Annals of Mathematics and Artificial Intelligence manuscript No.
(will be inserted by the editor)

Exploiting Conjunctive Queries in Description Logic Programs

Thomas Eiter · Giovambattista Ianni · Thomas Krennwallner · Roman Schindlauer

Received: date / Accepted: date

Abstract Towards combining rules and ontologies for the Semantic Web, nonmonotonic Description Logic Programs (dl-programs) have been proposed as a powerful formalism to couple nonmonotonic logic programming and Description Logic reasoning on a clear semantic basis. In this paper, we present cq-programs, which enhance dl-programs with conjunctive queries (CQ) and union of conjunctive queries (UCQ) over Description Logics knowledge bases, as well as with disjunctive rules. The novel formalism has two advantages. First, it offers increased expressivity because it allows for (U)CQs in the bodies of the rules. The (U)CQs allow one to access unnamed individuals in the rules and they increase the expressivity of the formalism, as evident from the increase in complexity from NEXP to 2-EXP. And second, when implemented as a combination between a logic programming system and a DL-reasoner, this integration of rules and ontologies gives rise to strategies for optimizing calls to the DL-reasoner, by exploiting specific support for (U)CQs. To this end, we present equivalence preserving transformations which can be used for program rewriting, and we present respective generic rewriting algorithms. Experimental results for a cq-program prototype show that this can lead to significant performance improvements, and suggest that cq-programs and program rewriting provide a useful basis for dl- and cq-program optimization.

Thomas Eiter
Institut für Informationssysteme 184/3, Technische Universität Wien
Favoritenstraße 9-11, A-1040 Vienna, Austria

Giovambattista Ianni
Dipartimento di Matematica, Università della Calabria,
I-87036 Rende (CS), Italy.

Thomas Krennwallner
Institut für Informationssysteme 184/3, Technische Universität Wien
Favoritenstraße 9-11, A-1040 Vienna, Austria

Roman Schindlauer
Institut für Informationssysteme 184/3, Technische Universität Wien
Favoritenstraße 9-11, A-1040 Vienna, Austria

E-mail: {eiter,ianni,tkren,roman}@kr.tuwien.ac.at

1 Introduction

Rule formalisms that combine logic programming with other sources of knowledge, especially terminological knowledge expressed in Description Logics (DLs), have gained increasing interest in the past years. This process was mainly fostered by current efforts in the Semantic Web development of designing a suitable rules layer on top of the existing ontology layer. Such couplings between DLs (in the form of ontologies) and logic programming appear in different flavors, which roughly can be categorized in (i) systems with full semantic integration, (ii) systems with strict semantic integration, and (iii) systems with strict semantic separation, which amounts to coupling heterogeneous systems [1,8,33,34]. In this paper, we will concentrate on the latter, considering ontologies as an external source of information with semantics that are independent from the logic program. One representative of this category was presented in [8,10], extending the answer-set semantics of logic programs towards so-called *dl-programs*, which have been conceived to couple existing reasoning engines for nonmonotonic logic programming and for Description Logics, respectively, in a meaningful way despite all syntactic and semantic mismatches between the underlying formalisms.

A dl-program consists of a DL part L and a rule part P, and allows queries from P to L. These queries are facilitated by a special type of atoms, which also permit to hypothetically enlarge the assertional part of L with facts imported from the logic program P, thus allowing for a bidirectional flow of information.

The types of queries expressible by dl-atoms in [8,10] are concept and role membership queries, as well as subsumption queries. Since the semantics of logic programs is usually defined over a domain of explicit individuals, this approach may fail to derive certain consequences, which are implicitly contained in L. This is illustrated by the following example.

Example 1 Consider the following simplified version of a scenario in [28].

$$L = \left\{ \begin{array}{l} \mathit{hates}(\mathit{Cain}, \mathit{Abel}), \mathit{hates}(\mathit{Romulus}, \mathit{Remus}), \\ \mathit{father}(\mathit{Cain}, \mathit{Adam}), \mathit{father}(\mathit{Abel}, \mathit{Adam}), \\ \mathit{father} \sqsubseteq \mathit{parent}, \\ \exists \mathit{father}.\exists \mathit{father}^{-}.\{\mathit{Remus}\}(\mathit{Romulus}) \end{array} \right\}$$

$$P = \{\mathit{BadChild}(X) \leftarrow \mathrm{DL}[\mathit{parent}](X, Z), \mathrm{DL}[\mathit{parent}](Y, Z), \mathrm{DL}[\mathit{hates}](X, Y)\}$$

Apart from the explicit facts, L states that each *father* is also a *parent* and that *Romulus* and *Remus* have a common father. The single rule in P specifies that an individual hating a sibling is a *BadChild*. From this dl-program, $\mathit{BadChild}(\mathit{Cain})$ can be concluded, but not $\mathit{BadChild}(\mathit{Romulus})$.

The reason is that, in a dl-program, variables must be instantiated over its Herbrand base (containing the individuals in L and P), and thus unnamed individuals, like the father of Romulus and Remus, are not considered. In essence, this means that dl-atoms only allow for building conjunctive queries that are *DL-safe* in the spirit of [28], which ensures that all variables in the query can be instantiated to named individuals. While DL-safeness was mainly motivated by retaining decidability of the formalisms, unsafe conjunctive queries are admissible under certain conditions [34]. In this vein, we extend dl-programs by permitting conjunctive queries or unions thereof (respectively, CQs and UCQs in the following), to L as first-class citizens in the language.

Example 2 In the example above, we may use

$$P' = \{BadChild(X) \leftarrow \mathrm{DL}[parent(X,Z), parent(Y,Z), hates(X,Y)](X,Y)\},$$

where the body of the rule is a CQ $\{parent(X,Z), parent(Y,Z), hates(X,Y)\}$ to L with distinguished variables X and Y. We then obtain the desired result; that is, we can derive the fact $BadChild(Romulus)$.

The extension of dl-programs to cq-programs, introduced in this paper, has some attractive features.

- First and foremost, the expressiveness of the formalism is significantly increased, since existentially quantified and therefore unnamed individuals can be respected in query answering with the help of (u)cq-atoms.
- In addition, cq-programs have the nice feature that the integration of rules and the ontology is decidable whenever answering (U)CQs over the ontology (possibly extended with assertions) is decidable. In particular, recent results on the decidability of answering (U)CQs for expressive DLs can be exploited in this direction [14,30,31]. Furthermore, it also allows us to express, via conjunction of cq-atoms and negated cq-atoms in rule bodies, certain decidable conjunctive queries with negations; note that negation quickly leads to undecidability [36].
- The availability of CQs opens the possibility to express joins in different, equivalent ways and therefore to the design of a software component, which employs automatic rewriting techniques. Such rewriting component, starting from a given program (L, P), might produce an equivalent, yet more efficient, program (L, P').

Example 3 Both

$$r : BadParent(Y) \leftarrow \mathrm{DL}[parent](X,Y), \mathrm{DL}[hates](Y,X)$$

and

$$r' : BadParent(Y) \leftarrow \mathrm{DL}[parent(X,Y), hates(Y,X)](X,Y)$$

equivalently single out (not necessarily all) bad parents. Here, in r the join between *parent* and *hates* is performed in the logic program, while in r' it is performed on the DL-side.

DL-reasoners including RACER, KAON2, and Pellet increasingly support answering CQs. This can be exploited to push joins of multiple atoms from the rule part to the DL-reasoner, or vice versa. Multiple calls to the DL-reasoner are an inherent bottleneck in evaluating cq-programs. Reducing the number of calls can significantly improve performance of reasoning.

Motivated by the last aspect, we then focus on the following contributions.

- We present a suite of equivalence-preserving transformation rules, by which rule bodies and rules involving (u)cq-atoms can be rewritten. Based on these rules, we then describe algorithms which transform a given cq-program P into an equivalent, optimized cq-program P'.
- We report an experimental evaluation of such rewriting techniques, based on a prototype implementation of cq-programs using dlvhex [9,38] and RACER. It shows the effectiveness of the techniques, and that significant performance increases can be gained. The experimental results are interesting in their own right, since they shed light on combining conjunctive query results from a DL-reasoner.

• We have implemented a prototype reasoner for cq-programs and present its architecture in Section 7. To the best of our knowledge, it is currently the most expressive implementation of a system integrating nonmonotonic rules and ontologies.
• Furthermore, we analyze the computational complexity of cq-programs and show that they have higher complexity than dl-programs; already for the description logic $\mathcal{SHIF}(\mathbf{D})$, which underlies the OWL-Lite standard, deciding the existence of an answer set is 2-EXP-complete, as compared to the $\mathrm{P}^{\mathrm{NEXP}}$-completeness of the problem for dl-programs. Thus, cq-programs are a more expressive formalism for representing problems than dl-programs from a computational perspective.

The rest of the paper is structured as follows. The next section recalls concepts of Description Logics. In Section 3, we formally define cq-programs and consider some elementary semantic properties, while in Section 4 we consider their computational complexity. After that, we present in Section 5 a suite of equivalence preserving rewriting rules, which are used by a generic rewriting algorithm that is given in Section 6. Experimental results for a prototype implementation are reported in Section 7. Section 8 concludes the paper with a discussion of related work and further issues.

2 Description Logics

In this section, we recall the Description Logics (DLs) $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$, which provide the logical underpinning of the Web ontology languages OWL-Lite and OWL-DL, respectively (see [2,16,18] for further details and background on DLs).[1] Intuitively, DLs model a domain of interest in terms of concepts and roles, which represent classes of individuals and binary relations on classes of individuals, respectively. A DL-knowledge base encodes on the one hand concept and role hierarchies, i.e., subset relationships between classes of individuals and between binary relations on classes of individuals. On the other hand, such knowledge bases may express membership assertions of individuals to classes, and membership of pairs of individuals to roles. Other important ingredients of $\mathcal{SHIF}(\mathbf{D})$ (resp., $\mathcal{SHOIN}(\mathbf{D})$) are datatypes (resp., datatypes and individuals) in concept expressions.

We first describe the syntax of $\mathcal{SHOIN}(\mathbf{D})$, which has the following datatypes and basic building blocks. We assume a set $\mathbf{E}$ of *elementary datatypes* and a set $\mathbf{V}$ of *data values*. A *datatype theory* $\mathbf{D} = (\Delta^{\mathbf{D}}, \cdot^{\mathbf{D}})$ consists of a *datatype* (or *concrete*) *domain* $\Delta^{\mathbf{D}}$ and a mapping $\cdot^{\mathbf{D}}$ that assigns to every elementary datatype a subset of $\Delta^{\mathbf{D}}$ and to every data value an element of $\Delta^{\mathbf{D}}$. Let $\Psi = (\mathbf{A} \cup \mathbf{R}_A \cup \mathbf{R}_D, \mathbf{I} \cup \mathbf{V})$ be a vocabulary, where $\mathbf{A}$, $\mathbf{R}_A$, $\mathbf{R}_D$, and $\mathbf{I}$ are pairwise disjoint (denumerable) sets of *atomic concepts*, *abstract roles*, *datatype* (or *concrete*) *roles*, and *individuals*, respectively. We denote by $\mathbf{R}_A^-$ the set of inverses R^- of all $R \in \mathbf{R}_A$.

Roles and concepts are defined as follows. A *role* is an element of $\mathbf{R}_A \cup \mathbf{R}_A^- \cup \mathbf{R}_D$. *Concepts* are inductively defined as follows. Every atomic concept $C \in \mathbf{A}$ is a concept. If $o_1, o_2, \ldots$ are individuals from $\mathbf{I}$, then $\{o_1, o_2, \ldots\}$ is a concept (called *oneOf*). If C and D are concepts, then also $(C \sqcap D)$, $(C \sqcup D)$, and $\neg C$ are concepts (called *conjunction*, *disjunction*, and *negation*, respectively). If C is a concept, R is an abstract role from $\mathbf{R}_A \cup \mathbf{R}_A^-$, and n is a nonnegative integer, then $\exists R.C$, $\forall R.C$, $\geq nR$, and $\leq nR$ are concepts (called *exists*, *value*, *atleast*, and *atmost restriction*, respectively). If D is

[1] We focus on these DLs because of the importance of the OWL standard. Conceptually, cq-programs can be defined for other DLs as well with little change.

a datatype, U is a datatype role from $\mathbf{R}_D$, and n is a nonnegative integer, then $\exists U.D$, $\forall U.D$, $\geq nU$, and $\leq nU$ are concepts (called *datatype exists*, *value*, *atleast*, and *atmost restriction*, respectively).

We next define axioms and knowledge bases as follows. An *axiom* is an expression of one of the following forms:

1. $C \sqsubseteq D$, called *concept inclusion axiom*, where C and D are concepts;
2. $R \sqsubseteq S$, called *role inclusion axiom*, where either $R, S \in \mathbf{R}_A$ or $R, S \in \mathbf{R}_D$;
3. Trans(R), called *transitivity axiom*, where $R \in \mathbf{R}_A$;
4. $C(a)$, called *concept membership axiom*, where C is a concept and $a \in \mathbf{I}$;
5. $R(a, b)$ (resp., $U(a, v)$), called *role membership axiom*, where $R \in \mathbf{R}_A$ (resp., $U \in \mathbf{R}_D$) and $a, b \in \mathbf{I}$ (resp., $a \in \mathbf{I}$ and v is a data value); and
6. $a = b$ (resp., $a \neq b$), or $= (a, b)$ (resp., $\neq (a, b)$), called *equality* (resp., *inequality*) *axiom*, where $a, b \in \mathbf{I}$.

The syntax of $\mathcal{SHIF}(\mathbf{D})$ is the one of $\mathcal{SHOIN}(\mathbf{D})$, but without the oneOf constructor and with the atleast and atmost constructors limited to 0 and 1.

Definition 1 A ($\mathcal{SHOIN}(\mathbf{D})$) *DL knowledge base* (DL-KB) L is a finite set of axioms. It is in $\mathcal{SHIF}(\mathbf{D})$, if all its axioms are from $\mathcal{SHIF}(\mathbf{D})$.

In the introductory Example 1, for instance, we have a DL-KB L which has four role membership axioms, one concept membership axiom, and one concept inclusion axiom; L is not in $\mathcal{SHIF}(\mathbf{D})$, since the last axiom involves the oneOf constructor.[2] The DL-KB L given in Example 8 below, however, is in $\mathcal{SHIF}(\mathbf{D})$.

The semantics of a DL-KB L is given in terms of first-order interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ with respect to a datatype theory $\mathbf{D} = (\Delta^{\mathbf{D}}, \cdot^{\mathbf{D}})$ (alternatively, it can be given by a mapping $\pi(L)$ of L to first-order logic, cf. [2]). It consists of a nonempty (*abstract*) *domain* $\Delta^{\mathcal{I}}$ disjoint from $\Delta^{\mathbf{D}}$, and a mapping $\cdot^{\mathcal{I}}$ that assigns to each $C \in \mathbf{A}$ a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, to each $o \in \mathbf{I}$ an element $o^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, to each $R \in \mathbf{R}_A$ a subset $R^{\mathcal{I}}$ of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and to each $U \in \mathbf{R}_D$ a subset $U^{\mathcal{I}}$ of $\Delta^{\mathcal{I}} \times \Delta^{\mathbf{D}}$; the mapping is extended to all concepts and roles as usual.

The interpretation $\mathcal{I}$ is a *model* of a L, if it satisfies each axiom α in L, where satisfaction $\mathcal{I} \models \alpha$ is defined as usual, cf. [2,16,18]. An axiom α is a logical consequence of L, denoted $L \models \alpha$, if $\mathcal{I} \models \alpha$ for each model $\mathcal{I}$ of L.

A DL-KB L is satisfiable, if L has some model. To gain decidability of this problem, number restrictions in L are restricted to so called *simple* abstract roles [17]; as for computability, we tacitly assume that DL-KBs fulfill this condition.

Example 4 The DL-KB L in Example 1 is clearly satisfiable; e.g., the interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ such that $\Delta^{\mathcal{I}} = \{Cain, Abel, Adam, Romulus, Remus, Mars\}$, each individual o is interpreted by itself (i.e., $o^{\mathcal{I}} = o$), and for $hates^{\mathcal{I}} = \{(Cain, Abel), (Romulus, Remus)\}$ and $father^{\mathcal{I}} = parent^{\mathcal{I}} = \{(Cain, Adam), (Abel, Adam), (Remus, Mars), (Romulus, Mars)\}$ is a model of L. Moreover, $L \models \neg parent \sqsubseteq \neg father$ but $L \not\models \exists father^{-}.\{Adam\}$.

2.1 (Unions of) Conjunctive Queries

Definition 2 A *conjunctive query* (CQ) $q(\mathbf{X})$ is an expression

$$\{\mathbf{X} \mid Q_1(\mathbf{X_1}), \ldots, Q_n(\mathbf{X_n})\}, \tag{1}$$

[2] We remark that L is in $\mathcal{SHOI}$, which is another subclass of $\mathcal{SHOIN}(\mathbf{D})$, for which answering conjunctive queries as in the example is decidable; see, e.g., [32].

where each Q_i is a concept or role expression and each $\mathbf{X_i}$ is a singleton or pair of variables and individuals, and where $\mathbf{X} \subseteq \bigcup_{i=1}^{n} \mathrm{vars}(\mathbf{X_i})$ are its *distinguished* (or *output*) variables. A *union of conjunctive queries* (UCQ) $q(\mathbf{X})$ is a disjunction

$$q_1(\mathbf{X}) \vee \cdots \vee q_m(\mathbf{X}) \tag{2}$$

of CQs $q_i(\mathbf{X})$, $1 \leq i \leq m$, whose *distinguished* variables are $\mathbf{X}$.

