Abstract. We propose a query answering technique applicable to a wide family of cyclic conjunctive queries with one distinguished variable. It exploits the fact that, given such a query, one can build in linear time a concept of some Description Logic (DL) such that to answer the query w.r.t. any knowledge base (KB) is the same as to find all instances of that concept. Notably, the method is uniform: the concept does not depend on a KB and even on a DL in which the KB is formulated. Thus, for these conjunctive queries, the problem of query answering is reduced to concept instance retrieval, which in turn is reducible to KB (un)satisfiability. The technique presented here is based on the modal correspondence theory (and close relationship between modal and description logics) and can be regarded as a practical application of that theory.

1 Introduction

Description Logics are family of knowledge representation formalisms closely related to first-order and modal logics. Most of them can be viewed as decidable fragments of the first-order logic. Their complexity was thoroughly investigated, and practical reasoning algorithms were designed (e.g., [9, 11, 14]) and implemented as highly optimised reasoning systems, e.g., Racer [7], FaCT [8], and Pellet [18]. Typical reasoning services provided by these systems are checking KB satisfiability, concept subsumption (and building a taxonomy, i.e., a computing subsumption relationships between all concept names occurring in a KB), concept instances retrieval etc. (see [1, Chap. 8] for an overview).

Among reasoning tasks, querying is a fundamental mechanism for extracting information from a KB. Two most important reasoning services involving queries—query answering and query containment, also called subsumption—were extensively explored [3–6, 10, 12, 17], but have not got a complete solution. A typical obstacle is the presence of cycles in a query (cf. [6]). In general, query containment and answering are reducible to each other and are at least as hard as concept subsumption or satisfiability problems (whence their lower complexity bounds). However, to the best of our knowledge, no tight upper bounds for their complexity are obtained so far. To illustrate the situation, recall that the Description Logic $SHIQ$ is known to be $ExpTime$-complete [11]. At the same...
time, the complexity of query answering over SHIQ KBs is shown in [5] to be 3coNExpTime, if a KB has no transitive roles, and 4coNExpTime otherwise.

In this paper, we concentrate on the query answering problem. We show that for a wide family of conjunctive queries (with one distinguished variable) one can build in linear time a concept (usually, in a very simple DL) whose instances are exactly the answers to the query (w.r.t. any KB). As a consequence, within this family, the query answering problem is reduced to concept instance retrieval, which in turn is reducible to KB unsatisfiability. Note that the reduction is applicable to many cyclic queries, and the resulting concept is not in general equivalent to the query. The technique presented here is based on the modal correspondence theory; this theory explores the relation between the properties of frames expressible in modal and first-order languages.

To illustrate the method, consider the following instance retrieval request: \( KB \models a : (\neg X \sqcup \exists R.X) \), where the concept name \( X \) does not occur in the knowledge base \( KB \). The task is to find all individuals \( a \) (called constants in first-order logic) occurring in \( KB \) that belong to the concept \( \neg X \sqcup \exists R.X \) in any model of \( KB \). It is not hard to show that exactly those individuals \( a \) will be retrieved that satisfy (in all models of \( KB \)) the condition \( aRa \). Hence, the concept \( \neg X \sqcup \exists R.X \) can be used for answering the cyclic query \( q(x) \leftarrow xRx \). Similarly, the request \( KB \models a : (\exists R.\neg X \sqcup \exists S.X) \), where \( X \) does not occur in \( KB \), returns exactly those individuals \( a \) that are answers of the query \( q(x) \leftarrow \exists y(xRy \land xSy) \). Thus, concepts involving fresh concept names can be used to compute the answers of even cyclic queries.

Now recall from modal logic that the formula \( p \rightarrow \Diamond p \) is valid at a point \( e \) of a frame iff this point is reflexive, i.e., \( eRe \) holds. Similarly, the formula \( \Box Rp \rightarrow \Diamond Sp \) is valid at a point \( e \) iff there is a point \( d \) such that \( eRd \) and \( eSd \) holds, i.e., \( e \) satisfies the first-order condition \( \alpha(x) := \exists y(xRy \land xSy) \). Comparing this with the above observations, we can generalise it as follows: whenever the validity of a modal formula at a point of a frame is equivalent to some first-order condition (and this condition has the form of conjunctive query), we can use this fact for query answering. This is the main result of this paper (Theorem 7), and an open question is whether the converse implication holds. An important feature of this approach is its uniformity: a concept used to answers a query is not only independent of a KB against which the query is answered, but also independent of a DL in which a KB is formulated. Therefore, extending the expressivity of a DL does not destroy our query answering algorithm (in contrast to other approaches where, e.g., introducing transitive roles can invalidate an algorithm).

In the next section we recall some notions concerning DLs and query answering. Section 3 contains the main theorem of the paper, which enables us to transfer results from modal correspondence theory to query answering. As an application, Sect. 4 and 5 present two families of queries that can be handled by this approach. As a by-product, in Sect. 5 we obtain an (at least syntactic) extension to the well-known in modal logic Sahlqvist theorem. Numerous examples of queries captured by this technique are shown in Sect. 6. We give conclusions and an outlook in Sect. 7. An Appendix contains all the proofs.
2 Preliminaries

First let us recall the notions related to the DL $\mathcal{ALC}$ and its extensions.

**Definition 1. (Syntax)** The vocabulary consists of finite sets of concept names $\mathcal{CN}$, role names $\mathcal{RN}$, and individual names $\mathcal{IN}$ (also called as constants). Concepts of the logic $\mathcal{ALC}$ are defined by the following syntax:

$$C ::= \top | A | \neg C | C \sqcap D | \exists R.C,$$

where $A$ is a concept name, $R$ a role name, and $C$ and $D$ are concepts. Other connectives are taken as customary abbreviations, e.g., $(C \sqcup D) := \neg(\neg C \cap \neg D)$, $(C \rightarrow D) := \neg(C \cap \neg D)$, $\bot := \neg \top$, $\forall R.C := \neg \exists R.\neg C$.

A terminology (or a TBox) $T$ is a finite set of axioms of the form $C \sqsubseteq D$, where $C, D$ are arbitrary concepts. An ABox $A$ is a finite set of assertions of the form $a : C$ and $a R b$, where $a, b \in \mathcal{IN}$, $C$ is a concept and $R$ a role. Finally, a knowledge base $KB = (T, A)$ consists of a TBox $T$ and an ABox $A$.

Appending the letters $I$, $Q$, $O$, or $H$ to the name of the logic refers to the following syntax extensions:

$\mathcal{I}$: inverse roles: if $R$ is a role, then $R^{-}$ is a role;

$\mathcal{Q}$: qualified number restrictions: if $R$ is a role, $C$ a concept, and $n > 0$, then $(\geq n R.C)$ is a concept; then we can use an abbreviation $\exists R.C := (\geq 1 R.C)$;

$\mathcal{O}$: nominals: if $a \in \mathcal{IN}$, then $\{a\}$ is a concept;

$\mathcal{H}$: role hierarchy: axioms of the form $R \sqsubseteq S$ are allowed in a TBox.

Replacing ‘$\mathcal{ALC}$’ with ‘$\mathcal{S}$’ in the name of a logic refers to allowing transitive roles, i.e., axioms of the form $\text{Trans}(R)$ in a TBox. To maintain decidability, in presence of $\mathcal{S}$, $\mathcal{H}$, and $\mathcal{Q}$ together, the restriction is imposed on the concept syntax: expressions of the form $(\geq n R.C)$ are regarded as well-formed concepts only if the role $R$ has no transitive subroles w.r.t. given TBox (cf. [9]).

