomo, G., Lenzerini, M., Vardi, M.Y.: Containment of conjunctive regular path queries with inverse. In: Cohn, A.G., Giunchiglia, F., Selman, B. (eds.) Proc. 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'00). pp. 176–185. Morgan Kaufmann (2000)
4. Calvanese, D., Eiter, T., Ortiz, M.: Answering regular path queries in expressive description logics: An automata-theoretic approach. In: Proc. 22nd AAAI Conf. on Artificial Intelligence (AAAI'07). pp. 391–396. AAAI Press (2007)
5. Calvanese, D., Eiter, T., Ortiz, M.: Regular path queries in expressive description logics with nominals. In: Boutilier, C. (ed.) Proc. 21st Int. Joint Conf. on Artificial Intelligence (IJCAI'09). pp. 714–720. IJCAI (2009)
6. Chandra, A.K., Kozen, D.C., Stockmeyer, L.J.: Alternation. J. of the ACM 28(1), 114–133 (1981)
7. Florescu, D., Levy, A., Suciu, D.: Query containment for conjunctive queries with regular expressions. In: Mendelzon, A.O., Paredaens, J. (eds.) Proc. 17th Symposium on Principles of Database Systems (PODS'98). pp. 139–148. ACM (1998)
8. Immerman, N.: Languages that capture complexity classes. SIAM J. Comput. 16(4), 760–778 (1987)
9. Krötzsch, M., Simančík, F., Horrocks, I.: A description logic primer. CoRR abs/1201.4089 (2012)
10. Pérez, J., Arenas, M., Gutierrez, C.: nSPARQL: A navigational language for RDF. J. of Web Semantics 8, 255–270 (2010)
11. Rudolph, S., Krötzsch, M.: Flag & check: Data access with monadically defined queries. In: Hull, R., Fan, W. (eds.) Proc. 32nd Symposium on Principles of Database Systems (PODS'13). pp. 151–162. ACM (2013)