We will omit $\mathbf{X}$ if it is clear from the context. Intuitively, a CQ $q(\mathbf{X})$ is a conjunction $Q_1(\mathbf{X_1}) \wedge \cdots \wedge Q_n(\mathbf{X_n})$ of concept and role expressions, which is true for a ground substitution σ of the variables in $\mathbf{X}$ by individuals and data values $\mathbf{X}\sigma$ (a so called answer), if in each model of the DL-KB the conjunction is satisfiable. A UCQ $q(\mathbf{X})$ is true for σ, whenever some $q_i(\mathbf{X})$ is true for σ. More formally, the semantics of CQs and UCQs is as follows.

Definition 3 For any CQ $q(\mathbf{X}) = \{\ \mathbf{X} \mid Q_1(\mathbf{X_1}), \ldots, Q_n(\mathbf{X_n})\}$, let

$$\phi_q(\mathbf{X}) = \exists \mathbf{Y} \bigwedge_{i=1}^{n} Q_i(\mathbf{X_i}), \tag{3}$$

where $\mathbf{Y}$ are the variables of $\mathbf{X_1} \cup \cdots \cup \mathbf{X_n}$ not in $\mathbf{X}$, and for any UCQ $q(\mathbf{X}) = \bigvee_{i=1}^{m} q_i(\mathbf{X})$, let

$$\phi_q(\mathbf{X}) = \bigvee_{i=1}^{m} \phi_{q_i}(\mathbf{X}). \tag{4}$$

Then, for any (U)CQ $q(\mathbf{X})$, the set of *answers of* $q(\mathbf{X})$ *on* L is the set of tuples

$$ans(q(\mathbf{X}), L) = \{\mathbf{c} \in (\mathbf{I} \cup \Delta^{\mathbf{D}})^{|\mathbf{X}|} \mid L \models \phi_q(\mathbf{c})\}. \tag{5}$$

Here $\phi_q(\mathbf{c})$ is short for $\phi_q(\mathbf{X}\sigma)$ such that $\mathbf{X}\sigma = \mathbf{c}$ for some ground substitution σ. Existential quantifiers in $\phi_q(\mathbf{c})$ are evaluated as usual in first-order logic.[3]

Example 5 Regarding Example 2, $cq_1(X,Y) = \{X, Y \mid parent(X,Z),\ parent(Y,Z),\ hates(X,Y)\}$ and $cq_2(X,Y) = \{X, Y \mid father(X,Y),\ father(Y,Z)\}$ are CQs with output X, Y, and $ucq(X,Y) = cq_1(X,Y) \vee cq_2(X,Y)$ is a UCQ. Since $L \models \phi_{cq_1}(Cain, Abel)$, the tuple $(Cain, Abel)$ is an answer of $cq_1(X,Y)$ on L. In fact, $ans(cq_1(X,Y), L) = \{(Cain, Abel), (Romulus, Remus)\}$, while $ans(cq_2(X,Y), L) = \emptyset$, as in L we know nothing about grandfathers. Furthermore, $ans(ucq(X,Y), L) = ans(cq_1(X,Y), L)$.

3 CQ Programs

After having recalled Description Logics, we now define rules on top of DL knowledge bases. To this end, we introduce cq-programs, which generalize nonmonotonic dl-programs [10,11] with disjunction in the head and allow for conjunctive and unions of conjunctive queries over DL knowledge bases. The former extension (disjunctive heads) had also been cursory introduced in [8], but was not further analyzed there. The latter extension is completely novel.

As in [8,10], we assume besides a vocabulary Ψ of a DL-KB, a function-free first-order vocabulary Φ of nonempty finite sets $\mathcal{C}$ and $\mathcal{P}$ of constant resp. predicate symbols,

[3] Here, typing of variables wrt. $\mathbf{I}$ and $\Delta^{\mathbf{D}}$ is implicit: syntactically malformed ground atoms evaluate to false.

and a set $\mathcal{X}$ of variables. It is assumed that $\mathcal{C} \subseteq \mathbf{I} \cup \Delta^{\mathbf{D}}$ holds, which serves to ensure that all objects in the rules (represented by constants) are known as individuals in the DL knowledge base; typically, $\mathcal{C} = \mathbf{I} \cup \Delta^{\mathbf{D}}$ holds.[4]

A *term* is either a constant from $\mathcal{C}$ or a variable from $\mathcal{X}$. Given $p \in \mathcal{P}$, an *atom* is defined as $p(t_1, \ldots, t_k)$, where k is called the arity of p and each $t_1, \ldots, t_k$ are terms. A *classical literal* (or simply *literal*) l is an atom a or a (classically) negated atom $\neg a$. A *weakly negated literal* (or *negation as failure* (*NAF*) *literal*) is a default-negated literal *not* l.

3.1 Syntax

Informally, a cq-program consists of a DL-KB L and a generalized disjunctive program P, which may involve queries to L. Roughly, such a query may ask whether a specific description logic axiom, a conjunction or a union of conjunctions of DL axioms is entailed by L or not. Note that predicate symbols in P are different from those in L.

Definition 4 A dl-atom α is in form $\mathrm{DL}[\lambda; q](\mathbf{X})$, where $\lambda = S_1\, op_1\, p_1, \ldots, S_m\, op_m\, p_m$ ($m \geq 0$) is a list of expressions $S_i\ op_i\ p_i$ called *input list*, each S_i is either a concept or a role, $op_i \in \{\uplus, \cup, \cap\}$, p_i is a predicate symbol matching the arity of S_i, and q is either

- a (U)CQ with output variables $\mathbf{X}$ (in this case, α is called a *(u)cq-atom*), or
- $q(\mathbf{X})$ is a dl-query as in [10] (in this case, α is called an *ordinary dl-atom*), i.e.,
  1. a concept inclusion axiom F or its negation $\neg F$ with $\mathbf{X}$ void, or
  2. of form $C(\mathbf{X})$ or $\neg C(\mathbf{X})$, where C is a concept, $\mathbf{X} = t$, and t is a term, or
  3. of form $R(\mathbf{X})$ or $\neg R(\mathbf{X})$, where R is a role, $\mathbf{X} = (t_1, t_2)$, and t_1 and t_2 are terms.

Each p_i is an *input predicate symbol*; intuitively, $op_i = \uplus$ increases S_i by the extension of p_i, while $op_i = \cup$ increases $\neg S_i$; $op_i = \cap$ constrains S_i to p_i.

Please note that a dl-atom α cannot be classically negated.

Example 6 In Ex. 1, $\mathrm{DL}[parent](X, Z)$, $\mathrm{DL}[parent](Y, Z)$, and $\mathrm{DL}[hates](X, Y)$ are ordinary dl-atoms, while in Ex. 2, $\mathrm{DL}[parent(X, Z), parent(Y, Z), hates(X, Y)](X, Y)$ is a cq-atom with output X, Y. The cq-atom $\mathrm{DL}[parent \uplus p; parent(X, Y), parent(Y, Z)](X, Z)$ with output X, Z extends L by adding the extension of p to the role *parent*, and then joins *parent* with itself.

Definition 5 A *cq-rule* r is of the form

$$a_1 \vee \cdots \vee a_k \leftarrow b_1, \ldots, b_m, not\ b_{m+1}, \ldots, not\ b_n, \tag{6}$$

where every a_i is a literal and every b_j is either a literal or a dl-atom. We define $H(r) = \{a_1, \ldots, a_k\}$ and $B(r) = B^+(r) \cup B^-(r)$, where $B^+(r) = \{b_1, \ldots, b_m\}$ and $B^-(r) = \{b_{m+1}, \ldots, b_n\}$. If $B(r) = \emptyset$ and $H(r) \neq \emptyset$, then r is a *fact*. If $H(r) = \emptyset$ and $B(r) \neq \emptyset$, then r is a *constraint*, and if $|H(r)| \leq 1$ then r is *non-disjunctive*.

A *cq-program* $KB = (L, P)$ consists of a DL-KB L and a finite set of cq-rules P. It is *non-disjunctive*, if each $r \in P$ is non-disjunctive, and *positive*, if $B^-(r) = \emptyset$ for all $r \in P$ and $\cap$ does not occur in P.

[4] This assumption may be varied, by allowing the rules to see only a subset of the individuals in the DL-KB, and/or allowing constants in the rules not occurring in the DL-KB; this can be modeled with the current convention. We omit further discussion here, and similar for access to equality in the DL-KB, which can be modeled like role access; see [7].

$$
\begin{aligned}
&\mathit{WhiteWine} \sqcup \mathit{RedWine} \sqsubseteq \mathit{Wine};\ \ \mathit{WhiteWine} \sqsubseteq \neg \mathit{RedWine};\\
&\mathrm{Trans}(\mathit{locatedIn});\ \geq 1.\mathit{locatedIn} \sqsubseteq \mathit{Wine} \sqcup \mathit{Region};\ \ \top \sqsubseteq \forall \mathit{locatedIn}.\mathit{Region};\\
&\mathit{locatedIn}(\mathit{southAustraliaRegion}, \mathit{australianRegion});\\
&\mathit{locatedIn}(\mathit{mountadamRiesling}, \mathit{southAustraliaRegion});\\
&\mathit{locatedIn}(\mathit{mountadamChardonnay}, \mathit{southAustraliaRegion});\\
&\mathit{locatedIn}(\mathit{mountadamPinotNoir}, \mathit{southAustraliaRegion});\\
&\mathit{WhiteWine}(\mathit{mountadamChardonnay});\\
&\mathit{RedWine}(\mathit{mountadamPinotNoir});\\
&\mathit{hasFlavor}(\mathit{mountadamRiesling}, \mathit{delicate});\\
&\mathit{locatedIn}(\mathit{stonleighSauvignonBlanc}, \mathit{newZealandRegion});\\
&\mathit{locatedIn}(\mathit{mongridgeMerlot}, \mathit{newZealandRegion});\\
&\mathit{locatedIn}(\mathit{selaksIceWine}, \mathit{newZealandRegion});\\
&\mathit{RedWine}(\mathit{longridgeMerlot});\\
&\mathit{WhiteWine}(\mathit{selaksIceWine});\\
&\mathit{hasFlavor}(\mathit{stonleighSauvignonBlanc}, \mathit{delicate})
\end{aligned}
$$

Fig. 1 Simplistic wine ontology

$$
\begin{aligned}
\mathit{visit}(L) \vee \neg \mathit{visit}(L) \leftarrow\ & \mathrm{DL}[\mathit{WhiteWine}](W), \mathrm{DL}[\mathit{RedWine}](R), && (r_1)\\
& \mathrm{DL}[\mathit{locatedIn}](W, L), \mathrm{DL}[\mathit{locatedIn}](R, L),\\
& \mathit{not}\ \mathrm{DL}\big[\mathit{locatedIn}(L, L')\big](L).\\
\leftarrow\ & \mathit{visit}(X), \mathit{visit}(Y), X \neq Y. && (r_2)\\
\mathit{some\_visit} \leftarrow\ & \mathit{visit}(X). && (r_3)\\
\leftarrow\ & \mathit{not}\ \mathit{some\_visit}. && (r_4)\\
\mathit{delicate\_region}(W) \leftarrow\ & \mathit{visit}(L), \mathit{delicate}(W), \mathrm{DL}[\mathit{locatedIn}](W, L). && (r_5)\\
\mathit{delicate}(W) \leftarrow\ & \mathrm{DL}[\mathit{hasFlavor}](W, \mathit{delicate}). && (r_6)
\end{aligned}
$$

Fig. 2 Delicate wine region program

Example 7 In Examples 1 and 2, P and P' contain single rules which are positive, and thus (L, P) and (L, P') are both non-disjunctive positive cq-programs.

Example 8 Let $KB = (L, P)$, where L is shown in Figure 1, a simplistic variant of the well-known Wine ontology [39], and P is the program shown in Figure 2. Informally, in L, the concepts *WhiteWine* and *RedWine* are disjoint, and *locatedIn* is a transitive role, whose domain the union of *Wine* and *Region*, and whose codomain is *Region*. The next statements are assertions about various wines and areas of cultivation. The rule r_1 in P selects a maximal region in which both red and white wine grow, and the next three rules make sure that exactly one such region is picked, by enforcing that no more than two regions are chosen (r_2) and that at least one is chosen (rules r_3 and r_4). The last two rules r_5 and r_6 single out all the sub-regions of the selected region producing some delicate wine, i.e., if a wine has a delicate flavor which is specified by individual *delicate*.

Note that the program P exclusively uses instance retrieval queries—with one exception in the first rule: the weakly negated dl-atom is a conjunctive query with only one query atom, since we have to remove the non-distinguished variable L' from the output to keep the rule safe. The program will be used throughout the paper for demonstrating our rewriting methods.

3.2 Semantics

Let $KB = (L, P)$ be a cq-program. The *Herbrand base* of P, denoted HB_P, is the set of all ground literals with a standard predicate symbol that occurs in P and constant symbols in $\mathcal{C}$. An *interpretation* I relative to P is a consistent subset of HB_P. The grounding of P, denoted $ground(P)$, is the set of all ground instances of rules in P (with respect to $\mathcal{C}$); here, in ordinary dl-atoms the output variables are replaced by constants and in (u)cq-atoms the distinguished variables in $q(\mathbf{X})$ are replaced by constants, and the output list $\mathbf{X}$ is replaced by the empty list; e.g., $\mathrm{DL}[parent](X, Y)$ is instantiated, for the substitution $X \mapsto Cain$, $Y \mapsto Adam$ to $\mathrm{DL}[parent](Cain, Adam)$ and $\mathrm{DL}[parent(X, Z), parent(Y, Z)](X, Y)$ is instantiated for the same substitution to $\mathrm{DL}[parent(Cain, Z), parent(Adam, Z)]()$.

We first define satisfaction of atoms with respect to an interpretation.

Definition 6 Let I be an interpretation of P. Then

- an ordinary ground atom $l \in HB_P$ is *satisfied* by I, or I is a *model* of l under L, denoted $I \models_L l$, iff $l \in I$;
- a ground ordinary dl-atom $a = \mathrm{DL}[\lambda; Q](\mathbf{c})$ is satisfied by I under L, denoted $I \models_L a$, if $L \cup \lambda(I) \models Q(\mathbf{c})$, where $\lambda(I) = \bigcup_{i=1}^{m} A_i(I)$ and
  - $A_i(I) = \{S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}$, for $op_i = \uplus$;
  - $A_i(I) = \{\neg S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}$, for $op_i = \uplus$;
  - $A_i(I) = \{\neg S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I \text{ does not hold}\}$, for $op_i = \cap$;[5]
- a ground instance $a(\mathbf{c})$ of a (u)cq-atom $a(\mathbf{X}) = \mathrm{DL}[\lambda; q](\mathbf{X})$, is satisfied by I under L, denoted $I \models_L a(\mathbf{c})$, if $\mathbf{c} \in ans(q(\mathbf{X}), L \cup \lambda(I))$.

We next define satisfaction of rules and models of a cq-programs.

Definition 7 Let r be a ground cq-rule. We define (i) $I \models_L H(r)$ iff there is some $a \in H(r)$ such that $I \models_L a$, (ii) $I \models_L B(r)$ iff $I \models_L a$ for all $a \in B^+(r)$ and $I \not\models_L a$ for all $a \in B^-(r)$, and (iii) $I \models_L r$ iff $I \models_L H(r)$ whenever $I \models_L B(r)$. We say that I is a *model* of a cq-program $KB = (L, P)$, or I *satisfies* KB, denoted $I \models KB$, iff $I \models_L r$ for all $r \in ground(P)$. A cq-program KB is *satisfiable*, if it has some model, and is *unsatisfiable* otherwise.

The *Gelfond-Lifschitz transform* of program P without dl-atoms relative to an interpretation $I \subseteq HB_P$, denoted P^I, is the positive program obtained from $ground(P)$ by (i) deleting every rule r with $B^-(r) \cap I \neq \emptyset$, and (ii) deleting the negative body

[5] We note that negative role assertions $S_i(\mathbf{e})$ in $\lambda(I)$, which are syntactically not allowed in $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$, can be emulated by using that $L' \cup \{\neg R(a, b)\}$ is unsatisfiable iff $L' \cup \{A(a), B(b), \exists R.B \sqsubseteq \neg A\}$ is unsatisfiable (where A and B are two fresh atomic concepts and L' is any DL-KB) [20]. Negated datatype role membership axioms can be removed in a similar way. In OWL 2 (formerly OWL 1.1), negative property membership assertions are allowed [40].

from every remaining rule. I is an *answer set* of P if it is a minimal model (w.r.t. set inclusion) of P^I.

We remark that any ordinary dl-atom $\mathrm{DL}[\lambda; q](\mathbf{X})$, where q is a role or a concept, can be easily cast to an equivalent cq-atom $\mathrm{DL}[\lambda; q'](\mathbf{X'})$, where q' is a CQ; for example, $\mathrm{DL}[parent](X, Z)$ can be cast to $\mathrm{DL}[parent(X, Z)](X, Z)$, and $\mathrm{DL}[hates](Cain, Z)$ to $\mathrm{DL}[hates(Cain, Z)](Z)$.

3.3 Minimal-model semantics for positive cq-programs

We first consider positive cq-programs. Like for ordinary positive programs, every non-disjunctive positive cq-program that is satisfiable has a single minimal model, which naturally characterizes its semantics. The proof of the next lemma is analogous to the proof for normal positive dl-programs [7].