**Definition 2. (Semantics)** An interpretation $I = (\Delta^I, \cdot^I)$ consists of non-empty domain $\Delta^I$ and an interpretation function $\cdot^I$ that maps:

- each constant $a \in \mathcal{IN}$ to an element $a^I \in \Delta^I$,
- each concept name $C \in \mathcal{CN}$ to a subset $C^I \subseteq \Delta^I$,
- each role name $R \in \mathcal{RN}$ to a binary relation $R^I \subseteq \Delta^I \times \Delta^I$;

and is extended to all concepts and roles by inductive clauses shown in Fig. 1. The unique name assumption (UNA) is customary made, i.e., only interpretations mapping distinct constants to distinct elements of the domain $\Delta^I$ are considered.

Fig. 1. Semantics for Description Logics.
Definition 3. (Queries) A conjunctive query is an expression of the form
\[
q(\vec{x}) \leftarrow t_1(\vec{x}, \vec{y}) \land \ldots \land t_n(\vec{x}, \vec{y}),
\]
where \(\vec{x}, \vec{y}\) are tuples of (distinguished, resp., non-distinguished) variables, and each atom \(t_i(\vec{x}, \vec{y})\) is of the form \(vC\) (concept atom) or \(vRz\) (role atom), where \(C\) is a concept, \(R\) a role, and \(v, z\) are either variables from \(\vec{x}, \vec{y}\) or constants. The arity of a query \(q(\vec{x})\) is the number of its distinguished variables: \(\ar(q) := |\vec{x}|\). Queries of arity 1 are called unary. Given an interpretation \(I = \langle \Delta, I^\Delta ; \Delta, I^\Sigma \rangle\), a query \(q\) of arity \(m\) is interpreted as follows:
\[
q^I := \{ \vec{c} \in \Delta^m \mid I \models q(\vec{c}, \vec{y}) \land \ldots \land t_n(\vec{c}, \vec{y}) \}.
\]

The answer set of a query \(q(\vec{x})\) w.r.t. a knowledge base \(KB\) is defined as the set of tuples of constants satisfying the query \(q\) in all models of \(KB\):
\[
\text{ans}_{KB}(q) := \{ \vec{a} \in \text{IN} \mid KB \models q(\vec{a}) \}.
\]

Note that the answer set is always finite: \(|\text{ans}_{KB}(q)| \leq |\text{IN}|^{\ar(q)}\), and corresponds, in a model \(I\) of \(KB\), to a subset (a named part) of \(q^I\), i.e., the inclusion \(\text{ans}_{KB}(q)^I \subseteq q^I\) always holds.

It is worth saying that the query answering problem is closely related to the query subsumption problem addressed by many researches \([3, 10]\). Recall that a query \(q(\vec{x})\) subsumes a query \(q'(\vec{x})\) w.r.t. a \(KB\) (written as \(KB \models q \sqsupseteq q'\)) if \(q^I \supseteq q'^I\) holds for every model \(I\) of \(KB\). In fact, these two reasoning problems are reducible to each other. Indeed, a tuple \(\vec{a}\) belongs to the answer set of \(q(\vec{x})\) iff the subsumption \(KB \models \top \sqsupseteq q(\vec{a})\) holds. Conversely, given a subsumption \(KB \models q(\vec{x}) \sqsupseteq q'(\vec{x})\) to be tested, let \(\vec{y}\) be the list of non-distinguished variables in \(q(\vec{x})\). Then we introduce tuples of fresh constants \(\vec{a}_x^r\) and \(\vec{a}_y^r\) corresponding to \(\vec{x}\) and \(\vec{y}\), resp., and define the canonical ABox \(A_q\) for \(q\) as the set of atoms of \(q(\vec{x})\) with constants \(\vec{a}_x^r\) and \(\vec{a}_y^r\) substituted for variables \(\vec{x}\) and \(\vec{y}\):
\[
\text{if } q(\vec{x}) = \bigwedge_{i=1}^n t_i(\vec{x}, \vec{y}) \text{ then } A_q := \{ t_i(\vec{a}_x^r, \vec{a}_y^r) \mid 1 \leq i \leq n \}.
\]

Then it is not hard to show that \(KB \models q \sqsubseteq q'\) iff the tuple \(\vec{a}_x^r\) belongs to the answer set of \(q'(\vec{x})\) w.r.t. the knowledge base \(KB \cup A_q\), i.e., \(KB \cup A_q \models q'(\vec{a}_x^r)\).

1 The existential quantifier \(\exists \vec{y}\) in front of the r.h.s. of the query is customary omitted (and so we did), though implicitly assumed, as follows from the semantics below.
3 Queries answered by concepts

In the remainder of the paper, we will be concerned with unary queries only. The traditional query answering algorithms are usually based on the so called rolling-up technique (see [17] for definitions). In its basic form, when applied to a unary query $q(x)$, it can be viewed as a method of building a concept $C$ that is equivalent to $q(x)$ (in the sense that $q^I = C^I$ for all models $I$ of $KB$), so that to answer the query $q(x)$ is the same as to retrieve all instances of the concept $C$. However, the equivalence of a concept $C$ to a query $q(x)$ is sufficient, but not necessary for query answering purposes. Moreover, a rather simple query $q(x) ← xRx$ is not equivalent to any $ALC$-concept; this can be easily shown using the tree model property for $ALC$. Rolling-up techniques work particularly good for tree-like queries, but face certain problems when applied to cyclic ones (see, e.g., [6]). To overcome these difficulties, we introduce the following main notion of our paper.

Definition 4. A unary query $q(x)$ is answered by a concept $C$ (written as $q(x) \approx C$) if $q(x)$ and $C$ (considered as a query $x:C$) have the same answer sets w.r.t. any knowledge base $KB$: $\text{ans}_{KB}(q) = \text{ans}_{KB}(x:C)$. In other words, $q(x) \approx C$ iff, for any $KB$ and any $a \in IN$, we have: $KB \models q(a) \iff KB \models a:C$.

Notice the quantification over any $KB$ in this definition. Strictly speaking, we should formulate the definition as follows: a query $q(x)$ is answered by a concept $C$ over a description logic $L$ if $\text{ans}_{KB}(q) = \text{ans}_{KB}(x:C)$ for any $KB$ formulated in the logic $L$. However, we will not complicate the matters in this way, since all the results in our paper will have the form: a certain query $q(x)$ is answered by a concept $C$ over any DL $L$ (at least containing the syntax necessary for formulating $C$ and $q$) and even over any first-order theory etc. The following lemma will be useful for us later.

Lemma 5. If the queries $q_1(x) ← \alpha(x,\vec{y})$ and $q_2(x) ← \beta(x,\vec{z})$ are answered by concepts $C$ and $D$, resp., and $\vec{y}, \vec{z}$ are disjoint lists of non-distinguished variables, then the query $q(x) ← \alpha(x,\vec{y}) \land \beta(x,\vec{z})$ is answered by the concept $C \sqcap D$.