Lemma 1 *Let $KB = (L, P)$ be a non-disjunctive positive cq-program. If the interpretations $I_1, I_2 \subseteq HB_P$ are models of KB, then $I_1 \cap I_2$ is also a model of KB.*

Proof Suppose that $I_1, I_2 \subseteq HB_P$ are models of KB, that is, $I_i \models_L r$ for every $r \in ground(P)$ and $i \in \{1, 2\}$. We show that $I = I_1 \cap I_2$ is also a model of KB, that is, $I \models_L r$ for every $r \in ground(P)$. Consider any $r \in ground(P)$, and assume that $I \models_L l$ for all $l \in B^+(r) = B(r)$. That is, $I \models_L l$ for all classical literals $l \in B(r)$ and $I \models_L a$ for all cq-atoms $a \in B(r)$. Hence, $I_i \models_L l$ for all classical literals $l \in B(r)$, for every $i \in \{1, 2\}$. Furthermore, since every cq-atom in $ground(P)$ is monotonic relative to KB, it holds that $I_i \models_L a$ for all dl-atoms $a \in B(r)$, for every $i \in \{1, 2\}$. Since I_1 and I_2 are models of KB, it follows that $I_i \models_L H(r)$, for every $i \in \{1, 2\}$, and thus $I \models_L H(r)$. This shows that $I \models_L r$. Hence, I is a model of KB. □

As an immediate corollary of this result, every satisfiable non-disjunctive positive cq-program KB has a unique minimal model, which is contained in every model of KB.

Corollary 1 *Let $KB = (L, P)$ be a non-disjunctive positive cq-program. If KB is satisfiable, then there is a unique model $I \subseteq HB_P$ of KB s.t. $I \subseteq J$ for all models $J \subseteq HB_P$ of KB.*

Example 9 The cq-program (L, P) in Ex. 1 has the single minimal model $\{BadChild(Cain)\}$, while (L, P') in Ex. 2 has the single minimal model $\{BadChild(Cain), BadChild(Romulus)\}$.

On the other hand, if a cq-program contains disjunction, then multiple minimal models of KB may exist.

Example 10 Consider the program in Example 8. If we remove "*not*" from P by replacing rule r_1 with

$$visit(L) \vee \neg visit(L) \leftarrow \mathrm{DL}[WhiteWine](W), \mathrm{DL}[RedWine](R), \\ \mathrm{DL}[locatedIn](W, L), \mathrm{DL}[locatedIn](R, L),$$

i.e., we abandon the maximality condition on the region L, then we get a positive cq-program which has three minimal models. The following minimal models are abridged versions of these models:

1. $\{delicate\_region(mountadamRiesling), visit(australianRegion), \dots\}$,
2. $\{delicate\_region(stonleighSauvignonBlanc), visit(newZealandRegion), \dots\}$,
3. $\{delicate\_region(mountadamRiesling), visit(southAustraliaRegion), \dots\}$,

3.4 Strong answer-set semantics for cq-programs

We now define the *strong answer-set semantics* of general cq-programs. It reduces to the minimal model semantics for positive cq-programs, using a generalized transformation that removes all NAF-literals and every *nonmonotonic* dl-atom. A dl-atom is said to be monotonic in the sense given by the following definition:

Definition 8 A ground dl-atom a is *monotonic* relative to $KB = (L, P)$ iff $I \models_L a$ implies $I' \models_L a$, for all $I \subseteq I' \subseteq HB_P$, otherwise a is *nonmonotonic*.

Possibly nonmonotonic dl-atoms are treated similarly as NAF-literals. This is particularly useful, if we do not know a priori whether some dl-atoms are monotonic, and determining this might be costly; notice, however, that absence of $\uplus$ in an input list of a dl-atom is a simple syntactic criterion that implies monotonicity of a dl-atom.

For any cq-program $KB = (L, P)$, we denote by DL_P the set of all ground dl-atoms that occur in $ground(P)$. We assume that KB has an associated set $DL_P^+ \subseteq DL_P$ of ground dl-atoms which are known to be monotonic, and we denote by $DL_P^? = DL_P \setminus DL_P^+$ the set of all other ground dl-atoms.[6] An input literal of $a \in DL_P$ is a ground literal with an input predicate of a and constant symbols in Φ.

Definition 9 The *strong dl-reduct* of P relative to L and an interpretation $I \subseteq HB_P$, denoted sP_L^I, is the set of all rules obtained from $ground(P)$ by

(i) deleting every cq-rule r such that either $I \not\models_L a$ for some $a \in B^+(r) \cap DL_P^?$, or $I \models_L l$ for some $l \in B^-(r)$; and
(ii) deleting from each remaining cq-rule r all literals in $B^-(r) \cup (B^+(r) \cap DL_P^?)$.

Notice that (L, sP_L^I) has only monotonic dl-atoms and no NAF-literals anymore. Thus, (L, sP_L^I) is a positive cq-program, and by Corollary 1, has a minimal model, if it is satisfiable and non-disjunctive. We thus define the strong answer-set semantics of general cq-programs by reduction to the minimal model semantics of positive cq-programs as follows.

Definition 10 Let $KB = (L, P)$ be a cq-program. A *strong answer set* of KB is an interpretation $I \subseteq HB_P$ such that I is a minimal model of (L, sP_L^I).

Example 11 The minimal models shown in Example 9 are strong answer sets of the resp. cq-programs.

The program KB from Example 8 has the following two answer sets (only the positive facts of the predicates *delicate_region* and *visit* are listed):

1. $\{delicate\_region(mountadamRiesling)$, $visit(australianRegion)$, $\dots\}$, and
2. $\{delicate\_region(stonleighSauvignonBlanc)$, $visit(newZealandRegion)$, $\dots\}$.

Compared to the program in Example 10, the third answer set of the latter is pruned, according to which *southAustraliaRegion* is selected for a visit, since that region is not a maximal feasible region (it is located in *australianRegion* of the first answer set).

[6] The set DL_P^+ is a parameter which allows one to freely reflect an epistemic state about the monotonic behavior of dl-atoms with respect to the underlying DL knowledge base L, which is crucial for the definition of strong answer sets. Note that determining whether a given ground dl-atom is monotonic with respect to L is computationally expensive in general; by modeling the monotonicity as unknown, we can trade the respective effort for a semantic weakening, as more answer sets will be possible. We remark that technically, for $DL_P^+ = \emptyset$ we obtain the concept of weak answer set from [10, 7], generalized to cq-programs.

The following result shows that the strong answer-set semantics of a cq-program $KB = (L, P)$ without dl-atoms coincides with the ordinary answer set semantics of P.

Theorem 1 *Let $KB = (L, P)$ be a cq-program without dl-atoms. Then, $I \subseteq HB_P$ is a strong answer set of KB iff it is an answer set of the ordinary program P.*

Proof Let $I \subseteq HB_P$. Then, P has no dl-atoms implies $sP_L^I = P^I$. Hence, I is a minimal model of (L, sP_L^I) iff I is a minimal model of P^I. Therefore, I is a strong answer set of (L, P) iff I is an answer set of P. □

The next result shows that, as desired, strong answer sets of a cq-program KB are models of KB, too, and moreover minimal models of KB if all dl-atoms are monotonic (and known as such, i.e., $DL_P^? = \emptyset$).

Theorem 2 *Let $KB = (L, P)$ be a cq-program, and let I be a strong answer set of KB. Then, (a) I is a model of KB, and (b) I is a minimal model of KB if $DL_P = DL_P^+$.*

Proof (a) Let I be a strong answer set of KB. To show that I is also a model of KB, we have to show that $I \models_L r$ for all $r \in \mathit{ground}(P)$. Consider any $r \in \mathit{ground}(P)$. Suppose that $I \models_L l$ for all $l \in B^+(r)$ and $I \not\models_L l$ for all $l \in B^-(r)$. Then, the cq-rule r' that is obtained from r by removing all the literals in $B^-(r) \cup (B^+(r) \cap DL_P^?)$ is contained in sP_L^I. Since I is a minimal model of (L, sP_L^I) and thus in particular a model of (L, sP_L^I), it follows that I is a model of r'. Since $I \models_L l$ for all $l \in B^+(r')$ and $I \not\models_L l$ for all $l \in B^-(r') = \emptyset$, it follows that $I \models_L H(r) = H(r')$. This shows that $I \models_L r$. Hence, I is a model of KB.

(b) By part (a), every strong answer set I of KB is a model of KB. Assume that every dl-atom of KB is monotonic, that is, $DL_P = DL_P^+$. We show now that I is a minimal model of KB. Towards a contradiction, suppose the contrary, that is, there is a $J \subset I$ such that J is a model of KB. Since J is a model of KB, we obtain that J is a model of (L, sP_L^J). Since every dl-atom $a \in DL_P$ is monotonic relative to KB, it follows that $sP_L^I \subseteq sP_L^J$. Hence, J is also a model of (L, sP_L^I). But this contradicts that I is a minimal model of (L, sP_L^I). Therefore, I is a minimal model of KB. □

These and many other of the semantic properties of dl-programs are naturally inherited to cq-programs, like the existence of a unique answer set for non-disjunctive positive programs (if any answer set exists), or for non-disjunctive programs if *not* is used in a stratified way.

Furthermore, the strong answer set semantics for cq-programs without $\cap$ can be equivalently defined, like for dl-programs without $\cap$, in terms of answer sets of HEX-programs (see [9,19]). The latter semantics is based on a characterization of answer sets of ordinary disjunctive logic programs that uses an alternative reduct [12], and, informally, states that M is an answer set if M is a minimal model of the rules in the grounding of the program whose body is satisfied by M. It can be used to emulate various extensions of normal logic programs apart from dl-programs, including programs with monotone cardinality atoms [22]. By means of this correspondence, one can easily implement cq-programs without $\cap$ on top of dlvhex, which is a prototype implementation of HEX programs; we provide additional details on an implementation of cq-programs in Section 7.

The examples in the introduction show that cq-programs are more expressive than dl-programs in [8,10]. This can be made also formally more precise by comparing the computational complexity of cq-programs and ordinary dl-programs, which shows that the former can express more difficult problems. This will be done in the next section.

Table 1 Complexity of deciding strong answer set existence for different cq-programs (completeness results); for positive such programs, it is listed in parentheses if different.

program $KB = (L, P)$	L in $\mathcal{SHIF}(\mathbf{D})$	L in $\mathcal{SHOIN}(\mathbf{D})$
non-disjunctive dl-program	NEXP (EXP)	$\mathrm{P}^{\mathrm{NEXP}}$(NEXP)
disjunctive dl-program	$\mathrm{NEXP}^{\mathrm{NP}}$ (NEXP)	$\mathrm{NEXP}^{\mathrm{NP}}$ (NEXP)
non-disjunctive cq-program	2-EXP	?
disjunctive cq-program	2-EXP	?

4 Computational Complexity

In this section, we address the computational complexity of cq-programs, and complement some results in [10] with results for disjunctive dl-programs. However, we refrain from giving an extensive complexity study here as in [10], and confine to consider the problem of deciding the existence of a strong answer set for a given (finite) cq-program $KB = (L, P)$. Clearly, this problem is decidable if answering (union of) conjunctive queries over L, augmented with positive and negative assertions, is decidable. This is the case for many description logics including $\mathcal{SHIF}(\mathbf{D})$, while it is currently unknown whether this is feasible for $\mathcal{SHOIN}(\mathbf{D})$, as the decidability of answering conjunctive queries for this logic is open, cf. [13].

The complexity results are compactly summarized in Table 1, in which the results for non-disjunctive dl-programs are recalled from [10]. Recall that NEXP are the problems solvable in non-deterministic exponential time, and that A^B is the class of problems solvable in class A with the help of an oracle for the class B; for further references and background, cf. [4].

Furthermore, we recall that deciding whether an ordinary disjunctive logic program (without dl-atoms) has some answer set is $\mathrm{NEXP}^{\mathrm{NP}}$-complete, and is NEXP-complete if the program is disjunction-free, cf. [4]; the latter result is the correspondent of the seminal result that stable model semantics of normal logic programs is NP-complete in the propositional case [24]. For $\mathcal{SHIF}(\mathbf{D})$, answering UCQs is 2-EXP-complete, i.e., complete for double exponential time, as follows from the results of [3] and [21];[7] in fact, Lutz has shown that answering CQs is already 2-EXP-hard for the description logic $\mathcal{ALC}$ [21], which is a core of expressive description logics.

The results show that allowing cq-queries significantly increases the expressiveness of programs compared to allowing only ordinary dl-queries (assuming the widely accepted hypothesis that $\mathrm{NEXP}^{\mathrm{NP}}$ is strictly included in 2-EXP). In fact, for $\mathcal{SHOIN}(\mathbf{D})$ it is currently not known whether cq-programs are decidable. Interestingly, for disjunctive dl-programs, the dl-atoms do not add complexity compared to the ordinary case, while for cq-programs the rules do not add complexity, i.e., KB has the complexity of answering (U)CQs over a DL-KB. This also means that we can transform, in polynomial time, disjunctive dl-programs to ordinary disjunctive logic programs (thus eliminating completely the ontology part), and disjunctive cq-programs to answering CQs (thus eliminating all rules); whether this will be of use in practice remains to be explored in future investigation.

[7] We always assume proper datatypes, such that they do not increase the complexity of query answering in the underlying DL $\mathcal{SHIF}$ resp. $\mathcal{SHOIN}$.

In the case of positive programs, i.e., in absence of $\cap$ and *not*, deciding strong answer set existence for disjunctive dl-programs has the same complexity as for non-disjunctive programs with a DL-KB from $\mathcal{SHOIN}(\mathbf{D})$, which is lower than for arbitrary dl-programs (assuming that NEXP is properly included in NEXP^{NP}). The reason is that the technique for the latter in [10] immediately extends to disjunctive programs. The NEXP lower bound is inherited from the NEXP-hardness of deciding the consistency of positive ordinary disjunctive logic programs with constraints, which can be easily shown adapting proofs e.g. in [4]. For cq-programs, on the other hand, the restriction to positive programs does not lower the complexity.

We now establish the results more formally. The following lemma is useful.

Lemma 2 *Let $KB = (L, P)$ be a cq-program, let I be an interpretation for KB, and let $a = \text{DL}[\lambda; Q](\mathbf{c})$ be a ground dl-atom. Then, deciding whether $I \models_L a$ is feasible (i) in* co-NEXP*, if L is from $\mathcal{SHOIN}(\mathbf{D})$ and a is not a (u)cq-atom, (ii) in* EXP*, if L is from $\mathcal{SHIF}(\mathbf{D})$ and a not a (u)cq-atom, and (iii) in* 2-EXP *if L is from $\mathcal{SHIF}(\mathbf{D})$.*

Proof Given I, we need to compute $\lambda(I)$, which can be done in polynomial time, and then test $L \cup \lambda(I) \models Q(\mathbf{c})$. Obviously, adding all unnegated assertions and negative concept assertions to L is straightforward, and the negative role assertions and datatype role memberships can be emulated, using e.g. the technique in Footnote 5, in polynomial time in a DL-KB L' such that $L \cup \lambda(I) \models Q(\mathbf{c})$ iff $L' \models Q(\mathbf{c})$, where L' is in $\mathcal{SHIF}(\mathbf{D})$ (resp., $\mathcal{SHOIN}(\mathbf{D})$) if L is in $\mathcal{SHIF}(\mathbf{D})$ (resp., $\mathcal{SHOIN}(\mathbf{D})$). Thus (iii) follows by the results of [3]. Items (i) and (ii) have been established implicitly in [10]. □

Theorem 3 *Given a vocabulary Φ and a cq-program $KB = (L, P)$, the problem of deciding whether KB has a strong answer set has the complexity as stated in Table 1 for each combination of program class P and DL knowledge base class L.*

Proof The results for non-disjunctive dl-programs were shown in [10,7], and we refer the reader to these papers for the (very detailed) proofs. Let us consider next disjunctive dl-programs. The NEXP^{NP} upper bound for the case where L is from $\mathcal{SHOIN}(\mathbf{D})$ is easily derived from Lemma 2 as follows. There are polynomially many ground dl-atoms that occur in the grounding $ground(P)$ of P. Recall that for such a ground dl-atom of form $a = \text{DL}[\lambda; Q](\mathbf{c})$, where $\lambda = S_1\ op_1\ p_1, \ldots, S_m\ op_m\ p_m$, and an interpretation I for KB, $\lambda(I) = \bigcup_{i=1}^{m} A_i(I)$ denotes the value of the input list λ with respect to I. Now if there are n constant symbols in Φ, then each value $\lambda(I)$ has size bounded by mn^2, and the number of distinct values $\lambda(I_1), \ldots, \lambda(I_r)$, over all interpretations I for KB, is bounded by $r \leq (2^{n^2})^m = 2^{mn^2}$, which is single exponential in m and n. Thus, each $\lambda(I)$ has size polynomial in the size of KB, and the number r of distinct $\lambda(I)$ is single exponential in the size of KB. As a consequence, all distinct updates $L \cup \lambda(I_1)$, $\ldots$, $L \cup \lambda(I_r)$ can be computed in single exponential time, and with the help of an NP oracle, all possible queries answers $L \cup \lambda(I_j) \models Q(\mathbf{c})$, for all $j = 1, \ldots, r$ and tuples $\mathbf{c}$, can be computed in single exponential time; note that a NP oracle is sufficient here and that we do not need a NEXP oracle: if we pad L with exponentially many dummy tautologies to L' (which is feasible in exponential time), then $L' \cup \lambda(I_j) \models Q(\mathbf{c})$ iff $L \cup \lambda(I_j) \models Q(\mathbf{c})$, and by its particular form deciding $L' \cup \lambda(I_j) \models Q(\mathbf{c})$ is in co-NP.