Proof. Follows from: $\exists \vec{y} \exists \vec{z} (\alpha(x,\vec{y}) \land \beta(x,\vec{z})) \equiv \exists \vec{y} \alpha(x,\vec{y}) \land \exists \vec{z} \beta(x,\vec{z})$. ⊣

Our task is to determine what kind of queries can be answered by concepts and when these concepts can be found efficiently (and preferably in the same language as the query). We will see that this can be done for a wide range of unary queries, provided that we are allowed to use “fresh” concept names not occurring in KBs and queries under consideration, i.e., specially reserved for query answering purposes. This is a rather customary assumption: in the rolling-up technique, new (“representative”) concept names are also freely introduced (cf. [17]). But, unlike rolling-up, we do not need to add assertions involving these fresh concept names to a $KB$. Fresh concept names will be denoted by capital letters from the end of alphabet ($X$, $Y$, $Z$, etc.), possibly sub- or superscripted.

The remainder of the section is devoted to formulation and proof of the main theorem of our paper. This theorem allows us to transfer results from the modal
correspondence theory to the query answering field. The background information on that theory can be found in [2, Chap. 3]; here we will recall its basic notions.

It is known (cf. [16]) that the DL $\mathcal{ALC}$ with role names $R_1, \ldots, R_n$ is a notational variant of the propositional multi-modal language with modalities $\Box_1, \ldots, \Box_n$, whose formulas are defined inductively: $\bot$ and propositional letters $p_0, p_1, \ldots$ are formulas; if $\varphi$ and $\psi$ are formulas, then so are $\varphi \rightarrow \psi$ and $\Box_1 \varphi$. Exploiting this fact, whenever $\varphi$ is a modal formula, we denote by $C_\varphi$ the concept obtained from $\varphi$ by replacing $\Box_1$ with $\forall R_i$ and $p_j$ with fresh concept name $X_j$. Kripke semantics for the modal language is introduced as usually (cf. [2]). A modal formula $\varphi$ is valid at the point $e$ of a frame $F$ (written as $F,e \models \varphi$) if, for any valuation $\nu$ on $F$, $\varphi$ is true at $e$ in a model $\langle F, \nu \rangle$.

Consider also the first-order language (with equality) having only binary predicate symbols $R_1, \ldots, R_n$. Note that conjunctive queries containing no concept atoms (we will call them relational queries) belong to this first-order language. The following is the key notion in modal correspondence theory.

Definition 6. A first-order formula $\alpha(x)$ with a single free variable locally corresponds to a modal formula $\varphi$ (written as $\alpha(x) \leftrightarrow \varphi$) if, for any frame $F$ and its point $e$, the equivalence holds: $F \models \alpha(e) \iff F,e \models \varphi$.

Now we are ready to prove the theorem that establishes a connection between the notion of local correspondence and the query answering problem for relational queries. Later on, we will extend the family of queries covered by this approach to the ones containing concept atoms as well.

Theorem 7 (Reduction). If a unary relational query $q(x)$ locally corresponds to a modal formula $\varphi$, then the query $q(x)$ is answered by the $\mathcal{ALC}$-concept $C_\varphi$. In symbols: $q(x) \leftrightarrow \varphi \Rightarrow q(x) \approx C_\varphi$.

The proof can be found in the Appendix. This theorem allows us to reuse positive results from the modal correspondence theory for query answering purposes. For example, it implies that the query $q(x) \leftarrow xRx$ is answered by the concept $X \rightarrow \exists R.X$ (where $X$ is a fresh concept name). Other applications will be demonstrated in the subsequent sections. A natural question is whether the converse of this theorem holds; it would enable also to transfer negative results. We have succeeded to prove only a half of the converse, in the following sense (the proof is again in the Appendix).

Lemma 8. Suppose that a unary relational query $q(x)$ is answered by a concept $C_\varphi$, i.e., $q(x) \approx C_\varphi$, for some modal formula $\varphi$. Then for any frame $F$ and its point $e$, the condition $F \models q(e)$ implies $F,e \models \varphi$.

In what follows, we consider only queries with no constants in role atoms, since the general query answering problem is reducible to this case, at the price of introducing nominals in concept atoms: we can equivalently rewrite an atom $yRa$ as $yRz \land z: \{a\}$, for a new non-distinguished variable $z$, and similarly for $aRy$.

$^2$Replacing this implication by equivalence would yield the converse of Theorem 7. After this paper has been accepted, the author has succeeded to prove the converse implication, but only for finitely branching frames.
4 Application 1: Queries and the Sahlqvist fragment

The notion of local correspondence is extensively explored in modal logic. The most prominent in this direction is the result that a wide family of modal formulas (known as Sahlqvist formulas) locally correspond to first-order (FO) conditions, and moreover, there is an algorithm that takes a Sahlqvist formula and returns the corresponding FO condition (see [2, §3.5–3.6] for details; originally in [15]). We will use here a sort of converse of this result, obtained by M. Kracht (see [2, §1.7]; originally in [13]), who described syntactically the class of all those FO formulas (now called Kracht formulas) that locally correspond to Sahlqvist formulas, and provided the corresponding algorithm. We will apply this result to those Kracht formulas that have the form of a (unary relational) conjunctive query and then extend it to queries containing also concept atoms.

First, recall the definition of Kracht formulas. Consider a FO language having only binary relation symbols $R_1, \ldots, R_n$. Formulas of the form

\[ \exists y (xRy \wedge \alpha(y)) \]

and

\[ \forall y (xRy \rightarrow \alpha(y)) \]

will be abbreviated as

\[ (\exists y Rx) \alpha(y) \]

and

\[ (\forall y Rx) \alpha(y), \]

resp. A formula is called clean if no variable occurs both free and bound in it, and no two distinct occurrences of quantifiers bind the same variable. The formulas $\top$, $\perp$, and $xR\sigma y$ are called quasi-atomic, where $\sigma = (i_1, \ldots, i_r)$ is a sequence of indices from $\{1, \ldots, n\}$, and $xR\sigma y$ stands for

\[ \exists z_2 \ldots \exists z_r (xR_{i_1}z_2 \wedge z_2R_{i_2}z_3 \wedge \ldots \wedge z_rR_{i_r}y). \]

Definition 9. A formula is restrictedly positive if it is built up from quasi-atomic ones using conjunction $\wedge$, disjunction $\vee$, and restricted quantifiers $(\exists y Rx)$ and $(\forall y Rx)$. A variable $y$ in a clean formula is called inherently universal if either $y$ is free, or $y$ is bound by a restricted universal quantifier $(\forall y Rx)$ which is not in the scope of any existential quantifier. Finally, a Kracht formula is a clean, restrictedly positive formula such that every of its quasi-atomic subformulas contains at least one inherently universal variable.

For example, $(\forall y Rx)(\forall z Rx)(\exists w Rx)yRz$ is a Kracht formula, whereas the formula $(\exists y Rx)(\forall z Rx)yRz$ is not. Now we present a family of relational queries that belong to the Kracht fragment. We will not prove that this family is exhaustive, in the sense that it covers, modulo equivalence, exactly the intersection of the Kracht fragment and conjunctive queries (although this is likely to be true).

Lemma 10. Any relational conjunctive query of the following form is (equivalent to a Kracht formula and hence) answered by an $\mathcal{ALC}$-concept:

\[ q(x) \leftarrow \exists \bar{y} ( \text{Tree}(x, \bar{y}) \wedge \wedge_{i,j} xR_{i}y_{j} \wedge \wedge_{k,l} y_{k}R_{l}x), \]

where $\text{Tree}(x, \bar{y})$ is a conjunction of role atoms forming an oriented $x$-rooted tree.\(^4\) In other words, $q(x)$ states the existence of an $x$-rooted oriented tree with additional oriented chains linking $x$ with other nodes (in any direction).