We can therefore decide the existence of some strong answer set M of (L, P) in two steps as follows. In a first step, we compute a table of all query results $L \cup \lambda(I_j) \models Q(\mathbf{c})$. Once we have this database, evaluating ground dl-atoms a with respect to a particular interpretation I is cheap (compute $\lambda(I)$, determine the I_j such that $\lambda(I) = \lambda(I_j)$,

and do a table lookup). In the second step, we proceed similarly as for an ordinary disjunctive logic program, but with an adapted algorithm: guess M, compute sP_L^M, and check whether M satisfies sP_L^M and, using an NP oracle, whether there is no $M' \subset M$ that satisfies sP_L^M. Overall, this is feasible in non-deterministic exponential time using an NP oracle, which shows that the problem is in NEXP^{NP}. The NEXP^{NP}-hardness is inherited from ordinary disjunctive logic programs [4].

In presence of (u)cq-atoms, 2-EXP is clearly a lower bound for the complexity of the problem, even for simple positive non-disjunctive cq-programs, since by the result of [21] evaluating a single ground dl-atom is 2-EXP-hard. On the other hand, 2-EXP is also an upper bound: as for dl-programs, we can first construct a table of all relevant (possibly exponentially many) ground cq-atoms a, with all distinct values $\lambda(I_1), \ldots, \lambda(I_r)$ for the input list λ of each a (again, r is at most single exponential) and query results $L \cup \lambda(I_j) \models Q(\mathbf{c})$, for all $j = 1, \ldots, r$ and tuples $\mathbf{c}$ can be constructed, such that deciding strong answer set existence for (L, P) can be done in a second step similarly as for ordinary disjunctive logic programs. Computing the table in the first step is feasible in 2-EXP, and the second step is then feasible in NEXP^{NP}, thus in 2-EXP. Overall, this yields a 2-EXP upper bound.

Finally, it remains to prove the entries for positive disjunctive dl-programs. Here, the same characterization for the existence of a strong answer set as in the non-disjunctive case from [10] can be exploited, whose proof is analogous: $KB = (L, P)$ has a strong answer set iff there exists an interpretation I and a subset $S \subseteq \{a \in DL_P \mid I \not\models_L a\}$ such that the ordinary positive program $P_{I,S}$, which is obtained from $\mathit{ground}(P)$ by deleting each rule that contains a dl-atom $a \in S$ and all remaining dl-atoms, has a model J such that $J \subseteq I$. Such an I and S can be guessed and verified in exponential time, which proves membership in NEXP. □

5 Rewriting Rules

In this section, we turn to equivalence preserving rewritings of (u)cq-atoms, which can be exploited for program optimization.

As shown in Example 3, in cq-programs we might have different possibilities for defining the same query. Indeed, the rules r and r' there are equivalent over any knowledge base L. However, the evaluation of r' might be implemented by performing the join between *parent* and *hates* on the DL side in a single call to a DL-reasoner, while r can be evaluated performing the join on the logic program side, over the results of two calls to the DL-reasoner. In general, making more calls is more costly, and thus r' may be preferable from an implementation point of view. Moreover, the size of the result transferred by the single call in this rule r' is smaller than the results of the two calls.

Towards exploiting such rewriting, we present some transformation rules for replacing a rule or a set of rules in a cq-program with another rule or set of rules, while preserving the semantics of the program (see Table 2). By means of (repeated) rule application, we can transform the program into another, equivalent program, which we consider in the next section. Indeed, a component for rewriting programs is conceivable, which rewrites a given cq-program (L, P) into a refined, equivalent cq-program (L, P'), which can be evaluated more efficiently. Recall that as for rule application, any ordinary dl-atom $\text{DL}[\lambda; Q](\mathbf{t})$, where $\mathbf{t}$ is a non-empty list of terms, is equivalent to the cq-atom $\text{DL}[\lambda; Q(\mathbf{t})](\mathbf{X})$, where $\mathbf{X} = \text{vars}(\mathbf{t})$. Throughout this and the next section, we disregard

for simplicity explicit consideration of datatypes; the results should be adjusted, without major problems, to accommodate them.

In the rewriting rules, the input lists λ_1 and λ_2 are assumed to be semantically equivalent (denoted $\lambda_1 \doteq \lambda_2$), that is, $\lambda_1(I) = \lambda_2(I)$, for every Herbrand interpretation I. This means that λ_1 and λ_2 modify the same concepts and roles with the same predicates in the same way; this can be easily recognized (in fact, in linear time). More liberal but more expensive notions of equivalence, taking L and/or P into account, might be considered.

Note that the rewriting rules (A), (B), (C), and (E) are applicable in an arbitrary program P, but (D) and (F) can only be applied in a specific context given by the preconditions on the rules in the program P.

Query Pushing (A) By this rule, cq-atoms $\mathrm{DL}[\lambda_1; cq_1](\mathbf{Y_1})$ and $\mathrm{DL}[\lambda_2; cq_2](\mathbf{Y_2})$ in the body of a rule (A1) can be merged to a rule (A2).

Example 12 The rule

$$a \leftarrow \mathrm{DL}[R_1(X,Y), R_2(Y,Z)](X), \mathrm{DL}[R_3(X,Y)](X,Y)$$

is equivalent to the rule

$$a \leftarrow \mathrm{DL}\big[R_1(X,Y'), R_2(Y',Z), R_3(X,Y)\big](X,Y).$$

Query Pushing can be similarly done when cq_1 and cq_2 are UCQs; here, we simply distribute the subqueries and form a single UCQ.

Variable Elimination I + II (B) Suppose an output variable X of a cq-atom in a rule r of form (B1a) or (B1b) occurs also in an atom $X = t$. Assume that t is different from X and that, in case of form (B1a) the underlying DL-KB is under Unique Name Assumption (UNA) whenever t is an output variable. Then, we can eliminate X from r as follows. Standardize the non-output variables of cq-atoms apart from the other variables in r, and replace uniformly X with t in cq, B, and H; let $cq_{X/t}$, $B_{X/t}$, and $H_{X/t}$ denote the respective results. Remove X from the output $\mathbf{Y}$ and, if t is a variable Z, add Z to them; the resulting rule r', in (B2) is then equivalent to the rule r_1 in (B1a) or to the rule r_2 in (B1b). By repeated application of this rule, we may eliminate multiple output variables of a cq-atom. Note that variables X in equalities $X = t$ not occurring in any output list can always be eliminated by simple replacement.

Example 13 The rules

$$r : a(X,Y) \leftarrow \mathrm{DL}[R(X,Z), C(Y), X = Y](X,Y), b(Y)$$

and

$$r' : a(Y,Y) \leftarrow \mathrm{DL}[R(Y,Z), C(Y)](Y), b(Y)$$

have the same outcome on every DL-KB L. Here, r' should be preferred due to the lower arity of its cq-atom. Similarly, the rule

$$a(X,Y) \leftarrow \mathrm{DL}[R(X,Z), C(Y), Y = c](X,Y), b(Y)$$

can be simplified to the rule

$$a(X,c) \leftarrow \mathrm{DL}[R(X,Z), C(c)](X), b(c).$$

Example 14 Assume that the Unique Name Assumption is not adopted in the Variable Elimination I rule. To show that we get wrong answers, take the cq-program (L,P), where $L = \{C(a), a = b\}$ and $P = \{p(X) \leftarrow \mathrm{DL}[C(X), X = a](X)\}$. Applying our Variable Elimination I rewriting, we get $P' = \{p(a) \leftarrow \mathrm{DL}[C(a)]()\}$. Now, in P we can infer both $p(a)$ and $p(b)$, whereas in P' only $p(a)$ holds.

Table 2 Equivalences ($H = a_1 \vee \cdots \vee a_k$; $B = b_1, \ldots, b_m, \mathit{not}\, b_{m+1}, \ldots, \mathit{not}\, b_n$)

QUERY PUSHING

$$r: \; H \leftarrow \mathrm{DL}[\lambda_1; cq_1](\mathbf{Y_1}), \mathrm{DL}[\lambda_2; cq_2](\mathbf{Y_2}), B. \tag{A1}$$

$$r': \; H \leftarrow \mathrm{DL}[\lambda_1; qp(cq_1, cq_2)](\mathbf{Y_1} \cup \mathbf{Y_2}), B. \tag{A2}$$

where $\lambda_1 \doteq \lambda_2$, $qp(cq_1, cq_2) = cq_1' \cup cq_2'$, and cq_1', cq_2' are constructed as follows. Let $\mathbf{Z_1}$ and $\mathbf{Z_2}$ be the non-distinguished (i.e., existential) variables of cq_1 and cq_2, respectively. Rename each $X \in \mathbf{Z_1}$ occurring in cq_2 and each $X \in \mathbf{Z_2}$ occurring in cq_1 to a fresh variable.

VARIABLE ELIMINATION I

$$r_1: \quad H \leftarrow \mathrm{DL}[\lambda_1; cq \cup \{X = t\}](\mathbf{Y}), B. \tag{B1a}$$

$$r': H_{X/t} \leftarrow \mathrm{DL}\big[\lambda_2; cq_{X/t}\big](\mathbf{Y} \setminus \{X, t\}), B_{X/t}. \tag{B2}$$

VARIABLE ELIMINATION II

$$r_2: \quad H \leftarrow \mathrm{DL}[\lambda_1; cq](\mathbf{Y}), X = t, B. \tag{B1b}$$

$$r': H_{X/t} \leftarrow \mathrm{DL}\big[\lambda_2; cq_{X/t}\big](\mathbf{Y} \setminus \{X, t\}), B_{X/t}. \tag{B2}$$

where $\lambda_1 \doteq \lambda_2$, $X \in \mathbf{Y}$, $\cdot_{X/t}$ denotes replacement of variable X by t, and L must be under UNA for Variable Elimination I.

INEQUALITY PUSHING

$$r: \; H \leftarrow \mathrm{DL}[\lambda_1; cq](\mathbf{Y}), X \neq t, B. \tag{C1}$$

$$r': \; H \leftarrow \mathrm{DL}[\lambda_2; cq \cup \{X \neq t\}](\mathbf{Y}), B. \tag{C2}$$

where $\lambda_1 \doteq \lambda_2$, $X \in \mathbf{Y}$, and L must be under UNA. If t is a variable, then also $t \in \mathbf{Y}$.

FACT PUSHING

$$\bar{P} = \left\{ \begin{array}{l} f(\mathbf{c_1}), f(\mathbf{c_2}), \ldots, f(\mathbf{c_\ell}), \\ H \leftarrow \mathrm{DL}[\lambda_1; ucq](\mathbf{Y}), f(\mathbf{Y'}), B. \end{array} \right\} \tag{D1}$$

$$\bar{P}' = \left\{ \begin{array}{l} f(\mathbf{c_1}), f(\mathbf{c_2}), \ldots, f(\mathbf{c_\ell}), \\ H \leftarrow \mathrm{DL}[\lambda_2; fp(ucq)](\mathbf{Y}), B. \end{array} \right\} \tag{D2}$$

where $\lambda_1 \doteq \lambda_2$, $\mathbf{c_j}$ are ground, $\mathbf{Y'} \subseteq \mathbf{Y}$, $ucq = \bigvee_{i=1}^{r} cq_i$, and $fp(ucq) = \bigvee_{i=1}^{r} \left(\bigvee_{j=1}^{\ell} cq_i \cup \{\mathbf{Y'} = \mathbf{c_j}\} \right)$. Applicability for $\bar{P} \subseteq P$ of a general cq-program (L, P) requires that f does not occur in heads of rules in $P \setminus \bar{P}$.

Let H, H', H_i be heads, B, B', B_i be bodies, and r be a rule of form $H \leftarrow a(\mathbf{Y}), B$.

UNFOLDING

$$\bar{P} = \{r\} \cup \{H' \vee a(\mathbf{Y'}) \leftarrow B'.\} \tag{E1}$$

$$\bar{P}' = \bar{P} \cup \{H'\theta \vee H\theta \leftarrow B'\theta, B\theta.\} \tag{E2}$$

where θ is the most general unifier (mgu) of $a(\mathbf{Y})$ and $a(\mathbf{Y'})$ (thus $a(\mathbf{Y}\theta) = a(\mathbf{Y'}\theta)$).

COMPLETE UNFOLDING

$$P = Q \cup \{r\} \cup \{\, r_i : H_i \vee a(\mathbf{Y_i}) \leftarrow B_i.\} \tag{F1}$$

$$P' = (P \setminus \{r\}) \cup \{\, r_i' : H_i\theta_i \vee H\theta_i \leftarrow B_i\theta_i, B\theta_i.\} \tag{F2}$$

where $1 \leq i \leq \ell$, Q has no rules of form r, r_i, no $a(\mathbf{Z}) \in H_i$ is unifiable with $a(\mathbf{Y})$, and θ_i is the mgu of $a(\mathbf{Y})$ and $a(\mathbf{Y_i})$ (thus $a(\mathbf{Y}\theta_i) = a(\mathbf{Y_i}\theta_i)$).

Inequality Pushing (C) If the DL-engine is used under the UNA and supports inequalities in the query language, we can easily rewrite rules with inequality ($\neq$) in the body by pushing it to the cq-query. A rule of form (C1) can be replaced by (C2).

Example 15 Consider the rule

$$big(M) \leftarrow \mathrm{DL}[Wine](W_1), \mathrm{DL}[Wine](W_2), W_1 \neq W_2, \\ \mathrm{DL}[hasMaker](W_1, M), \mathrm{DL}[hasMaker](W_2, M).$$

Here, we want to know all wineries producing at least two different wines. We can rewrite above rule, by Query and Inequality Pushing, to the rule

$$big(M) \leftarrow \mathrm{DL}\left[\begin{array}{c} Wine(W_1), Wine(W_2), W_1 \neq W_2, \\ hasMaker(W_1, M), hasMaker(W_2, M) \end{array}\right](M, W_1, W_2).$$

A similar rule is applicable to a ucq-atom $\mathrm{DL}[\lambda; ucq](\mathbf{Y})$ in place of $\mathrm{DL}[\lambda; cq](\mathbf{Y})$. In that case, we have to add $\{X \neq t\}$ to each cq_i in $ucq = \bigvee_{i=1}^{m} cq_i$.

Example 16 To illustrate what goes wrong if we would not adopt the Unique Name Assumption in Inequality Pushing, take the cq-program (L, P), where $L = \{A(a), B(b)\}$ and $P = \{p(X) \leftarrow \mathrm{DL}[B(X)](X), X \neq a\}$. Applying Inequality Pushing, we get $P' = \{p(X) \leftarrow \mathrm{DL}[B(X), X \neq a](X)\}$. Now, in P we can infer $p(b)$, whereas in P' we cannot infer $p(b)$, since in some models of L it holds that $a = b$.

Fact Pushing (D) Suppose we have a program with "selection predicates," i.e., facts which serve to select a specific property in a rule. We can push such facts into a ucq-atom and remove the selection atom from the rule body.

Example 17 Consider the program P, where we only want to know the children of *joe* and *jill*:

$$P = \left\{ \begin{array}{l} f(joe).\ f(jill). \\ fchild(Y) \leftarrow \mathrm{DL}[isFatherOf](X, Y), f(X). \end{array} \right\}$$

We may rewrite the program to a more compact one with the help of ucq-atoms:

$$f(joe).\ f(jill).$$
$$fchild(Y) \leftarrow \mathrm{DL}\left[\left\{\begin{array}{c} isFatherOf(X, Y), \\ X = joe \end{array}\right\} \vee \left\{\begin{array}{c} isFatherOf(X, Y), \\ X = jill \end{array}\right\}\right](X, Y).$$

Such a rewriting makes sense in situations were *isFatherOf* has many values and thus would lead to query, while uselessly, for each known father-child relationship.

The program $\bar{P}$ in (D1) can be rewritten to $\bar{P}'$ in (D2).

Unfolding (E) and Complete Unfolding (F) Unfolding rules is a standard method for partial evaluation of ordinary disjunctive logic programs under answer set semantics, cf. [37]. It can be also applied in the context of cq-programs, with no special adaptation. After folding rules with (u)cq-atoms in their body into other rules, subsequent Query Pushing might be applied. In this way, inference propagation can be shortcut.

The following results state that the above rewritings preserve equivalence. Let $P \equiv_L Q$ denote that (L, P) and (L, Q) have the same answer sets.

Theorem 4 *Let r and r' be rules of form $(\Theta 1)$ and $(\Theta 2)$, respectively, $\Theta \in \{A, B, C\}$, where UNA is adopted for Variable Elimination I and Inequality Pushing. Let (L, P) be a cq-program with $r \in P$. Then, $P \equiv_L (P \setminus \{r\}) \cup \{r'\}$.*

Theorem 5 *Let $\bar{P}$ and $\bar{P}'$ be rule sets of form $(\Theta 1)$ and $(\Theta 2)$, respectively, $\Theta \in \{D, E\}$. Let (L, P) be a cq-program such that $\bar{P} \subseteq P$. Then, $\bar{P} \equiv_L \bar{P}'$ and $P \equiv_L (P \setminus \bar{P}) \cup \bar{P}'$.*

Theorem 6 *Let P and P' be rule sets of form* (F1) *and* (F2)*. Then, $P \equiv_L P'$.*

In the remainder of this section, we formally prove these results, where we first consider Theorem 4, and then Theorems 5 and 6 in Section 5.2.

5.1 Proof of Theorem 4

We first state some useful Lemmas.

Lemma 3 *Let* $a = \mathrm{DL}[\lambda_1; cq_1](\mathbf{Y_1})$ *and* $b = \mathrm{DL}[\lambda_2; cq_2](\mathbf{Y_2})$ *be two cq-atoms such that* $\lambda_1 \doteq \lambda_2$*, and* θ *be a ground substitution over domain* $\mathbf{Y_1} \cup \mathbf{Y_2}$*. Then,* $I \models_L a\theta$ *and* $I \models_L b\theta$ *iff* $I \models_L \mathrm{DL}[\lambda_1; (cq_1' \cup cq_2')\theta]()$.

Proof ($\Rightarrow$) Suppose $I \models_L a\theta$ and $I \models_L b\theta$. Therefore both $L \cup \lambda_1(I) \models \phi_{cq_1}(\mathbf{Y_1}\theta)$ and $L \cup \lambda_2(I) \models \phi_{cq_2}(\mathbf{Y_2}\theta)$ hold. Thus, $L \cup \lambda_1(I) \models \phi_{cq_1'}(\mathbf{Y_1}\theta) \wedge \phi_{cq_2'}(\mathbf{Y_2}\theta)$ because of $\lambda_1 \doteq \lambda_2$, and this implies that $L \cup \lambda_1(I) \models \phi_{cq_1' \cup cq_2'}((\mathbf{Y_1} \cup \mathbf{Y_2})\theta)$, because for the rewritten non-distinguished variables $\mathbf{Y_1'} \cup \mathbf{Y_2'}$ of $cq_1'(\mathbf{Y_1}\theta) \cup cq_2'(\mathbf{Y_2}\theta)$, it holds that $\mathbf{Y_1'} \cap \mathbf{Y_2'} = \emptyset$ due to the variable renaming used during the rewriting. Consequently, $I \models_L \mathrm{DL}[\lambda_1; (cq_1' \cup cq_2')\theta]()$.

($\Leftarrow$) Let $I \models_L \mathrm{DL}[\lambda_1; (cq_1' \cup cq_2')\theta]()$, hence $L \cup \lambda_1(I) \models \phi_{cq_1' \cup cq_2'}((\mathbf{Y_1} \cup \mathbf{Y_2})\theta)$ implies that both $L \cup \lambda_1(I) \models \phi_{cq_1}(\mathbf{Y_1}\theta)$ and $L \cup \lambda_1(I) \models \phi_{cq_2}(\mathbf{Y_2}\theta)$ hold. From $\lambda_1 \doteq \lambda_2$, we conclude that $L \cup \lambda_2(I) \models \phi_{cq_2}(\mathbf{Y_2}\theta)$, hence $I \models_L a\theta$ and $I \models_L b\theta$.

For the case were we have UCQs $ucq_1 = \bigvee_{i=1}^{r_1} cq_{1,i}$ and $ucq_2 = \bigvee_{i=1}^{r_2} cq_{2,i}$ in place of cq_1 and cq_2, respectively, the proof is straightforward by using $\bigvee_{i=1}^{r_1} \left(\bigvee_{j=1}^{r_2} cq_{1,i}' \cup cq_{2,j}' \right)$ instead of $cq_1' \cup cq_2'$. □

Lemma 4 *Let* $a = \mathrm{DL}[\lambda_1; cq \cup \{X = t\}](\mathbf{Y})$ *and* $b = \mathrm{DL}[\lambda_2; cq_{X/t}](\mathbf{Y} \setminus \{X, t\})$ *be cq-atoms such that* $\lambda_1 \doteq \lambda_2$*, and* θ *be a ground substitution over domain* $\mathbf{Y}$*. The following statements hold:*

(1) If $t \in \mathbf{Y}$ *and* L *is under UNA, then* $I \models_L a\theta$ *iff* $I \models_L b\theta$.
(2) If $t \notin \mathbf{Y}$*, then* $I \models_L a\theta$ *iff* $I \models_L b\theta$.

Proof (1) ($\Rightarrow$) Suppose $I \models_L a\theta$. $I \models_L (X = t)\theta$ and UNA in L imply that $X\theta$ and $t\theta$ denote the same individual symbol. Hence, $cq(\mathbf{Y}\theta) = cq_{X/t}(\mathbf{Y} \setminus \{X, t\})\theta$ (even if X or t do not occur in the query atoms in cq) and $\lambda_1 \doteq \lambda_2$ implies $I \models_L b$.
($\Leftarrow$) Now suppose $I \models_L b\theta$. Since X does not appear in b, we replace occurrences of t in $cq_{X/t}(\mathbf{Y} \setminus \{X, t\})$ to X such that $cq(\mathbf{Y})$ is obtained. Moreover, setting $X\theta$ to $t\theta$ implies $(X = t)\theta$. Therefore, $I \models_L a\theta$.

(2) The proof is essentially the same as (1). Here, we do not need UNA for replacing X by t, since t is not in the domain of θ. $X = t$ assures then that both terms denote the same individual in the universe. See also Lemma 6.1 in [29]. □

Lemma 5 *Let* $a = \mathrm{DL}[\lambda_1; cq](\mathbf{Y})$ *and* $b = \mathrm{DL}[\lambda_2; cq \cup \{X \neq t\}](\mathbf{Y})$ *be cq-atoms such that* $\lambda_1 \doteq \lambda_2$*,* $X \in \mathbf{Y}$*, and* θ *be a ground substitution over a domain* $\mathbf{Y}$*. Then, for* L *being under UNA,* $I \models_L a\theta$ *and* $I \models_L (X \neq t)\theta$ *iff* $I \models_L b\theta$.

Proof ($\Rightarrow$) Suppose $I \models_L a\theta$ and $I \models_L (X \neq t)\theta$. We derive that $X\theta$ and $t\theta$ are syntactically different. Hence, $cq(\mathbf{Y}\theta) \cup \{X \neq t\}\theta$ holds in $L \cup \lambda_2(I)$ by $\lambda_1 \doteq \lambda_2$, therefore $I \models_L b\theta$.

($\Leftarrow$) Now we assume that $I \models_L b\theta$. Since $L \cup \lambda_2(I)$ satisfies $cq(\mathbf{Y}\theta)$ and $\{X \neq t\}\theta$, we conclude that $L \cup \lambda_1(I) \models_L cq(\mathbf{Y}\theta)$ and hence $I \models_L a\theta$ and $I \models_L (X \neq t)\theta$. □

In the following, let ρ be a rule of form r_1, r_2 (i.e., of form (B1a) resp. (B1b)), or r (i.e., (A1) resp. (C1)). Let r' be a rule of form (A2), (B2), and (C2), resp. Then, let $P' = (P \setminus \{\rho\}) \cup \{r'\}$, where ρ and r' are equivalent rules according to the rewriting rules (A), (B), or (C). We will show now that I is a (strong) answer set of (L, P) iff I is a (strong) answer set of (L, P').

Proof (for rewriting rules (A), (B), *and* (C) *of Theorem 4)* We first show for positive cq-programs (L, P), I is a minimal model of (L, P) iff I is a minimal model of (L, P').

($\Rightarrow$) Suppose I is a minimal model of (L, P). Towards a contradiction, assume I is not a model of (L, P'). Thus, for a ground substitution θ, there is a ground version of r' in $\mathit{ground}(P')$, $r'\theta$, such that $I \not\models_L H(r'\theta)$ and $I \models_L B(r'\theta)$. Since $I \models_L P$, in particular $\rho\theta \in \mathit{ground}(P)$, we get that (i) $I \models_L B(\rho\theta)$ and $I \models_L H(\rho\theta)$, or (ii) $I \not\models_L B(\rho\theta)$. In case of (i), we get a contradiction for $I \not\models_L H(r'\theta)$, since $I \models_L H(\rho\theta)$ and $H(\rho\theta) = H(r'\theta)$, hence I is a model of (L, P'). Now for case (ii), we have that $I \not\models_L B(\rho\theta)$, hence a literal of $B(\rho\theta)$ is false in I. If $a \in B(\rho\theta)$ is false in I, then $a \in B(r'\theta)$ is false in I by Lemma 3 or 5 (resp. 4) for ρ of form r (resp. r_1 or r_2), which is a contradiction for $I \models_L B(r'\theta)$. Again, I is a model of (L, P').

Now assume that $J \subset I$ is a minimal model of (L, P'), therefore J is not a model of (L, P). For a ground substitution θ, there is a ground version of $\rho\theta$ in $\mathit{ground}(P)$ such that $J \not\models_L H(\rho\theta)$ and $J \models_L B(\rho\theta)$. Since $J \models_L r'\theta$ for a ground $r'\theta \in \mathit{ground}(P')$, we obtain the following cases. If $J \models_L B(r'\theta)$ and $J \models_L H(r'\theta)$, we derive a contradiction, since $H(\rho\theta) = H(r'\theta)$. Otherwise, if $J \not\models_L B(r'\theta)$, we derive a contradiction at $J \models_L B(\rho\theta)$, since Lemma 3, 4, and 5 applies here as well. Consequently, I is a minimal model of (L, P').

($\Leftarrow$) Let I be a minimal model of (L, P'). We assume now that I is not a model of (L, P). Thus, for a ground substitution θ, there is a ground version of ρ in $\mathit{ground}(P)$, $\rho\theta$, such that $I \not\models_L H(\rho\theta)$ and $I \models_L B(\rho\theta)$. By ($\Rightarrow$), we derive a contradiction, hence I is a model of (L, P).

To show that I is also a minimal model of (L, P), assume the contrary, there is a $J \subset I$ such that J is a minimal model of (L, P). This entails that J is not a model for P'. Again, using ($\Rightarrow$) and Lemma 3, 4, or 5, we conclude that J cannot be a minimal model of (L, P), hence I is a minimal model of (L, P).

Now we establish the proof for rewriting rules (A), (B), and (C) in Section 5.

Let I be a strong answer set of (L, P). Since sP_L^I and $sP_L'^I$ are positive cq-programs, we show now that I is a minimal model of sP_L^I iff I is a minimal model of $sP_L'^I$. To this end, consider a ground rule of form $\rho \in P$ with $\rho\theta \in \mathit{ground}(P)$, where θ is a ground substitution. We distinguish the cases:

(i) $\rho\theta \notin sP_L^I$: this implies that $I \not\models_L a$ for $a \in B^+(\rho\theta) \cap DL_P^?$, or $I \models_L l$ for $l \in B^-(\rho\theta)$. We conclude that $r'\theta \notin sP_L'^I$, since whenever $b \in B^+(\rho\theta) \cap DL_P^?$ is used in the process of the rewriting, and I does not satisfy b, and by the actual Lemma 3, 4, or 5, I does not satisfy $b' \in B^+(r'\theta) \cap DL_{P'}^?$ either, where b' is the outcome of the resp. rewriting rule. Thus, $sP_L^I = sP_L'^I$, which implies I is a minimal model of sP_L^I iff I is a minimal model of $sP_L'^I$.

(ii) $\rho\theta \in sP_L^I$: then, $I \models_L a$ for all $a \in B^+(\rho\theta) \cap DL_P^?$, and $I \not\models_L l$ for all $l \in B^-(\rho\theta)$. Therefore, by applying the actual Lemma 3, 4, or 5, $r'\theta \in sP_L'^I$. Hence, $I \models_L \rho\theta$ iff $I \models_L r'\theta$, so I is a minimal model of sP_L^I iff I is a minimal model of $sP_L'^I$.

Therefore, (L, P) has the same answer sets as (L, P'). □

5.2 Proofs for Theorems 5 and 6

We split the proofs for Theorems 5 and 6 in two parts, the first part considers rewriting rule (D) of Theorem 5, while the second part deals with rewriting rules (E) and (F) of Theorems 5 and 6, respectively. We will show for each part of the proof that I is a strong answer set of (L, P) iff I is a strong answer set of (L, P'). Again, we first state some useful lemmas.

Lemma 6 *Let r and r' be positive cq-rules of form*

$$r: \ H \leftarrow \mathrm{DL}\left[\lambda_1; \bigvee_{i=1}^{r} cq_i\right](\mathbf{Y}), f(\mathbf{Y'}), B$$

and

$$r': \ H \leftarrow \mathrm{DL}\left[\lambda_2; \bigvee_{i=1}^{r}\left(\bigvee_{j=1}^{\ell} cq_i \cup \{\mathbf{Y'} = \mathbf{c_j}\}\right)\right](\mathbf{Y}), B,$$

respectively, where $\lambda_1 \doteq \lambda_2$ and $\mathbf{Y'} \subseteq \mathbf{Y}$, let θ be a ground substitution over a domain $\mathbf{Y}$, and let I be a Herbrand interpretation such that $f(\mathbf{c_j}) \in I$ for $1 \leq j \leq \ell$ are all the literals with predicate f in I. Then, $I \models_L r\theta$ if and only if $I \models_L r'\theta$.

Proof ($\Rightarrow$) Assume $I \models_L f(\mathbf{c_j})$ for $1 \leq j \leq \ell$ and $I \models_L r\theta$ hold. By $I \models_L r\theta$, either (i) $I \models_L H(r\theta)$ and $I \models_L B^+(r\theta)$ or (ii) $I \not\models_L B^+(r\theta)$.

(i) $I \models_L B^+(r\theta)$ implies $I \models_L f(\mathbf{Y'}\theta)$. Since $I \models_L f(\mathbf{c_j})$ for $1 \leq j \leq \ell$, we obtain that $f(\mathbf{Y'}\theta) = f(\mathbf{c})$ for a $\mathbf{c} \in \{\mathbf{c_1}, \ldots, \mathbf{c_\ell}\}$. Thus, the disjunction over $cq_i\theta \cup \{\mathbf{Y'}\theta = \mathbf{c}_j\}$ for $1 \leq j \leq \ell$ must hold for some $\mathbf{c}_j = \mathbf{c}$. By $\lambda_1 \doteq \lambda_2$, we obtain $L \cup \lambda_2(I) \models \bigvee_{i=1}^{r}\left(\bigvee_{j=1}^{\ell} cq_i\theta \cup \{\mathbf{Y'}\theta = \mathbf{c_j}\}\right)$, therefore I satisfies

$$\mathrm{DL}\left[\lambda_2; \bigvee_{i=1}^{r}\left(\bigvee_{j=1}^{\ell} cq_i\theta \cup \{\mathbf{Y'}\theta = \mathbf{c_j}\}\right)\right]()$$

under L, and $I \models_L r'\theta$.

(ii) We obtain another two cases. First, $I \not\models_L B\theta$ implies $I \not\models B^+(r'\theta)$, hence $I \models_L r'\theta$. Secondly, some of $f(\mathbf{Y'}\theta)$ and $\mathrm{DL}[\lambda_1; \bigvee_{i=1}^{r} cq_i\theta]()$ are not satisfied under L. By $\lambda_1 \doteq \lambda_2$, this implies that I does not satisfy

$$\mathrm{DL}\left[\lambda_2; \bigvee_{i=1}^{r}\left(\bigvee_{j=1}^{\ell} cq_i\theta \cup \{\mathbf{Y'}\theta = \mathbf{c_j}\}\right)\right]()$$

under L either, thus $I \models_L r'\theta$.

($\Leftarrow$) Assume $I \models_L r'\theta$. Either $I \models_L H(r'\theta)$ and $I \models_L B^+(r'\theta)$, or $I \not\models_L B^+(r'\theta)$. Similar to the ($\Rightarrow$) direction, we obtain now that $I \models_L r\theta$. □

The next lemma is a generalization of a similar lemma in [37] for ordinary positive disjunctive logic programs to cq-programs.

Lemma 7 *Let (L, P) be a positive cq-program and I a minimal model of (L, P). Then, an atom a is in I iff there is a ground rule $a \vee H \leftarrow B$ from P such that $I \setminus \{a\} \models B$ and $I \setminus \{a\} \not\models H$.*

Proof ($\Rightarrow$) Suppose for some atom a in I, there is no ground rule $a \vee H \leftarrow B$ from P such that $I \setminus \{a\} \models_L B$ and $I \setminus \{a\} \not\models_L H$. Then, for each ground rule r of the form $a \vee H \leftarrow B$, $I \setminus \{a\} \not\models_L B$ or $I \setminus \{a\} \models_L H$; hence it holds that $I \setminus \{a\} \models_L B$ implies $I \setminus \{a\} \models_L H$. In this case, $I \setminus \{a\}$ satisfies each rule r and becomes a model of (L, P), which contradicts the assumption that I is a minimal model. Hence the result follows.

($\Leftarrow$) Assume that a is not in I. Then $I \setminus \{a\} = I$, and for a ground rule $a \vee H \leftarrow B$ in P, $I \models_L B$ and $I \not\models_L H$ imply $a \in I$, which is a contradiction. □

We are now ready to give proofs for Theorems 5 and 6.

Proof (for rewriting rule (D) *of Theorem 5)* For positive cq-programs $(L, \bar{P})$ and $(L, \bar{P}')$ (where $\bar{P}' = (\bar{P} \setminus \{r\}) \cup \{r'\}$) the minimal models coincide; this follows from Lemma 6 and the fact that for every minimal model I of $(L, \bar{P})$ resp. $(L, \bar{P}')$, it holds that $f(\mathbf{c}) \in I$ iff $\mathbf{c} = \mathbf{c_j}$ for some $j \in \{1, \ldots, \ell\}$.