\(^3\) This is similar to designating a formula $\exists y (x \leq y \wedge \alpha(y))$ by $(\exists y \geq x) \alpha(y)$.

\(^4\) An oriented tree is a connected oriented graph without cycles (and loops $(e, e)$) s.t. each node has at most one incoming edge. Its root is a node with no incoming edges.
Proof. The conjunct \( \text{Tree}(x, \vec{y}) \) allows to equivalently rewrite the quantifier prefix \( \exists \vec{y} \) into a restricted one:

\[
\exists \vec{y} \left( \text{Tree}(x, \vec{y}) \land \alpha(x, \vec{y}) \right) \equiv (\exists y_1 R_1 y_1') \ldots (\exists y_r R_r y_r') \alpha(x, \vec{y}),
\]

where the condition \( y'_i \in \{x, y_1, \ldots, y_{i-1}\} \) holds, since the formula is clean and \( x \) is the only free variable. It is easily seen that the r.h.s. is a Kracht formula, and hence locally corresponds to a modal formula. Now we apply Theorem 7 to conclude that the query \( q(x) \) is answered by some \( \mathcal{ALC} \)-concept. \( \dagger \)

Next we describe an algorithm for answering these queries; it also captures queries with concept atoms. By Lemma 5, we can consider queries \( q(x) \) without atoms of the form \( x: A \) and \( xRx \), since adding these atoms is equivalent to adding the conjuncts \( A \) and \( X \rightarrow \exists R.X \) to the concept that answers the query.

The algorithm for answering queries within the Kracht fragment

Assume that we are given a conjunctive query of the form

\[
q(x) \leftarrow \exists \vec{y} \left( \text{Tree}(x, \vec{y}) \land \bigwedge_{\alpha(x, \vec{y})} \bigwedge \bigwedge_{\beta(x, \vec{y})} \bigwedge_{\gamma(\vec{y})} \right), \tag{K}
\]

where \( \text{Tree}(x, \vec{y}) \) is a conjunction of role atoms forming an oriented \( x \)-rooted tree; expressions of the form \( yRz \) are role chains linking \( y \) with \( z \); the components of the query are denoted by \( \alpha, \beta, \gamma \) for easy reference. We assume that the conjuncts in the above formula \( (K) \) are distinct. Then we proceed as follows:

1) Introduce fresh concept names: \( Y_{i,j} \) for each conjunct \( xR_\sigma y_j \) in \( \alpha(x, \vec{y}) \), and \( X_{k,\ell} \) for each conjunct \( y_k R_\tau x \) in \( \beta(x, \vec{y}) \).

2) Build two conjunctions \( \text{PRE} \) and \( \text{CON} \) (for ‘premise’ and ‘conclusion’, resp.) by first putting \( \text{PRE} := x: \top \) and \( \text{CON} := y_1: \top \land \ldots \land y_r: \top \) and then proceed as follows (here \( \forall R_\sigma.Y \) stands for \( \forall R_1 \ldots \forall R_{i_r}.Y \), if \( \sigma = (i_1, \ldots, i_r) \), and similarly for \( \exists R_\tau.X \)):
   - for each \( xR_\sigma y_j \) in \( \alpha \), add \( x: \forall R_\sigma, Y_{i,j} \) to \( \text{PRE} \) and \( y_j: Y_{i,j} \) to \( \text{CON} \);
   - for each \( y_k R_\tau x \) in \( \beta \), add \( x: X_{k,\ell} \) to \( \text{PRE} \) and \( y_k: \exists R_\tau, X_{k,\ell} \) to \( \text{CON} \);
   - for each \( y_k R_\tau x \) in \( \gamma \), add the conjunct \( y_k: \exists B_{\ell} \) to \( \text{CON} \).

3) In \( \text{PRE} \) and \( \text{CON} \) obtained, merge conjuncts related to the same variables, i.e., replace \( z: C \land z: D \) by \( z: (C \sqcap D) \). After this, \( \text{PRE} \) will be of the form \( x: C \) and \( \text{CON} \) of the form: \( y_1: D_1 \land \ldots \land y_r: D_r \), for some concepts \( C \) and \( D_i \).

4) Form the expression \( \text{PRE} \rightarrow (\exists \vec{y} (\text{Tree}(x, \vec{y}) \land \text{CON} \land x: \top)) \), or explicitly:

\[
x: C \rightarrow (\exists y_1 R_1 y_1') \ldots (\exists y_r R_r y_r') (y_1: D_1 \land \ldots \land y_r: D_r \land x: \top)
\]

where \( y_i' \in \{x, y_1, \ldots, y_{i-1}\} \). Now apply the standard rolling-up technique to turn the restricted quantifiers \( (\exists y_i R_i y_i') \) into DL-quantifiers, starting from the innermost one. For example, assuming for notational simplicity that \( y_r' \equiv y_1 \) (the case \( y_r' \equiv x \) is similar), the variable \( y_r \) is eliminated as follows:

\[
(\exists y_r R_r y_r') (y_1: D_1 \land \ldots \land y_r: D_r \land x: E) \equiv \quad y_1: (D_1 \sqcap \exists R_r.D_r) \land y_2: D_2 \land \ldots \land y_{r-1}: D_{r-1} \land x: E.
\]
After eliminating all the variables $y_i$, $\text{CON}$ will be of the form $x: D$. Hence the expression considered at the beginning of Item 4 is equivalently transformed into $x: C \rightarrow x: D$. The desired concept answering the query $q(x)$ is $(C \rightarrow D)$, or, in the traditional DL notation, $\neg C \sqcup D$.

Note that the resulting concept belongs to the minimal DL containing the concepts $B_i$ occurring in the original query. In particular, for a relational query we obtain an $\mathcal{ALC}$-concept. We will illustrate how this algorithm works in Ex. 5 (see Sect. 6). The algorithm described above is taken from the general Kracht’s Theorem (see [2, §3.7]) and simplified for the case of FO formulas having the form of conjunctive queries (and slightly modified to capture concept atoms). One can follow the proof of Kracht’s Theorem to show that the algorithm indeed returns a concept that answers the original query. We sum up with the following

Theorem 11. The query answering problem for unary conjunctive queries from the family $\mathcal{K}$ is linearly reduced to the problem of concept instance retrieval (which in turn is reducible to knowledge base unsatisfiability).

5 Application 2: Parallel-serial queries

It is known that the Sahlqvist fragment does not cover all modal formulas having FO correspondents, as well as the Kracht fragment does not contain all FO formulas that locally correspond to modal formulas. In this section we show that the family $\mathcal{K}$ of queries (which are equivalent to some Kracht formulas) does not cover all the conjunctive queries answered by $\mathcal{ALC}$-concepts. This result can be regarded as syntactic extension to Kracht’s theorem. Since we did not prove that the family $\mathcal{K}$ is exactly the restriction of the Kracht fragment to the set of conjunctive queries, we cannot conclude that the queries considered below form a semantically proper extension to the Kracht fragment. At the end of this section we also describe a wider family of queries answered by $\mathcal{ALCI}$-concepts.