Now let (L, P) and (L, P') be positive cq-programs, where $\bar{P} \subseteq P$ and $P' = (P \setminus \bar{P}) \cup \bar{P}'$ such that f does not occur in the heads of $P \setminus \bar{P}$. Since P' is logically equivalent to P, we obtain that the minimal models of (L, P) and (L, P') coincide.

For the general case, $(L, \bar{P})$ and $(L, \bar{P}')$ are cq-programs without restriction, we show now that $s\bar{P}_L^I = (s\bar{P}_L^I)'$, where $(s\bar{P}_L^I)'$ is obtained from applying rewriting rule (D) to the ground program $(L, s\bar{P}_L^I)$.

Let I be a strong answer set of $(L, \bar{P})$. I is a minimal model of the positive cq-program $(L, s\bar{P}_L^I)$. As shown above, I is a minimal model of $(L, s\bar{P}_L^I)$ iff I is a minimal model of $(L, (s\bar{P}_L^I)')$. Consider $r \in \bar{P}$, for a ground substitution θ of r; we obtain the case distinction:

(i) $I \not\models_L B^-(r\theta)$ and $I \models_L B^+(r\theta) \cap DL_P^?$: In this case, $r\theta \in s\bar{P}_L^I$, therefore $r'\theta \in (s\bar{P}_L^I)'$. Since $r' \in \bar{P}'$ and (i) hold, we conclude that $r' \in s\bar{P}'^I_L$.

(ii) for some $l \in B^-(r\theta)$, $I \models_L l$, or for some $a \in B^+(r\theta) \cap DL_P^?$, $I \not\models_L a$ hold: In this case, $r\theta \notin s\bar{P}_L^I$, therefore $r'\theta \notin (s\bar{P}_L^I)'$. Since $r' \in \bar{P}'$ and (ii) hold, we conclude that $r' \notin s\bar{P}'^I_L$.

Thus, $s\bar{P}'^I_L = (s\bar{P}_L^I)'$, that is, the reduct of the rewritten rules $\bar{P}'$ is equal to the rewritten rules of the reduct of $\bar{P}$, hence I is a minimal model of $(L, (s\bar{P}_L^I)')$ iff I is a minimal model of $(L, s\bar{P}'^I_L)$. Therefore, I is a strong answer set of $(L, \bar{P})$ iff I is a strong answer set of $(L, \bar{P}')$.

Now we are ready to finish the proof and show coincidence of strong answer sets for unrestricted (L, P) and (L, P'), where $\bar{P} \subseteq P$ and $P' = (P \setminus \bar{P}) \cup \bar{P}'$ such that f does not occur in the heads of $P \setminus \bar{P}$. Since P' is logically equivalent to P, we obtain that the strong answer sets of (L, P) and (L, P') are in one-to-one correspondence. □

Proof (for rewriting rules (E) *and* (F) *of Theorem 6)* We first show that for positive cq-programs $(L, \bar{P})$ and $(L, \bar{P}')$, the minimal models coincide.

To this end, let $\bar{P}$ consist of the positive cq-rules

$$r : H \leftarrow a(\mathbf{Y}), B$$

and

$$r_1 : H' \vee a(\mathbf{Y'}) \leftarrow B',$$

where $B = b_1, \ldots, b_m$, $B' = b'_1, \ldots, b'_n$, $H = a_1 \vee \cdots \vee a_k$, $H' = a'_1 \vee \cdots a'_l$, and $DL_{\bar{P}} = DL^+_{\bar{P}}$, such that for an mgu θ of $a(\mathbf{Y})$ and $a(\mathbf{Y'})$, $a(\mathbf{Y}\theta) = a(\mathbf{Y'}\theta)$. And let $\bar{P}'$ consist of all the rules in $\bar{P}$ and the positive cq-rule

$$r'_1 : H'\theta \vee H\theta \leftarrow B'\theta, B\theta.$$

Due to the unfolding rule (E), $\bar{P}' = \bar{P} \cup \{r'_1\}$, which is logically equivalent to $\bar{P}$, hence $(L, \bar{P})$ and $(L, \bar{P}')$ have the same minimal models and thus $\bar{P} \equiv_L \bar{P}'$. Similarly, when $\bar{P} \subseteq P$ for an arbitrary positive set of cq-rules P and $P' = P \cup \{r'_1\}$, I is a minimal model of P iff I is a minimal model of P'.

Now we show that in case of Complete Unfolding (F), the positive cq-program (L, P) has the same minimal models as the positive cq-program (L, P').

Let r be as above, Q be a set of positive cq-rules such that no rules of form r and r_i appear in it, where r_i is a cq-rule of form

$$r_i : \ H_i \vee a(\mathbf{Y_i}) \leftarrow B_i \qquad (1 \leq i \leq \ell),$$

such that each H_i either does not contain a literal of form $a(\mathbf{Z})$, or no $a(\mathbf{Z}) \in H_i$ is unifiable with $a(\mathbf{Y})$; and P be the set of cq-rules $Q \cup \{r\} \cup \{r_i \mid 1 \leq i \leq \ell\}$, while $P' = (P \setminus \{r\}) \cup \{r'_i : H_i\theta_i \vee H\theta_i \leftarrow B_i\theta_i, B\theta_i \quad (1 \leq i \leq \ell)\}$ for mgus θ_i such that $a(\mathbf{Y})$ and $a(\mathbf{Y_i})$ unify.

($\Rightarrow$) Assume I is a minimal model of (L, P). Since I satisfies ground versions of r and all ground r_i, we obtain that I satisfies all of the corresponding ground versions of r'_i. Thus, we get that I is a model of (L, P'). Towards a contradiction, assume that J is a minimal model of (L, P'), such that $J \subset I$. J is not a model of (L, P) and a ground r must occur unsatisfied in $\mathit{ground}(P)$, thus for a ground substitution η of r, $J \not\models_L r\eta$, which implies $J \models_L B(r\eta)$ and $J \not\models_L H(r\eta)$. By $J \models_L B(r\eta)$, it follows that $J \models_L a(\mathbf{Y}\eta)$. By Lemma 7, we get for a ground $r'\sigma$ of a rule $r' \in P'$, i.e., either $r'_i\sigma$ or $r_i\sigma$, where σ is a ground substitution, $a(\mathbf{Y}\eta) \in H(r'\sigma)$. Since $r'\sigma = r'\sigma\eta$, we get $a(\mathbf{Y}\eta) = a(\mathbf{Y}\sigma\eta)$, and hence $a(\mathbf{Y}\sigma\eta) \in H(r'\sigma\eta)$. From $J \models_L B(r\eta)$ and $J \not\models_L H(r\eta)$, we conclude $J \models_L B(r\sigma\eta)$ and $J \not\models_L H(r\sigma\eta)$. We distinguish the cases:

(i) $r' = r'_i$: Assume that $r'\sigma\eta$ is a ground instance of r'_i and $a(\mathbf{Y}\sigma\eta) \in H(r'_i\sigma\eta)$. The mgu θ_i of $a(\mathbf{Y})$ and $a(\mathbf{Y_i})$ implies $\sigma\eta = \theta_i\rho$ for some ρ. Since $a(\mathbf{Y}\theta_i) = a(\mathbf{Y_i}\theta_i)$, we get $a(\mathbf{Y}\sigma\eta) = a(\mathbf{Y_i}\sigma\eta)$. Since $a(\mathbf{Y})$ is not unifiable with any literal in H_i of r_i, $a(\mathbf{Y_i}\sigma\eta) \notin H_i\sigma\eta$. Thus, $a(\mathbf{Y_i}\sigma\eta)$ must be one of $H\theta\sigma\eta$. $J \models_L a(\mathbf{Y}\sigma\eta)$ implies $J \models_L H\theta\sigma\eta$, but this contradicts $J \not\models_L H(r\sigma\eta)$. Therefore, I is also a minimal model of (L, P').

(ii) $r' = r_i$: Now suppose $r'\sigma\eta$ is a ground instance of r_i with $a(\mathbf{Y}\sigma\eta) = a(\mathbf{Y_i}\sigma\eta)$. Applying Lemma 7, from $a(\mathbf{Y}\sigma\eta) \in J$, we conclude $J \setminus \{a(\mathbf{Y}\sigma\eta)\} \models_L B(r_i\sigma\eta)$ and $J \setminus \{a(\mathbf{Y}\sigma\eta)\} \not\models_L H_i\sigma\eta$. Since $a(\mathbf{Y}\sigma\eta) \notin B(r_i\sigma\eta)$, we get $J \models_L B(r_i\sigma\eta)$. Since $a(\mathbf{Y})$ is not unifiable with any literal in H_i of r_i, we obtain $a(\mathbf{Y}\sigma\eta) \notin H_i\sigma\eta$ and also $J \not\models_L H_i\sigma\eta$. $J \models_L B(r\sigma\eta)$ and $J \not\models_L H(r\sigma\eta)$ now implies that $J \not\models_L r'_i\sigma\eta$. Since $a(\mathbf{Y}\sigma\eta) = a(\mathbf{Y_i}\sigma\eta)$ and $a(\mathbf{Y}\theta_i) = a(\mathbf{Y_i}\theta_i)$, we get $\sigma\eta = \theta_i\rho$ for some ρ, thus $r'_i\sigma\eta$ is a ground instance of $r'_i \in P'$. Hence, $r'_i\rho \in \mathit{ground}(P')$ is not satisfied, which contradicts the assumption that J is a model of (L, P'). Therefore, I is a minimal model of (L, P').

($\Leftarrow$) Let I be a minimal model of (L, P'). Assuming that I is not a model of (L, P), then $I \not\models_L r\eta$ for a ground substitution η. This implies $I \not\models_L H(r\eta)$ and $I \models_L B(r\eta)$, which in turn guarantees that $I \models_L a(\mathbf{Y}\eta)$. By Lemma 7, we obtain for a ground

version $r'\sigma$ of a rule $r' \in \bar{P}'$, i.e., either $r'_i\sigma$ or $r_i\sigma$, where σ is a ground substitution, $a(\mathbf{Y}\eta) \in H(r'\sigma)$. We will now apply a similar proof to the ($\Rightarrow$) direction and get the desired contradictions. Thus, I is a model of (L, P). Now we show that I is in fact a minimal model. To this end, assume that there is a minimal model $J \subset I$ of (L, P). Proceeding as in ($\Rightarrow$), J is also a minimal model of (L, P'), which contradicts our assumption that I is a minimal model of (L, P'), hence I is also a minimal model of (L, P).

Now we turn our attention to the general case, that is, (L, P) and (L, P') are cq-programs without restrictions. We show that $sP'^I_L = (sP^I_L)'$, where $(sP^I_L)'$ is the complete unfolded positive cq-program of the reduct of (L, P).

Let I be a strong answer set of (L, P). I is a minimal model of (L, sP^I_L), which is a positive program. Hence, by our first part of the proof, I is a minimal model of (L, sP^I_L) iff I is a minimal model of $(L, (sP^I_L)')$. Let us consider $r, r_i \in P$. We have an mgu θ_i for $a(\mathbf{Y}\theta_i) = a(\mathbf{Y_i}\theta_i)$, and for a ground substitution η, $a(\mathbf{Y}\eta) = a'(\mathbf{Y_i}\eta)$. This implies that $\eta = \theta_i\rho$ for some substitution ρ. We now distinguish the cases:

(i) $I \not\models_L B^-(r\eta)$, $I \not\models_L B^-(r_i\eta)$, $I \models_L B^+(r\eta) \cap DL^?_P$, and $I \models_L B^+(r_i\eta) \cap DL^?_P$: Here, $r\eta, r_i\eta \in sP^I_L$. By our unfolding rule, we get that $r'_i\eta \in (sP^I_L)'$. Since $r'_i \in \bar{P}'$, $\eta = \theta_i\rho$, and (i) hold, we conclude $r'_i\eta$ is in sP'^I_L.

(ii) for some $l \in B^-(r\eta) \cup B^-(r_i\eta)$, $I \models_L l$, or for some $a \in (B^+(r\eta) \cup B^+(r_i\eta)) \cap DL^?_P$, $I \not\models_L a$ hold: In this case, some of $r\eta$ and $r_i\eta$ is not in sP^I_L. Therefore, $r'_i\eta$ is not in $(sP^I_L)'$. Since $r'_i \in P'$, $\eta = \theta_i\rho$, and (ii) hold, we conclude $r'_i\eta$ is not in sP'^I_L either.

Thus, $sP'^I_L = (sP^I_L)'$, i.e., the reduct of the complete unfolded program P' and the complete unfolded reduct of P coincide. This implies I is a minimal model of $(L, (sP^I_L)')$ iff I is a minimal model of (L, sP'^I_L). Therefore, I is a strong answer set of (L, P'). □

6 Rewriting Algorithms

Based on the results above, we describe algorithms which combine rewriting rules into a single module for optimizing cq-programs. The optimization process takes several steps. In each step, a special rewriting algorithm works on the result handed over by the preceding step. Note that, in general, some of the rewriting rules might eliminate some predicate name from a given program. This might not be desired if such predicate names play the role of output predicates. Indeed, usually a program P contains auxiliary rules conceived for importing knowledge from an ontology, or to compute intermediate results, while important information, from the user's point of view, is carried by output predicates. We introduce thus a set F of *filter predicates* which are explicitly preserved from possible elimination.

The first step performs unfolding, taking filter predicates from F into account. That is, only literals with a predicate from F are kept.

Algorithm 1 uses the function *factpush*(P) for Fact Pushing. This function tries to turn a program P into a more efficient one by merging rules according to the Fact Pushing (D) equivalence in Section 5. The algorithm also combines filtering and unfolding (see equivalences (E) and (F)) using *unfold*(a, r_H, r_B), which takes two rules r_H and r_B and returns the unfolding of r_B with r_H w.r.t. a literal a. Note that *do_unfold*(a, r_H, r_B, P) is a generic function for deciding whether the unfolding of a rule r_H in r_B w.r.t. a given program P and a literal a can be done (or is worth being

Algorithm 1: *merge*(P, F): Merge cq-rules in program P w.r.t. F

Input: Program P, Filter $F = \{p_1, \ldots, p_n\}$
Result: Unfolded program P

repeat
  $P^l = P = \mathit{factpush}(P)$
  $C = \{a, a' \mid \exists r, r' \in P : a' \in H(r'), a \in B^+(r),$ and a' unifiable with $a\}$
  if $C \neq \emptyset$ **then**
    choose $a \in C$
    $P' = \emptyset$
    $R_H = \{r \in P \mid a$ unifies with $a' \in H(r)\}$
    $R_B = \{r \in P \mid a$ unifies with $a' \in B^+(r)\}$
    stop_unfold = true
    forall $r_B \in R_B$ **do**
      forall $r_H \in R_H$ **do**
        if *do_unfold*(a, r_H, r_B, P) **then**
          stop_unfold = false
          add r_H and *unfold*(a, r_H, r_B) to P'
          if $\mid \{b \in H(r_H)$ *such that* b *unifies with* $a\} \mid > 1$ **then** add r_B to P'
        else
          add r_H and r_B to P'
        end
      end
    end
    $P = P' \cup (P \setminus (R_B \cup R_H))$
  end
until $P^l = P$ *or stop_unfold is true*
return *filter*(P, F)

Algorithm 2: *RuleOptimizer*(P): Optimize the bodies of all cq-rules in P

foreach $r \in P$ *such that* $B^+(r) \neq \emptyset$ **do**
  choose $b \in B^+(r)$
  $B^+(r) = \mathit{BodyOptimizer}\big(b, B^+(r) \setminus \{b\}, \emptyset, \emptyset\big)$
  $r = \mathit{VarElim}(r)$
end
return P

done); this decision may be taken, e.g., using a cost model (as we will see later in this section). *do_unfold* may also use, e.g., an internal counter for the numbers of iterations or rule unfoldings, and return false if a threshold is exceeded. The case where more than one atom in the head of r_B unifies with a must be considered in *do_unfold*, because we cannot apply the complete unfolding rewriting rule in this scenario. The function *filter*(P, F) eliminates rules which have no influence on the filtered output. Such rules are those of form $H \leftarrow B$ where H is nonempty and has no predicate from F and no literal a unifiable either (i) with some literal in the body of a rule from P, or (ii) with some literal in a disjunctive rule head in P, or (iii) with the opposite of some literal in a rule head in P.

The following theorem states that Algorithm 1 works correctly. For finite sets of cq-rules P and Q, a DL-KB L, and a set of predicates F, let $P \equiv_L^F Q$ denote that (L, P) and (L, Q) have the same strong answer sets w.r.t. F. That is, if M is a strong answer set of (L, P), then (L, Q) has a strong answer set N such that $M \setminus \{p(\mathbf{c}) \mid p \notin F\} = N \setminus \{p(\mathbf{c}) \mid p \notin F\}$ and vice versa.

Algorithm 3: *BodyOptimizer*(o, B, C, O): Push queries in body B w.r.t. o

Input: atom o, body B, carry atoms C, and optimized body O
Result: pushed optimized body B

if $B \neq \emptyset$ **then**
  choose $b \in B$
  if $do\_push(o, b)$ **then**
    $o = push(o, b)$
  else
    $C = C \cup \{b\}$
  end
  if $|B| > 1$ **then**
    return $BodyOptimizer(o, B \setminus \{b\}, C, O)$
  else if $|C| \neq \emptyset$ **then**
    choose $c \in C$
    return $BodyOptimizer(c, C \setminus \{c\}, \emptyset, O \cup \{o\})$
  end
end
return $O \cup \{o\}$

Algorithm 4: *VarElim*(r): Eliminate variables in r

Input: cq-rule r
Result: optimized r

forall $a = \mathrm{DL}[\lambda; cq](\mathbf{Y})$ *in* $B^+(r)$ *s.t.* $X \in \mathbf{Y}$ **do**
  if *either* $X = t$ *is in* cq *and* L *is under UNA, or* $X = t$ *is in* $B^+(r)$ **then**
    $r = H(r)_{X/t} \leftarrow \mathrm{DL}[\lambda; cq_{X/t}](\mathbf{Y} \setminus \{X, t\}),\ (B^+(r) \setminus \{a\})_{X/t},\ not\ B^-(r)_{X/t}$
  end
end
return r

Theorem 7 *For a cq-program* (L, P) *and filter* F, $P \equiv_L^F merge(P, F)$.

Proof Algorithm 1 first copies P to P^l and applies Fact Pushing to P. Now suppose that we cannot do the Unfolding part of the algorithm, i.e., $C = \emptyset$ and only the Fact Pushing step takes part in the optimization process. $merge(P)$ eventually halts, since we cannot push any facts in P, therefore $P = P^l$. By part (D) of Theorem 5, Fact Pushing preserves the answer sets, hence $P \equiv_L^F merge(P, F)$.

Now assume that we unfold some rules in P, i.e., $C \neq \emptyset$. Some rules in P have a common atom $a \in C$ in the head and in the positive body, while a does not occur in the negative part of any rule in P. These a can be unfolded using the Unfolding rule (E). Algorithm 1 then proceeds by possibly unfolding all the rules $r_H \in R_H$ and $r_B \in R_B$ by means of $unfold(a, r_H, r_B)$, i.e., folding r_H into r_B w.r.t. a. Since $pred(H(r)) \cap \mathcal{P} \neq \emptyset$, we always add r to P'. Thus, either $unfold(a, r_H, r_B) \cup \{r_H\}$ or $\{r_H, r_B\}$ are contained in P', depending on the outcome of *do_unfold*. Eventually, after all the unfolding had been carried out for a particular $a \in C$, we replace P by $P' \cup (P \setminus (R_B \cup R_H))$, which amounts to replacing P by $(P \setminus \bar{P}) \cup \bar{P}'$ for all possible $\bar{P}$ and $\bar{P}'$, which are defined as in Theorem 5. Therefore, by part (E) of Theorem 5, one unfolding step for an $a \in C$ preserves the answer sets, hence after all other atoms of C had been unfolded, we still have the same answer sets as the program we started the unfolding procedure with. Ultimately, for this case, the unfolding procedure halts, since in each round of $merge(P, F)$'s main-loop, we check whether P equals P^l, the program P from which we started an optimization round, which indicates that no Unfolding or Fact Pushing

could take place. Thus, at the end of the main loop, $P \equiv_L Q$ holds. The final call of $\mathit{filter}(P, F)$ removes rules which may lead only to the inclusion of atoms $p(\mathbf{c})$ in the strong answer sets where $p \notin F$, and $p(\mathbf{c})$ can not interfere with other rules by the conditions (i)–(iii). Thus, $P \equiv_L^F Q$ holds. □

After the unfolding process, we can use Algorithm 2 for optimizing all the different kinds of queries in P. Here, inside of the subroutine *BodyOptimizer*() (Algorithm 3), we utilize $\mathit{push}(o, b)$, which takes any combination of two dl-atoms and generates an optimized (u)cq-atom according to the rewriting rules (A) and (C). Similar to *do_unfold* in Algorithm 1, $\mathit{do\_push}(o, b)$ is a generic function for checking the applicability of the rewriting rules (A) and (C), i.e., it checks for compatibility of the input lists of the atoms o and b, and decides whether pushing of o and b should be done (for instance, UNA is necessary for (C)). After that, the algorithm eliminates variables with help of Algorithm 4 in the output of dl-atoms according to Variable Elimination (B).

Theorem 8 *For every cq-program* (L, P), $P \equiv_L \mathit{RuleOptimizer}(P)$.

Proof We show now that $AS(P) = AS(\mathit{RuleOptimizer}(P))$. Since $\mathit{RuleOptimizer}(P)$ takes each $r \in P$ and tries to optimize it, we have to check that each round of the main-loop preserves the answer sets.

For each r with $B^+(r) = \emptyset$, it is clear that no pushing can be performed, hence the answer sets remain the same.

For a rule r with dl-atoms in the positive body, i.e., with an arbitrary $b \in B^+(r)$, $I \models_L B^+(r)$ iff $I \models_L \mathit{BodyOptimizer}\big(b, B^+(r) \setminus \{b\}, \emptyset, \emptyset\big)$. Since the whole optimization procedure boils down to repetitive pushing of atoms via $\mathit{push}(o, b)$, we only have to check that o and b in contrast to $\mathit{push}(o, b)$ have the same answers over an arbitrary DL-KB L. We obtain that in a rule r with o, b in $B^+(r)$, we get a rule r' by replacing o, b in r with its optimized form $\mathit{push}(o, b)$. Thus, by Theorem 4, we immediately get that $AS(P) = AS((P \setminus \{r\}) \cup \{r'\})$.

The next subroutine called is $\mathit{VarElim}(r)$, which implements Variable Elimination I and II by carefully taking each dl-atom in r into account, which has an atom $X = t$ in its CQ or in the rule body. Again, by Theorem 4, each replacement in the rules preserves the answer sets. □

Example 18 Let us reconsider the region program on the wine ontology in Example 8. Using the optimization methods for cq-programs we obtain from P an equivalent program P', where the rule r_1 in P is replaced by

$$\mathit{visit}(L) \vee \neg \mathit{visit}(L) \leftarrow \mathrm{DL}\left[\begin{array}{c} \mathit{WhiteWine}(W_1), \mathit{RedWine}(W_2), \\ \mathit{locatedIn}(W_1, L), \mathit{locatedIn}(W_2, L) \end{array}\right](W_1, W_2, L), \\ \mathit{not}\, \mathrm{DL}\big[\mathit{locatedIn}(L, L')\big](L),$$

and rule r_5 in P is replaced by

$$\mathit{delicate\_region}(W) \leftarrow \mathit{visit}(L), \mathrm{DL}\left[\begin{array}{c} \mathit{hasFlavor}(W, \mathit{delicate}), \\ \mathit{locatedIn}(W, L) \end{array}\right](W, L).$$

The dl-queries in the first rule were pushed into a single CQ. Furthermore, the rule defining *delicate* was folded into the last rule, and subsequently Query Pushing was applied to it.

Regarding the computational cost of the rewriting algorithms, Algorithm 1 runs, in general, in exponential time in the size of the program P, due to unfolding of the rules in all possible ways. Fact pushing and filtering are cheap pre- and post-processing steps, respectively, and mgus can be computed in linear time; $unfold(a, r_H, r_B)$ runs in linear time as well. Using $do\_unfold(a, r_H, r_B, P)$, we can control the unfolding operations; if we only allow unfolding of r_B from the initial program P, we get all unfolded rules in one step from P. More generally, if r_B must have been unfolded from P in constantly many steps, Algorithm 1 can be implemented to run in polynomial time.

Algorithm 2 and 4 are linear in the size of P modulo $BodyOptimizer()$. Algorithm 3 is quadratic in the size of the supplied body atoms B in the worst case (due to recursive calls on the carry atoms in C), but is linear if we always push atoms, i.e., $do\_push(o, b)$ always returns true; for small rules (size bounded by a constant), the cost is also small.

Cost Based Query Pushing The functions *do_unfold* and *do_push* in Algorithm 1 and 3 determine whether we can benefit from unfolding or query pushing. Given the input parameters, they should know whether doing the operation leads to a "better" program in terms of evaluation time, size of the program, arity of (u)cq-atoms, data transmission time, etc.

In the database area, cost estimations are based on a cost model, which usually contains information about the size of a database and its relations, an estimate of the selectivity of joins and selections, the cost of the data transfer, etc. In our setting, similar knowledge can be used to determine the cost for pushed queries.

An example for a useful strategy estimating the costs is to exploit knowledge about presence of functional roles in L. A role R is *functional*, if for all individuals x, y_1, y_2 it holds that $R(x, y_1) \wedge R(x, y_2) \rightarrow y_1 = y_2$, i.e., x is a key in R. For functional roles, query pushing is always useful since they act as a selective filter that might drastically decrease the result set.

Example 19 The fact that every person has only one mother may be stated by the functional property *hasMother*, expressed by the axiom $person \sqsubseteq \leq 1.hasMother$. The following rule retrieves all mothers of men:

$$r : a(Y) \leftarrow \mathrm{DL}[hasMother](X, Y), \mathrm{DL}[Man](X).$$

After application of Query Pushing, we obtain the rule

$$r' : a(Y) \leftarrow \mathrm{DL}[hasMother(X, Y), Man(X)](X, Y).$$

In r we get two answers with size $|hasMother| + |Man|$, while in r' we retrieve at most $|Man|$ many tuples. Pushing would be even more effective if the concept was very selective, e.g., if we had *Nobel_Laureate* instead of *Man*.

For further discussion of possible cost model strategies see [19].

7 Implementation and Experiments

In this section, we provide experimental results for the rule transformations and the performance gain obtained by applying the various optimization techniques. We have tested the rule transformations using the prototype implementation of the DL-plugin for dlvhex,[8] a logic programming engine featuring higher-order syntax and external

[8] both available at `http://www.kr.tuwien.ac.at/research/dlvhex/`

Table 3 Some test queries

region program: (Full program optimization) The unoptimized program P is in Fig. 2, the final result P' after the program optimization is:

$$\begin{aligned}
visit(L) \vee \neg visit(L) \leftarrow\ & \mathrm{DL}\left[\begin{array}{c} WhiteWine(W_1), RedWine(W_2), \\ locatedIn(W_1, L), locatedIn(W_2, L) \end{array}\right](W_1, W_2, L), \\
& not\ \mathrm{DL}\big[locatedIn(L, L')\big](L). \\
\leftarrow\ & visit(X), visit(Y), X \neq Y. \\
some_visit \leftarrow\ & visit(X). \\
\leftarrow\ & not\ some_visit. \\
delicate_region(W) \leftarrow\ & visit(L), \mathrm{DL}[hasFlavor(W, delicate), locatedIn(W, L)](W, L). \\
delicate(W) \leftarrow\ & \mathrm{DL}[hasFlavor](W, delicate).
\end{aligned}$$

VICODI program: (Fact Pushing)

$$P_v = \left\{ \begin{array}{l} c(vicodi{:}Economics), c(vicodi{:}Social), \\ v(X) \leftarrow \mathrm{DL}[hasCategory](X, Y), c(Y). \end{array} \right\}$$

$$P'_v = \left\{ \begin{array}{l} c(vicodi{:}Economics), c(vicodi{:}Social), \\ v(X) \leftarrow \mathrm{DL}\left[\left\{ \begin{array}{c} hasCategory(X, Y), \\ Y = vicodi{:}Economics \end{array} \right\} \vee \left\{ \begin{array}{c} hasCategory(X, Y), \\ Y = vicodi{:}Social \end{array} \right\} \right](X, Y). \end{array} \right\}$$

SEMINTEC query: (Query Pushing)

$$P_{s_2} = \left\{ \begin{array}{l} s_2(X, Y, Z) \leftarrow \mathrm{DL}[Man](X), \mathrm{DL}[isCreditCard](Y, X), \mathrm{DL}[Gold](Y), \\ \qquad\qquad \mathrm{DL}[livesIn](X, Z), \mathrm{DL}[Region](Z) \end{array} \right\}$$

$$P'_{s_2} = \left\{ s_2(X, Y, Z) \leftarrow \mathrm{DL}\left[\begin{array}{c} Man(X), Gold(Y), Region(Z), \\ isCreditCard(Y, X), livesIn(X, Z) \end{array} \right](X, Y, Z). \right\}$$

SEMINTEC costs: (Query Pushing, Functional Property)

$$P_l = \{ l(X, Y) \leftarrow \mathrm{DL}[hasLoan](X, Y), \mathrm{DL}[Finished](Y). \}$$

$$P'_l = \{ l(X, Y) \leftarrow \mathrm{DL}[hasLoan(X, Y), Finished(Y)](X, Y). \}$$

LUBM faculty: (Query Pushing, Inequality Pushing, Variable Elimination)

$$P_f = \left\{ \begin{array}{l} f(X, Y) \leftarrow \mathrm{DL}[Faculty](X), \mathrm{DL}[Faculty](Y), D_1 = D_2, U_1 \neq U_2, \\ \qquad\qquad \mathrm{DL}[doctoralDegreeFrom](X, U_1), \mathrm{DL}[worksFor](X, D_1), \\ \qquad\qquad \mathrm{DL}[doctoralDegreeFrom](Y, U_2), \mathrm{DL}[worksFor](Y, D_2). \end{array} \right\}$$

$$P'_f = \left\{ f(X, Y) \leftarrow \mathrm{DL}\left[\begin{array}{c} Faculty(X), Faculty(Y), U_1 \neq U_2, \\ worksFor(X, D_1), worksFor(Y, D_1), \\ doctoralDegreeFrom(X, U_1), \\ doctoralDegreeFrom(Y, U_2) \end{array} \right](X, Y, U_1, U_2, D_1). \right\}$$

atoms (see [9,38]), which uses RACER 1.9 as DL-reasoner (cf. [15]). To our knowledge, this is currently the only implemented system for such a coupling of nonmonotonic logic programs and Description Logics.

In [19], a partial equivalence between strong answer set semantics and HEX semantics has been given, which is the foundation for our prototype implementation. More specifically, every cq-program without $\uplus$ in its dl-atom input lists can be translated into

a HEX-program with the same answer set (modulo auxiliary atoms), i.e., only monotonic dl-atoms are supported.

The DL-plugin supports all forms of dl-atoms, including (U)CQs, by rewriting them to corresponding external atoms (and additional auxiliary rules) in a HEX-program. Due to the nature of RACER's (U)CQ implementation—only named individuals are under consideration—our prototype is also limited to this restricted form of (U)CQs. Regarding optimization, the DL-plugin features a software component for caching dl-queries, pushing of DL external atoms, and a minimalistic form for unfolding rules in a HEX-program.

Regarding our experiment setup, the tests were done on a P4 3GHz PC with 1GB RAM under Linux 2.6. As an ontology benchmark, we used the testsuite described in [27], which is available on the Web.[9] The testsuite includes the following four families of ontologies:

- the well-known Wine ontology [39], which describes wine types, wineries, wine-growing regions, and related information. The standard ontology is wine_0, and wine_i ($1 \leq i \leq 9$) is wine_0 with 2^i repetitions of the ABox.
- VICODI,[10] an ontology about European history. It describes relationships between historic persons, their role in history, and locations. VICODI_0 is the original ontology, and VICODI_i ($1 \leq i \leq 4$) consists of additional i copies of the ABox statements.
- SEMINTEC,[11] a financial ontology with information about loans, credit cards, and personal details like place of residence. Again, SEMINTEC_0 is the standard ontology, and the result of replicating i times the ABox is denoted as SEMINTEC_i, for $1 \leq i \leq 4$. In P_l we use the role *hasLoan*, which is an inverse functional property with $|hasLoan| = 682(n+1)$, $|Finished| = 234(n+1)$, where n is obtained from the ontology instance SEMINTEC_n.
- LUBM is short for Lehigh University Benchmark ontology,[12] which has been conceived as a tool for benchmarking DL reasoning systems. It encodes knowledge about fictional universities, their departments, students and staff. We used the LUBM Data Generator to create the Department ontologies for University 1. We then created 15 ontologies out of this setup, where each ontology LUBM_n has Department 1 up to Department n in the ABox.

A more detailed description of the used ontologies is given in [27]. For further experiments and results see [19]. The experiments included particular query rewriting rules for the test queries and a full program optimization of the region program (see Table 3 for the test programs and their rewritten counterparts with $'$). The evaluation task of the given test programs was to compute all strong answer sets of the unoptimized and the optimized programs; P_v, P_{s_2}, P_l, and P_f had one answer set, whereas our region example had several answer sets. Since all test programs are very small, the time needed to compute the program optimizations can be ignored. The results of our experiments are shown in Fig. 3. Missing entries mean memory exhaustion during evaluation.