First, we define the family of parallel-serial queries (or ps-queries, for short) with two poles $x$ and $y$ (where $x$ is the distinguished and $y$ a non-distinguished variable) inductively. A query $q(x) \leftarrow xRy$ is an atomic ps-query with the poles $x$ and $y$. If $q_1(x) \leftarrow \alpha(x, y, \vec{u})$ and $q_2(x) \leftarrow \beta(x, y, \vec{v})$ are ps-queries with the poles $x$ and $y$ and the tuples $\vec{u}$ and $\vec{v}$ are disjoint, then $q(x) \leftarrow \alpha(x, y, \vec{u}) \land \beta(x, y, \vec{v})$ is a ps-query with the same poles $x$ and $y$ called a parallel connection of $q_1$ and $q_2$ and denoted as $q_1 \parallel q_2$. If $q_1(x) \leftarrow \alpha(x, y, \vec{u})$ is a ps-query with the poles $x$ and $y$, $q_2(y) \leftarrow \beta(y, z, \vec{v})$ is a ps-query with the poles $y$ and $z$, and $\alpha$ and $\beta$ have only $y$ as a common variable, then $q(x) \leftarrow \alpha(x, y, \vec{u}) \land \beta(y, z, \vec{v})$ is a ps-query with the poles $x$ and $z$ called a serial connection of $q_1$ and $q_2$ and denoted as $q_1 \circ q_2$.

To a ps-query $q(x)$ we associate a role expression $\mathcal{R}(q)$ built from role names using intersection and composition: for an atomic query $q(x) \leftarrow xRy$, we put $\mathcal{R}(q) := R$; and inductive steps are: $\mathcal{R}(q_1 \parallel q_2) := \mathcal{R}(q_1) \cap \mathcal{R}(q_2)$, $\mathcal{R}(q_1 \circ q_2) := \mathcal{R}(q_1) \circ \mathcal{R}(q_2)$. By induction, it is straightforward to show that any ps-query $q(x)$ is answered by (and even equivalent to) the $\mathcal{ALC}(\cap, \circ)$-concept $\exists \mathcal{R}(q)$. $\top$ (note that this concept contains no concept names).
Unfortunately, for most DLs used in practice, this solution to query answering is unsatisfactory, since adding role intersection and composition to a logic (together with arbitrary concept and role inclusion axioms in a TBox) leads to undecidability. Therefore, our next aim is to find conditions under which a ps-query is answered by an $\mathcal{ALC}$-concept. For convenience, we extend Definition 4 to the case of two concepts: we say that concepts $C$ and $D$ are similar (written as $C \approx D$) if they have the same instances w.r.t. any knowledge base $\mathcal{KB}$, i.e., $\mathcal{KB} \models a : C \iff \mathcal{KB} \models a : D$, for any $a \in \mathbb{IN}$.

**Lemma 12 (Elimination Lemma).** The concept $\exists (R \cap S).C$ is similar to the concept $\forall R.Y \rightarrow \exists S.(Y \cap C)$ (where $Y$ is a fresh concept name). In general:

$$\exists (R_0 \cap \ldots \cap R_n).C \approx (\forall R_1.Y_1 \cap \ldots \cap \forall R_n.Y_n \rightarrow \exists R_0.(Y_1 \cap \ldots \cap Y_n \cap C)).$$

**Proof.** The concept $\exists (R \cap S).C$ is equivalent to the conjunctive query $q(x) \leftarrow (xRy \land xSy \land x:C)$, which belongs to the family $\mathcal{K}$ from Sect. 4. Applying the algorithm described there, we obtain that $q(x)$ is answered by the concept $\forall R.Y \rightarrow \exists S.(Y \cap C)$. The proof for several roles is completely analogous. $\dashv$

Now we introduce a new family $\mathcal{Z}$ of queries: it consists of queries whose relational part is a ps-query built up from atomic ones using parallel connections and restricted serial connections: $q_1 \circ q_2$ is allowed only if $q_2$ is atomic.

**Theorem 13.** Any query from the family $\mathcal{Z}$ is answered by a concept that can be built in linear time. Relational queries from $\mathcal{Z}$ are answered by $\mathcal{ALC}$-concepts.

The proof is in the Appendix. The families $\mathcal{K}$ and $\mathcal{Z}$ are incomparable w.r.t. inclusion: indeed, the following query belongs to $\mathcal{K} \setminus \mathcal{Z}$:

Query: $q(x) \leftarrow xRy \land yRz \land yRw \land xPz \land xQw$

Concept: $\forall P.Z \cap \forall Q.W \rightarrow \exists R.((\exists R.Z \cap \exists R.W))$

whereas the following one belongs to $\mathcal{Z} \setminus \mathcal{K}$ (but we do not know whether it is equivalent to any Kracht formula beyond the family $\mathcal{K}$):

Query: $q(x) \leftarrow xRy \land xSy \land yPw \land xRz \land xSz \land zQw$

Concept: $\forall R.(Y \cap Z) \cap \forall S.(Y \rightarrow \forall P.W) \rightarrow \exists S.(Z \cap \exists Q.W)$

It is straightforward to “lift” Theorem 13 to modal logic, thus obtaining the statement: any first-order formula $q(x)$ from the family $\mathcal{Z}$ locally corresponds to a modal formula that can be obtained efficiently. This is a syntactic extension to Kracht’s theorem (as already pointed out, we do not know if this is a semantically proper extension). At the same time, this can be regarded as an argument in favour of the (unproved) converse of Theorem 7: the family $\mathcal{Z}$ did not bring us a counterexample to the converse of Theorem 7.

Notice that all relational queries considered so far were answered by $\mathcal{ALC}$-concepts. What if we allow inverse roles? If we “forget” the direction of edges in the queries from the families $\mathcal{K}$ and $\mathcal{Z}$, then we arrive to the following family $\mathcal{Y}$ of queries (formulated in terms of the underlying graph of a query, whose nodes are variables and edges correspond to role atoms): $\mathcal{Y}$ is the family of connected
queries \( q(x) \) without cycles (even non-oriented) consisting of non-distinguished variables only, i.e., every cycle in \( q(x) \) must contain the distinguished variable \( x \).

We conclude with the following theorem (its proof is in the Appendix).

**Theorem 14.** Any query from the family \( \mathcal{Y} \) is answered by a concept that can be built in linear time. Relational queries from \( \mathcal{Y} \) are answered by \( \mathcal{ALC} \)-concepts. The family \( \mathcal{Y} \) includes both families \( \mathcal{K} \) and \( \mathcal{Z} \), i.e., \( \mathcal{K} \cup \mathcal{Z} \subset \mathcal{Y} \).

6 The harvest

Here we present concrete examples of the results obtained. As already mentioned above, we can consider queries \( q(x) \) without an atom of the form \( x:A \).

**Example 1.** Since reflexivity is expressible by the modal formula \( p \to \Diamond p \), we conclude that the query \( q(x) \leftarrow xRx \) is answered (over any DL knowledge base) by the concept \( X \to \exists R.X \). Similarly, the results from Sect. 4 yield the following (cyclic) queries and corresponding concepts:

<table>
<thead>
<tr>
<th>Query ( q )</th>
<th>Concept ( Y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_1(x) \leftarrow xRy \land ySx )</td>
<td>( X \to \exists R.\exists S.X )</td>
</tr>
<tr>
<td>( q_2(x) \leftarrow xRy \land xSy )</td>
<td>( \forall R.Y \to \exists S.Y )</td>
</tr>
<tr>
<td>( q_3(x) \leftarrow xRy \land ySx \land y:B )</td>
<td>( X \to \exists R.(B \cap \exists S.X) )</td>
</tr>
<tr>
<td>( q_4(x) \leftarrow xRy \land xSy \land y:B )</td>
<td>( \forall R.Y \to \exists S.(B \cap Y) )</td>
</tr>
</tbody>
</table>

The query \( q_1(x) \) can be easily generalised to the case of an oriented cycle of the length \( n \) starting and ending at \( x \), and iterating \( \exists R \) in the corresponding concept. Similarly, one can generalise the query \( q_2(x) \), replacing \( R \) and \( S \) by chains of the length \( n \) and \( m \) both going from \( x \) to \( y \), as shown in our next example.

**Example 2.** For any \( m \geq 0 \) and \( n \geq 0 \) with \( m + n > 0 \), the query\(^5\)

\[
q(x) \leftarrow xR_1y_2 \land y_2R_2y_3 \land \ldots \land y_mR_my \land xS_1z_2 \land z_2S_2z_3 \land \ldots \land z_nS_ny
\]

is answered by an \( \mathcal{ALC} \)-concept \( \forall R_1,\ldots,\forall R_m. Y \to \exists S_1,\ldots,\exists S_n. Y \), and the query

\[
q(x) \leftarrow xR_1y_2 \land y_2R_2y_3 \land \ldots \land y_mR_my \land y_2:B_2 \land \ldots \land y_m:B_m \land xS_1z_2 \land z_2S_2z_3 \land \ldots \land z_nS_ny \land z_2:C_2 \land \ldots \land z_n:C_n \land y:D
\]

(with \( B_i, C_j, D \) being concepts in a language \( \mathcal{L} \)) is answered by the \( \mathcal{L} \)-concept

\[
\exists R_1.(B_2 \cap \exists R_2.(B_3 \cap \ldots (B_m \cap \exists R_m.(\neg Y) \ldots )))) \sqcup \\
\exists S_1.(C_2 \cap \exists S_2.(C_3 \cap \ldots (C_n \cap \exists S_n.(Y \cap D) \ldots ))).
\]

The query \( q_2(x) \) in Example 1 can be generalised in another direction: instead of two roles linking \( x \) with \( y \), we can take several ones. This leads to the following example; and it is the first time where we need more than one fresh concept name.

\(^5\) For \( m = 0 \) the first line in \( q \) is dropped and \( y \) replaced by \( x \); and similarly for \( n = 0 \).
Example 3. A query $q(x) \leftarrow xR_0y \wedge \ldots \wedge xR_ny$ is answered by the $\mathcal{ALC}$-concept

$$\forall R_1Y_1 \cap \ldots \cap \forall R_nY_n \rightarrow \exists R_0(Y_1 \cap \ldots \cap Y_n).$$

Now if we stretch each $R_i$ into a chain linking $x$ with $y$, we obtain the following.

Example 4. The query of the form $q(x) \leftarrow q_0(x) \wedge \ldots \wedge q_n(x)$, where each $q_i(x)$ is a chain of $m_i$ edges linking $x$ with $y$:

$$q_i(x) \leftarrow xR_1^iy^i_1 \wedge y^i_2 \wedge \ldots \wedge y^i_{m_i} \wedge R_m^iy^i,$$

(superscripts are indices not powers) is answered by the $\mathcal{ALC}$-concept

$$\forall R_1Y_1 \cap \ldots \cap \forall R_nY_n \rightarrow \exists R_0(Y_1 \cap \ldots \cap Y_n),$$

where $\forall R_i$ stands for $\forall R_1^i \ldots \forall R_{m_i}^i$, and similarly for $\exists R_0$. We leave adding concept atoms to this query as an exercise.

All these queries belong to the family $\mathcal{K}$ from introduced in Sect. 4: in the last example, the chain $q_0(x)$ can be regarded as a $\text{Tree}(x, y)$, while the other chains $q_i(x)$ together constitute the conjunct $\alpha(x, y)$. An example of the query from the family $\mathcal{Z}$ (to be more exact, from $\mathcal{Z} \setminus \mathcal{K}$) was given after Theorem 13. Now we will illustrate how the algorithm presented in Sect. 4 works.

Example 5. Consider the query $q(x) \leftarrow \text{Tree}(x, y) \wedge \alpha(x, y) \wedge \beta(x, y)$, where

$$\text{Tree}(x, y) \equiv xR_1y_1 \wedge y_1R_2y_2 \wedge y_1R_3y_3 \wedge y_1R_4y_4 \wedge y_4R_5y_5 \wedge y_4R_6y_6,$$

$$\alpha(x, y) \equiv xS_1y_1 \wedge xS_4y_6,$$

$$\beta(x, y) \equiv y_2S_2x \wedge y_6S_3x.$$
7 Conclusions and outlook

One of the achievements of this paper is the established relationship between the notion of local correspondence from modal logic and the notion ‘a concept answers a query’ (Theorem 7). As an application, this enabled us to reuse the results from modal correspondence theory for answering a wide range of conjunctive queries with one distinguished variable. We have also found a syntactic (and probably semantic) extension to Kracht’s theorem. Please note that, although at the first glance, only $\mathcal{ALC}$ or $\mathcal{ALCI}$ occur explicitly in Theorems 11, 13, 14, these should not be misunderstood as giving a query answering method for $\mathcal{ALC}$ or $\mathcal{ALCI}$ knowledge bases only. In fact, as we pointed out earlier, the results obtained provide us with algorithms for query answering over KBs formulated in any DL (extending $\mathcal{ALC}$), using as minimal means as possible (i.e., the resulting concept belongs to a simple logic $\mathcal{ALC}$ or $\mathcal{ALCI}$).

Analysing these families of queries, one can observe that the inherent complexity of query answering lies in the relational structure (i.e., the underlying graph) of the query. This structure was captured by concepts of $\mathcal{ALC}$ (Theorems 11 and 13) or $\mathcal{ALCI}$ (Theorem 14). A natural extension would be to invoke into this framework the correspondence theory for richer modal logics. The usage of qualified number restrictions (i.e., $\mathcal{ALCQ}$-concepts) for answering relational queries would involve the correspondence theory for graded modal logic. However, to the best of our knowledge, no general results similar to Sahlqvist’s theorem are known for graded modal logic. Another possible direction is to extend this technique to DLs with relations of arbitrary arity, thus involving the correspondence theory for polyadic modal logics (which is explored extensively).

The need to investigate extensions stems from the restricted applicability of the technique we developed so far. For example, we have not succeeded to find an $\mathcal{ALC}$-concept answering the query $q(x) \leftarrow xRy \land ySy$. Moreover, if the converse of Theorem 7 holds (which is still an open question), then it would imply that such an $\mathcal{ALC}$-concept does not exist, since the property of frames expressed by $q(x)$ is not modally definable (see [2, Chap. 3], discussion after Corollary 3.16). At the same time, this and other queries shown in the table below can be answered by concepts involving extra role operations (whether they can be answered by any $\mathcal{ALCQI}$-concept is unknown):

<table>
<thead>
<tr>
<th>Query</th>
<th>Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q(x) \leftarrow xRy \land ySy$</td>
<td>$\exists R \exists (S \cap id(\top)) \uparrow$</td>
</tr>
<tr>
<td>$q(x) \leftarrow xRy \land ySz \land yPz$</td>
<td>$\exists R \exists (S \cap P) \uparrow$</td>
</tr>
<tr>
<td>$q(x) \leftarrow xRy \land xSz \land yPz \land zQy$</td>
<td>$\forall S \exists Z \exists R \exists (P \cap (Q^{-})).Z$</td>
</tr>
<tr>
<td>$q(x) \leftarrow xRyR^{<em>}w \land xSzS^{</em>}w \land yPz$</td>
<td>$\forall S \exists Z \exists R \exists (P \cap (R^{<em>} \circ S^{</em>})).Z$</td>
</tr>
</tbody>
</table>

Finally, it is interesting whether all conjunctive queries can be answered by some concepts. As an example, for the following “tetrahedron” query

$$q(x) \leftarrow xRyR^{*}w \land xSzS^{*}w \land yPz \land xQw$$

we have neither found a concept in any DL that would answer it, nor proved that such a concept (in a certain DL) does not exist.
Acknowledgements

The research has been supported by EPSRC, Grant No. GR/S63168/01. The author would like to thank Dr. Ulrike Sattler and Dr. Ian Horrocks for guiding him in Description Logic and the help during the research. The author is also grateful to Birte Glimm for fruitful discussions on the subject of query answering and some useful suggestions.