In most of the tested programs, the performance boost using the aforementioned optimization techniques was substantial. Due to the size of the respective ontologies, in some cases the DL-engines failed to evaluate the original dl-queries, while the optimized

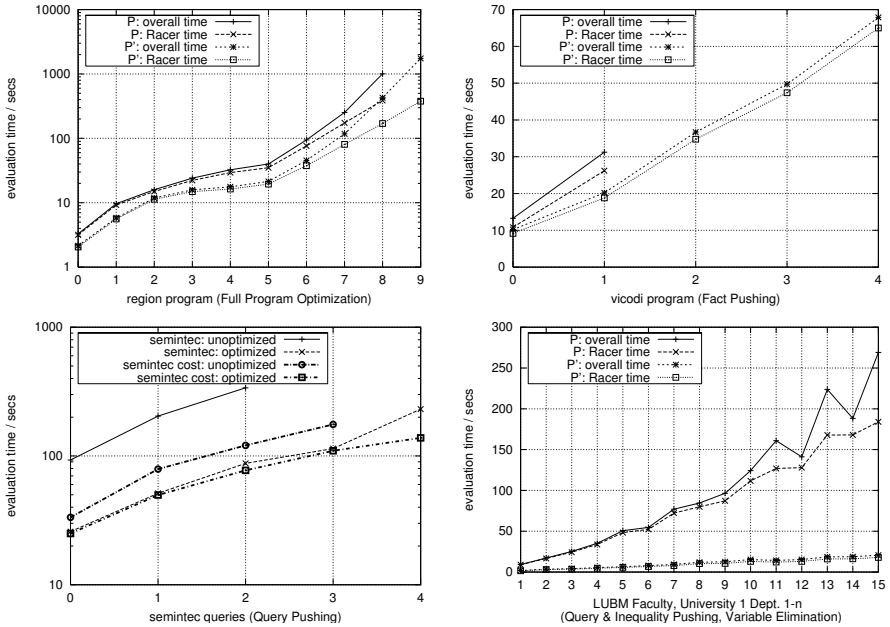
[9] http://kaon2.semanticweb.org/download/test_ontologies.zip

[10] http://www.vicodi.org/

[11] http://www.cs.put.poznan.pl/alawrynowicz/semintec.htm

[12] http://swat.cse.lehigh.edu/projects/lubm/

Fig. 3 Evaluation time for the examples.



programs did terminate with the correct result. All ontologies have been kept under UNA.

In detail, for the region program, we used the ontologies wine_0 through wine_9. As can be seen from the top-left graph in Fig. 3, there is a significant speedup, and in case of wine_9 only the optimized program could be evaluated. Most of the computation time was spent by RACER. We note that the result of the join in the first rule had only size linear in the number of top regions L; a higher performance gain may be expected for ontologies with larger joins.

The VICODI test series revealed the power of Fact Pushing (see the top-right graph in Fig. 3). While the unoptimized VICODI program (Table 3) could be evaluated only with ontologies VICODI_0 and VICODI_1, all ontologies VICODI_0 up to VICODI_4 could be handled with the optimized program.

The SEMINTEC tests dealt with Query Pushing for single rules. The rule in P_{s_2} is from one of the benchmark queries in [27], while P_l tests the performance increase when pushing a query to a functional property (see Table 3). In both cases, we performed the tests on the ontologies SEMINTEC_0 up to SEMINTEC_4. As shown in Fig. 3 (bottom-left graph) the evaluation speedup was significant. We could complete the evaluations of P_{s_2} on all SEMINTEC ontologies only with the optimization. The performance gain for P_l is in line with the constant join selectivity.

In the LUBM test setup, the test query P_f select all faculty members which work for the same department, but obtained their doctoral degree from different universities. The results in the bottom-right graph of Fig. 3 showed a drastic performance improvement.

8 Conclusion

In this paper, we have presented cq-programs, which generalize dl-programs in [10] with disjunctive rule heads (which had been cursory considered in [8]) and the possibility to pose also conjunctive queries (CQs) and unions of conjunctive queries (UCQs) against a Description Logic (DL) knowledge base. These programs are more expressive than dl-programs, as they allow to access unnamed individuals in the DL-knowledge base, and also offer higher problem solving capacity as a host language in terms of computational complexity. Furthermore, the framework can be easily adapted to other Description Logics besides the ones considered here, and has the nice feature of retaining decidability as long as answering CQs or UCQs, respectively (after possible enrichment of the DL knowledge base), is decidable.

A number of other approaches for combining rules and ontologies have been proposed; we refer to [1,5,8,33,34] for surveys and comparisons, as well as for discussions of general issues that arise with this problem. Roughly, the various approaches can be divided into three groups: (i) approaches fostering a loose coupling between rules and ontologies, in which the parts are kept separate but are connected via well-defined reasoning interfaces; (ii) approaches pursing a tight integration, in which the vocabulary of the rules and the ontology parts are kept separately but a common model-based is semantics is defined; and (iii) approaches fostering a full integration, in which a common vocabulary is used though rules and ontology axioms may be handled differently.

Rosati's well-known $\mathcal{DL}$+*log* formalism [35,34], which belongs to the second class, and the more expressive hybrid MKNF knowledge bases [25,26] and Quantified Equilibrium Logic (QEL) [6], which belong to the third group, are closest in spirit to dl- and cq-programs, as they support nonmonotonic negation and use constructions from nonmonotonic logics. However, it seems that the expressiveness of all these formalisms is different from dl- and cq-programs, as far as embeddings are concerned. It is reported in [25] that dl-programs (and hence also cq-programs) can not be captured using MKNF rules. In turn, the semantics of $\mathcal{DL}$+*log*-programs inherently involves deciding containment of CQs in UCQs, which seems to be not expressible in cq-programs in general. No detailed comparison between QEL and dl-programs is made in [6], but like for $\mathcal{DL}$+*log* and hybrid MKNF, intuitive embeddings between QEL and cq-programs are not straightforward. The reason is that cq-programs can combine hypothetical inferences under different (yet not independent) assumptions in a non-trivial way, which seems more difficult to achieve in the QEL framework. On the other hand, QEL allows for an easy extension of the language, for instance to accommodate nested expressions in which rule and ontology predicates occur at varying levels; a similar extension for cq-programs is not obvious. A detailed study of the expressive relationships between cq-programs and other formalisms remains for future work.

We remark, however, that as concerns particular reasoning tasks, cq-programs and dl-programs are as expressive as $\mathcal{DL}$+*log* and hybrid MKNF, relative to Description Logics of choice. It was reported in [35] that the satisfiability problem of $\mathcal{DL}$+*log* bases is Σ_2^p-complete for the Description Logic DL-Lite under data complexity, i.e., the knowledge base is fixed except that assertions in the DL knowledge base, which must be of form $A(\mathbf{c})$ for atomic roles and concepts A, and facts in the program part may change. In DL-Lite, answering CQs and UCQs is polynomial under data complexity; it is not difficult to establish, by adapting the arguments in Lemma 2 and Theorem 3 that dl- and cq-programs are Σ_2^p-complete under data complexity for DL-Lite (we recall

that, under data complexity, deciding the existence of an answer set for an ordinary function-free disjunctive logic program is Σ_2^p-complete, cf. [4]).

In [25, 26], the data complexity of entailment from hybrid MKNF knowledge bases has been studied for a range of DLs and syntactic fragments of the rules part. It was shown that, for DL-Lite, entailment of a ground atom (prefixed with a modal operator) is Π_2^p-complete for DL-Lite under data complexity, as well as for generic DLs in which the inference of ground atoms is in co-NP under data complexity. This can be similarly established for dl- and cq-programs, as long as the data complexity of (U)CQ answering is co-NP-complete (after possible enrichment of the DL knowledge base with negative assertions); for $\mathcal{SHIF}$ and $\mathcal{SHIQ}$, this follows from the results in [13].

Apart from increasing the expressiveness of dl-programs, we have also shown that CQs and UCQs can be fruitfully used for program optimization and rewriting. By pushing CQs to the highly optimized DL-reasoner, significant speedups can be gained, and in some cases evaluation is only feasible in that way. The results are promising and suggest that this path of optimization should be further explored. To this end, refined strategies implementing the tests *do_unfold* and *do_push* are desirable, as well as further rewriting rules. In particular, an elaborated cost model for query answering would be interesting. However, given the continuing improvements on DL-reasoners, such a model had to be revised more frequently and thus developing a particular model at this point seems less attractive.

Another interesting issue is to interface other DL-reasoners than RACER that host CQs, e.g., KAON2 or Pellet. In particular, interfacing with an engine for answering arbitrary CQs or UCQs on highly expressive DLs would be intriguing; respective algorithms are currently crafted, and prototype implementations are expected to be available in the near future. On the other hand, also an investigation of cq-programs for Description Logics with limited expressiveness, such that answering CQs and/or UCQs is tractable, or even rewritable to first-order expressions, is of interest. Under suitable syntactic restrictions, this facilitates the compilation of cq-programs to fragments of nonmonotonic logics programs that can be evaluated efficiently. Finally, a study of the expressibility of cq-programs, in terms of defining multi-valued functions as in [23], is on the agenda of future work.

Acknowledgements This work has been partially supported by the EC NoE REWERSE (IST 506779), the Austrian Science Fund (FWF) projects P17212-N04, P20840, and P20841, and the Italian Research Ministry (MUR) project Interlink II04CG8AGG. The authors would like to express their gratitude for the helpful and constructive comments from the anonymous reviewers.

References

1. Grigoris Antoniou, Carlos Viegas Damásio, Benjamin Grosof, Ian Horrocks, Michael Kifer, Jan Maluszynski, and Peter F. Patel-Schneider. Combining rules and ontologies: A survey. Tech. Rep. IST506779/Linköping/I3-D3/D/PU/a1, Linköping University, Feb. 2005.
2. Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.
3. Diego Calvanese, Thomas Eiter, and Magdalena Ortiz. Answering regular path queries in expressive description logics: An automata-theoretic approach. In *AAAI*, pages 391–396. AAAI Press, 2007.

4. Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.
5. Jos de Bruijn, Thomas Eiter, Axel Polleres, and Hans Tompits. On representational issues about combinations of classical theories with nonmonotonic rules. In *Proceedings KSEM-2006*, volume 4092 of *LNCS/LNAI*, pages 1–22. Springer, 2006.
6. Jos de Bruijn, David Pearce, Axel Polleres, and Agustín Valverde. Quantified equilibrium logic and hybrid rules. In *Proceedings First International Conference on Web Reasoning and Rule Systems (RR2007), Innsbruck, 2007*, volume 4524 of *LNCS*, pages 58–72. Springer, 2007.
7. Thomas Eiter, Giovambattista Ianni, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Combining Answer Set Programming with Description Logics for the Semantic Web. *Artificial Intelligence*, 2008. In press. doi:10.1016/j.artint.2008.04.002. Preliminary version available as Tech.Rep. INFSYS RR-1843-07-04, Institute of Information Systems, TU Vienna, January 2007.
8. Thomas Eiter, Giovambattista Ianni, Axel Polleres, Roman Schindlauer, and Hans Tompits. Reasoning with rules and ontologies. In Pedro Barahona, François Bry, Enrico Franconi, Nicola Henze, and Ulrike Sattler, editors, *Reasoning Web, Second International Summer School 2006, Lissabon, Portugal, September 25-29, 2006, Tutorial Lectures*, number 4126 in LNCS, pages 93–127. Springer, 2006.
9. Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits. A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In *Proceedings IJCAI-2005*, pages 90–96. Professional Book Center, 2005.
10. Thomas Eiter, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Combining Answer Set Programming with Description Logics for the Semantic Web. In Didier Dubois, Christopher Welty, and Mary-Anne Williams, editors, *Proceedings Ninth International Conference on Principles of Knowledge Representation and Reasoning (KR 2004), June 2-5, Whistler, British Columbia, Canada*, pages 141–151. Morgan Kaufmann, 2004.
11. Thomas Eiter, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Well-founded semantics for description logic programs in the Semantic Web. In *Proceedings RuleML-2004*, volume 3323 of *LNCS*, pages 81–97. Springer, 2004.
12. W. Faber, N. Leone, and G. Pfeifer. Recursive aggregates in disjunctive logic programs: Semantics and complexity. In *Proceedings JELIA-2004*, volume 3229 of *LNCS/LNAI*, pages 200–212. Springer, 2004.
13. B. Glimm, C. Lutz, I. Horrocks, and U. Sattler. Conjunctive query answering for the description logic SHIQ. *Journal of Artificial Intelligence Research*, 31:157–204, 2008.
14. Birte Glimm, Ian Horrocks, Carsten Lutz, and Ulrike Sattler. Conjunctive query answering for the description logic $\mathcal{SHIQ}$. In M. Veloso, editor, *Proc. 20th International Joint Conference on Artificial Intelligence* (*IJCAI 2007*), pages 399–404. AAAI Press/IJCAI, 2007.
15. V. Haarslev and R. Möller. RACER system description. In *Proceedings IJCAR-2001*, volume 2083 of *LNCS/LNAI*, pages 701–705. Springer, 2001.
16. I. Horrocks and P. F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. *Journal of Web Semantics*, 1(4):345–357, 2004.
17. I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In *Proceedings LPAR-1999*, volume 1705 of *LNCS/LNAI*, pages 161–180. Springer, 1999.
18. Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a Web ontology language. *J. Web Sem.*, 1(1):7–26, 2003.
19. Thomas Krennwallner. Integration of Conjunctive Queries over Description Logics into HEX-Programs. Master's thesis, Vienna University of Technology, Karlsplatz 13, A-1040 Wien, October 2007.
20. Thomas Lukasiewicz. A novel combination of answer set programming with description logics for the Semantic Web. In *Proceedings ESWC-2007*, volume 4519 of *LNCS*, pages 384–398. Springer, 2007.
21. Carsten Lutz. Inverse roles make conjunctive queries hard. In *Proceedings of the 2007 International Workshop on Description Logics (DL2007), Bressanone, 2007*, number 250 in CEUR Workshop Proceedings, `http://ceur-ws.org/`, pages 100–111, 2007.
22. V. Marek, I. Niemelä, and M. Truszczyński. Logic programs with monotone cardinality atoms. In *Proceedings LPNMR-2004*, volume 2923 of *LNCS*, pages 154–166, 2004.
23. W. Marek and J. Remmel. On the expressibility of stable logic programming. *Theory and Practice of Logic Programming*, 3(4-5):551–567, 2003.

24. W. Marek and M. Truszczyński. Autoepistemic Logic. *Journal of the ACM*, 38(3):588–619, 1991.
25. Boris Motik, Ian Horrocks, Riccardo Rosati, and Ulrike Sattler. Can OWL and logic programming live together happily ever after? In *Proceedings ISWC-2006*, volume 4273 of *LNCS*, pages 501–514. Springer, 2006.
26. Boris Motik and Riccardo Rosati. A faithful integration of description logics with logic programming. In M. Veloso, editor, *Proceedings 20th International Joint Conference on Artificial Intelligence* (*IJCAI 2007*), pages 477–482. AAAI Press/IJCAI, 2007.
27. Boris Motik and Ulrike Sattler. A comparison of reasoning techniques for querying large description logic aboxes. In Miki Hermann and Andrei Voronkov, editors, *LPAR*, volume 4246 of *Lecture Notes in Computer Science*, pages 227–241. Springer, 2006.
28. Boris Motik, Ulrike Sattler, and Rudi Studer. Query answering for OWL-DL with rules. *J. Web Sem.*, 3(1):41–60, 2005.
29. Andreas Nonnengart and Christoph Weidenbach. Computing Small Clause Normal Forms. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 6, pages 335–367. Elsevier Science, 2001.
30. Magdalena Ortiz de la Fuente, Diego Calvanese, and Thomas Eiter. Data Complexity of Answering Unions of Conjunctive Queries in SHIQ. In B. Parsi, U. Sattler, and D. Toman, editors, *Proceedings of the 2006 International Workshop on Description Logics (DL2006), The Lake District of the UK, May 30-June 1, 2006*, number 189 in CEUR Workshop Proceedings, pages 62–73, 2006. Online `http://CEUR-WS.org/Vol-189`.
31. Magdalena Ortiz de la Fuente, Diego Calvanese, Thomas Eiter, and Enrico Franconi. Characterizing data complexity for conjunctive query answering in expressive description logics. In *Proceedings AAAI-2006*. AAAI Press, 2006.
32. M.Magdalena Ortiz de la Fuente, Diego Calvanese, and Thomas Eiter. Data complexity of query answering in expressive description logics via tableaux. Technical Report INFSYS RR-1843-07-07, Institut für Informationssysteme, Technische Universität Wien, A-1040 Vienna, Austria, November 2007. Accepted for publication in *Journal of Automated Reasoning*.
33. J. Z. Pan, E. Franconi, S. Tessaris, G. Stamou, V. Tzouvaras, L. Serafini, I. Horrocks, and B. Glimm. Specification of coordination of rule and ontology languages. Project Deliverable D2.5.1, KnowledgeWeb NoE, June 2004.
34. Riccardo Rosati. Integrating ontologies and rules: Semantic and computational issues. In Pedro Barahona, François Bry, Enrico Franconi, Nicola Henze, and Ulrike Sattler, editors, *Reasoning Web*, volume 4126 of *LNCS*, pages 128–151. Springer, 2006.
35. Riccardo Rosati. $\mathcal{DL}$+*log*: Tight integration of description logics and disjunctive datalog. In *Proceedings 10th International Conference on Principles of Knowledge Representation and Reasoning* (*KR 2006*), pages 68–78. AAAI Press, 2006.
36. Riccardo Rosati. The limits of querying ontologies. In *Proc. ICDT 2007*, volume 4353 of *LNCS*, pages 164–178. Springer, 2007.
37. Chiaki Sakama and Hirohisa Seki. Partial deduction in disjunctive logic programming. *Journal of Logic Programming*, 32(3):229–245, 1997.
38. Roman Schindlauer. *Answer-Set Programming for the Semantic Web*. PhD thesis, Vienna University of Technology, Austria, December 2006.
39. Michael Smith, Chris Welty, and Deborah McGuinness. OWL Web Ontology Language Guide. W3C Recommendation, W3C, February 2004. Available at `http://www.w3.org/TR/owl-guide/`. The Wine ontology is available from there, or directly at `http://www.w3.org/TR/owl-guide/wine.rdf`.
40. W3C. OWL 1.1 Web ontology language overview, 2006. Available at `http://www.w3.org/Submission/owl11-overview/`.