References

will prove the following equivalence:

<table>
<thead>
<tr>
<th>Containing</th>
<th>$\text{ALC}$</th>
</tr>
</thead>
</table>

Suppose that a unary relational query $q(x)$ is answered by the $\text{ALC}$-concept $C_\varphi$. In symbols: $q(x) \iff \varphi \implies q(x) \approx C_\varphi$.

**Proof.** Suppose that $q(x) \iff \varphi$. Then, given a knowledge base $\mathcal{KB}$ (in any DL containing $\text{ALC}$, see the discussion after Definition 4) and a constant $a \in \mathbb{N}$, we will prove the following equivalence: $\mathcal{KB} \models q(a) \iff \mathcal{KB} \models a : C_\varphi$.

($\implies$) Take any model $\mathcal{I}$ of $\mathcal{KB}$. By assumption, $\mathcal{I} \models q(a)$. We need to show that $\mathcal{I} \models a : C_\varphi$ (independently of how the fresh concept names occurring in $C_\varphi$ are interpreted in $\mathcal{I}$). Let $F$ be the frame underlying $\mathcal{I}$ and denote $e := a^F$. By definition, from $q(x) \iff \varphi$ it follows that, for these $F$ and $e$, we have: $F \models q(e) \iff F, e \models \varphi$. But we also know that $F \models q(e)$, because $\mathcal{I} \models q(a)$ and $q(x)$ is relational. Hence we conclude that $F, e \models \varphi$, i.e., $e \in \varphi^{\nu}$ for any valuation $\nu$.

Now take the valuation $\nu$ that is “read-off” from our interpretation $\mathcal{I}$ by putting $p_i^\nu := X_i^\mathcal{I}$, for all propositional letters $p_i$ occurring in the formula $\varphi$. It is easily seen that $\varphi^{\nu} = C_\varphi^{\mathcal{I}}$, since $C_\varphi$ is just a notational variant of $\varphi$, whereas $\nu$ and $\mathcal{I}$ are essentially the same. As shown above, $e \in \varphi^{\nu}$ for this valuation $\nu$ and so $a^F = e \in \varphi^{\nu} = C_\varphi^{\mathcal{I}}$. Thus we have proven that $\mathcal{I} \models a : C_\varphi$.

($\impliedby$) Take any model $\mathcal{I}$ of $\mathcal{KB}$. By assumption, $\mathcal{I} \models a : C_\varphi$. Let $F$ be a frame underlying $\mathcal{I}$ and $e := a^F$. We need to show that $\mathcal{I} \models q(a)$; since $q(x)$ is relational, it suffices to show that $F \models q(e)$. Due to the assumption that $q(x) \iff \varphi$, it remains to show that $F, e \models \varphi$.

To this end, take any valuation $\nu$ and show that $e \in \varphi^{\nu}$. Let $\mathcal{J}$ be an interpretation differing from $\mathcal{I}$ only in how it interprets the fresh concept names occurring in $C_\varphi$, namely it is “read-off” from the valuation $\nu$ by putting $X_i^\mathcal{J} := p_i^\nu$. Observe that $C_\varphi^{\mathcal{J}} = \varphi^{\nu}$. Since $\mathcal{J}$ and $\mathcal{I}$ agree on concept, role and individual names occurring in $\mathcal{KB}$, we conclude that $\mathcal{J} \models \mathcal{KB}$. Now we use our assumption $\mathcal{KB} \models a : C_\varphi$ to infer that $\mathcal{J} \models a : C_\varphi$, from which it follows that $e = a^F = a^\mathcal{J} \in C_\varphi^{\mathcal{J}} = \varphi^{\nu}$.

This completes the proof of Theorem 7.

**Lemma 8.** Suppose that a unary relational query $q(x)$ is answered by a concept $C_\varphi$, i.e., $q(x) \approx C_\varphi$, for some modal formula $\varphi$. Then for any frame $F$ and its point $e$, the condition $F \models q(e)$ implies $F, e \models \varphi$.  

---


**Appendix A: Proofs**

**Theorem 7 (Reduction).** If a unary relational query $q(x)$ locally corresponds to a modal formula $\varphi$, then the query $q(x)$ is answered by the $\text{ALC}$-concept $C_\varphi$. In symbols: $q(x) \iff \varphi \implies q(x) \approx C_\varphi$.

**Proof.** Suppose that $q(x) \iff \varphi$. Then, given a knowledge base $\mathcal{KB}$ (in any DL containing $\text{ALC}$, see the discussion after Definition 4) and a constant $a \in \mathbb{N}$, we will prove the following equivalence: $\mathcal{KB} \models q(a) \iff \mathcal{KB} \models a : C_\varphi$.

($\implies$) Take any model $\mathcal{I}$ of $\mathcal{KB}$. By assumption, $\mathcal{I} \models q(a)$. We need to show that $\mathcal{I} \models a : C_\varphi$ (independently of how the fresh concept names occurring in $C_\varphi$ are interpreted in $\mathcal{I}$). Let $F$ be the frame underlying $\mathcal{I}$ and denote $e := a^F$. By definition, from $q(x) \iff \varphi$ it follows that, for these $F$ and $e$, we have: $F \models q(e) \iff F, e \models \varphi$. But we also know that $F \models q(e)$, because $\mathcal{I} \models q(a)$ and $q(x)$ is relational. Hence we conclude that $F, e \models \varphi$, i.e., $e \in \varphi^{\nu}$ for any valuation $\nu$.

Now take the valuation $\nu$ that is “read-off” from our interpretation $\mathcal{I}$ by putting $p_i^\nu := X_i^\mathcal{I}$, for all propositional letters $p_i$ occurring in the formula $\varphi$. It is easily seen that $\varphi^{\nu} = C_\varphi^{\mathcal{I}}$, since $C_\varphi$ is just a notational variant of $\varphi$, whereas $\nu$ and $\mathcal{I}$ are essentially the same. As shown above, $e \in \varphi^{\nu}$ for this valuation $\nu$ and so $a^F = e \in \varphi^{\nu} = C_\varphi^{\mathcal{I}}$. Thus we have proven that $\mathcal{I} \models a : C_\varphi$.

($\impliedby$) Take any model $\mathcal{I}$ of $\mathcal{KB}$. By assumption, $\mathcal{I} \models a : C_\varphi$. Let $F$ be a frame underlying $\mathcal{I}$ and $e := a^F$. We need to show that $\mathcal{I} \models q(a)$; since $q(x)$ is relational, it suffices to show that $F \models q(e)$. Due to the assumption that $q(x) \iff \varphi$, it remains to show that $F, e \models \varphi$.

To this end, take any valuation $\nu$ and show that $e \in \varphi^{\nu}$. Let $\mathcal{J}$ be an interpretation differing from $\mathcal{I}$ only in how it interprets the fresh concept names occurring in $C_\varphi$, namely it is “read-off” from the valuation $\nu$ by putting $X_i^\mathcal{J} := p_i^\nu$. Observe that $C_\varphi^{\mathcal{J}} = \varphi^{\nu}$. Since $\mathcal{J}$ and $\mathcal{I}$ agree on concept, role and individual names occurring in $\mathcal{KB}$, we conclude that $\mathcal{J} \models \mathcal{KB}$. Now we use our assumption $\mathcal{KB} \models a : C_\varphi$ to infer that $\mathcal{J} \models a : C_\varphi$, from which it follows that $e = a^F = a^\mathcal{J} \in C_\varphi^{\mathcal{J}} = \varphi^{\nu}$.

This completes the proof of Theorem 7.

**Lemma 8.** Suppose that a unary relational query $q(x)$ is answered by a concept $C_\varphi$, i.e., $q(x) \approx C_\varphi$, for some modal formula $\varphi$. Then for any frame $F$ and its point $e$, the condition $F \models q(e)$ implies $F, e \models \varphi$.  

---
Proof. Assume that $F \models q(e)$. The query $q(x)$ has the form $\exists \bar{y} \bigwedge_{i=1}^{m} t_i(x, \bar{y})$, where $t_i$ are role atoms. Then there exist $\bar{o} \in \Delta$ such that $F \models t_i(e, \bar{o})$ for all $i \leq m$. Now, take the canonical ABox $A_q$ for $q(x)$, i.e., introduce new constants $a_x$ and $a_\bar{y}$ and put $A_q := \{ t_i(a_x, a_\bar{y}) \mid 1 \leq i \leq m \}$, and consider a knowledge base $KB_q := (\emptyset, A_q)$. Since $KB_q \models \bigwedge_{i=1}^{m} t_i(a_x, a_\bar{y})$, we have $KB_q \models \exists \bar{y} \bigwedge_{i=1}^{m} t_i(a_x, \bar{y})$, and hence $KB_q \models q(a_x)$. Applying the condition of our Lemma $q(x) \approx C_\varphi$, we obtain that $KB_q \models a_x: C_\varphi$.

To prove that $F, e \models \varphi$, take an arbitrary valuation $\nu$ on the frame $F$ and show that $e \in C_\varphi$. Let $I$ be an interpretation based on $F$ such that it is “read-off” from $\nu$ by putting $X^I_i := p^I_e$, for all fresh concept names $X_i$ occurring in $C_\varphi$, and extended to the new constants by putting $a^I_x := e$ and $a^I_\bar{y} := \bar{o}$. Since $I$ is based on $F$ and $F \models t_i(a, \bar{o})$, we have $I \models t_i(a_x, a_\bar{y})$, for all $i \leq m$, and hence $I \models KB_q$.

As shown above, $KB_q \models a_x: C_\varphi$. Therefore, we conclude that $I \models a_x: C_\varphi$ and finally $e \in C^I_\varphi = C_\varphi$.

Theorem 13. Any query from the family $Z$ is answered by a concept that can be built in linear time. Relational queries from $Z$ are answered by $\mathcal{ALC}$-concepts.

Proof. First assume that $q(x)$ is a relational query from $Z$. Then:

1) Build the role expression $R(q)$ (see Sect. 5). As already pointed out, the query $q(x)$ is equivalent to the $\mathcal{ALC}($ and $\circ)-$concept $\exists R(q)$. $\top$.

2) Starting from the concept $\exists R(q). \top$, repeatedly apply the following rewriting rules (each application of the second rule introduces a new concept name $X$), until all role operations are eliminated and we arrive to an $\mathcal{ALC}$-concept that answers the query $q(x)$:

$$
D \sqcup \exists (R \circ S). C \quad \equiv \quad D \sqcup \exists R. \exists S. C
$$

$$
D \sqcup \exists (R \sqcap S). C \quad \equiv \quad D \sqcup \exists R. \neg X \sqcup \exists S. (X \sqcup C)
$$

The second rule follows from the Elimination Lemma and the following observation: if $C \approx D$ then $(C \sqcup E) \approx (D \sqcup E)$. Since, in the definition of the family $Z$, a serial connection $q_1 \circ q_2$ is allowed only if $q_2$ is atomic, each time the first rule is applied, $S$ will be a role name. Hence we will never obtain an expression of the form $\exists (R \cap S). C$ inside another quantifier, and so will be able to eliminate all occurrences of role intersection using the second rule. Therefore, the algorithm always terminates and produces an $\mathcal{ALC}$-concept $C$ such that $C \approx \exists R(q). \top$.

It is easily seen that the procedure works in linear time in size of $q(x)$.

Now suppose that $q(x)$ belongs to $Z$ and contains concept atoms. Let $y: C$ be one of them, where $y$ is a non-distinguished variable (as already mentioned above, we can always consider queries $q(x)$ without atoms of the form $x: A$). Then we introduce a new non-distinguished variable $y'$ and replace the atom $y: C$ by $y(id(C))y'$ and each atom of the form $yRz$ by $y'Rz$ (recall that $id(C)$ is a notation that goes back to the propositional dynamic logic, and it denotes the role with the semantics: $id(C)^x := \{(e, e) \mid e \in C^x\}$). This can be done for all concept atoms in $q(x)$ and results in an equivalent relational query $q'(x)$ (in an extended language) that belongs to $Z$. Now we apply the above algorithm and finally replace in the resulting concept each occurrence of $\exists id(C). D$ by the equivalent expression $(C \sqcap D)$.

$\square$
Theorem 14. Any query from the family $\mathcal{Y}$ is answered by a concept that can be built in linear time. Relational queries from $\mathcal{Y}$ are answered by $\mathcal{ALCI}$-concepts. The family $\mathcal{Y}$ includes both families $\mathcal{K}$ and $\mathcal{Z}$, i.e., $\mathcal{K} \cup \mathcal{Z} \subset \mathcal{Y}$.

Proof. Every query $q(x)$ in $\mathcal{K} \cup \mathcal{Z}$ is connected. Furthermore, if we remove from its graph the node $x$ together with the edges incident to $x$, we will obtain an acyclic graph. This shows that $q(x) \in \mathcal{Y}$, hence the inclusion $\mathcal{K} \cup \mathcal{Z} \subset \mathcal{Y}$.

Now suppose we are given a query $q(x) \in \mathcal{Y}$. We will give a sketch of the proof that $q(x)$ can be represented as a query from “non-oriented” version of the family $\mathcal{K}$ (i.e., with edges oriented arbitrarily). First remove from its graph the node $x$ together with the edges incident to $x$. This yields an acyclic non-oriented graph, i.e., a forest (a finite set of trees). Since the original graph (before removing $x$) was connected, it had edges linking $x$ with the trees in this forest; take one edge per tree. Then the node $x$ together with this forest and edges linking $x$ with the trees in this forest form a non-oriented tree $\text{Tree}(x, y)$. Other edges can be considered as chains linking $x$ with some non-distinguished variables. Thus, we have obtained a representation of the query $q(x)$ in the “non-oriented” version of $(\mathcal{K})$. Applying the algorithm described in Sect. 4 yields an $\mathcal{ALCI}$-concept that answers our query. ⊣