LTCS-Report

Actions with Conjunctive Queries: Projection, Conflict Detection and Verification

Patrick Koopmann

LTCS-Report 18-08

Contents

Abstract

Description Logic actions specify adaptations of description logic interpretations based on some preconditions defined using a description logic. We consider DL actions in which preconditions can be specified using DL axioms as well as using conjunctive queries, and combinatiosn thereof. We investigate complexity bounds for the executability and the projection problem for these actions, which respectively ask whether an action can be executed on models of an interpretation, and which entailments are satisfied after an action has been executed on this model. In addition, we consider a set of new reasoning tasks concerned with conflicts and interactions that may arise if two action are executed at the same time. Since these problems have not been investigated before for Description Logic actions, we investigate the complexity of these tasks both for actions with conjunctive queries and without those. Finally, we consider the verification problem for Golog programs formulated over our family of actions. Our complexity analysis considers several expressive DLs, and we provide tight complexity bounds for those for which the exact complexity of conjunctive query entailment is known.

1 Introduction

Ontology-Mediated Data Access (OBDA) is a well-investigated technique to query data with the help of an ontology [3]. Usually, here one considers a database that is seen as a set of facts, the ABox, together with an ontology defining higher-order terms, usually defined using a description logic (DL). The ABox and the ontology together form the knowledge base (KB) over which SQL-like queries are executed. Importantly, the queries may use terms that do not explicitly occur in the data, but which are defined in the ontology, and thus query information that is implicit in the data. One application of OBDA is in situation recognition for self-adaptive systems [8]: here, a continuously changing data set describes the current state of the system, and queries are continuously evaluated to detect a situation where the process has to adapt. As an example, consider a process management system on a multi-server cluster, which assigns processes to different servers. To reduce the overall energy consumption, servers that are not running any processes are turned to hibernation. A relevant situation on such a system would be that an overloaded server is running a time critical process, while another server has computation resources available. The corresponding adaptation would then move the time critical situation to another server. The domain knowledge could be (simplified) modeled as follows by an ontology.

$$\begin{aligned}\mathsf{Server} &\sqsubseteq \forall \mathsf{runsProcess}.\mathsf{Provess}\\ \mathsf{Server} \sqcap {\geq}7.\mathsf{Process} &\sqsubseteq \mathsf{OverloadedServer}\\ \mathsf{Server} \sqcap {\leq}4.\mathsf{Process} \sqcap \neg\mathsf{Hibernated} &\sqsubseteq \mathsf{AvailableResourceServer}\end{aligned}$$

The ontology states that servers run processes, that a server is overloaded when it runs 7 or more processes, and that a server has available resources when it runs 4 or less processes. Using this ontology, the described situation can be captured by the following query q_{ov}:

$$\begin{aligned}q(x,y) \leftarrow \exists z.&\mathsf{OverloadedServer}(z) \wedge \mathsf{runsProcess}(z,x) \wedge \mathsf{TimeCriticalProcess}(x)\\ &\wedge \mathsf{AvailableResourceServer}(y).\end{aligned}$$

The query has two answer variables, x and y, and results to the query would match these respectively to a time critical process running on an overloaded server and a server with available resources. The self-adaptive system would receive a trigger whenever the query has an answer, and may then decide the move the process to the respective server.

In addition, an aim of the system is to reduce the overall energy consumption. If a server is not running any processes, it should be put into hibernate modus to safe energy. We model the additional domain knowledge using the following axioms in the ontology.

$$\begin{aligned}\mathsf{VacantServer} &\equiv \mathsf{Server} \sqcap \neg\mathsf{HibernatedServer} \sqcap {=}0\mathsf{runsProcess}.\mathsf{Process}\\ \mathsf{HibernatedServer} &\sqsubseteq {=}0\mathsf{runsProcess}.\mathsf{Process}\end{aligned}$$

Here, a vacant server is defined as one that does not run any processes, and which is not hibernated yet. In addition, the ontology specifies that a hibernated server cannot run any processes.

To query the situation when a server should be hibernated using this ontology, we would use the following query q_h

$$q(x) \leftarrow \mathsf{Vacant}(x)$$

If applied just like this, these two adaptations strategies can lead to a conflicting behvaiour: namely, it is possible that the first situation triggers the migration of a time critical process to an vacant server (which is also a server with available resources), while the second situation will trigger the system to put this server into hibernate mode. While both situations can be triggered at the same time, they cannot both be executed at the same time. Alternatively, we may assume a situation described by a query q_{h2}, which detects servers that are almost vacant, in order to migrate the remaining processes to other servers so that the server can be put into hibernate mode. While this adaptation does not conflict with the other migration strategy in the same way, it might also trigger an undesired behaviour, where the same process is migrated back and forth between two servers.

While in these examples, these conflicts can be easily recognised, in realistic, more complex systems they might be hard to detect without appropriate tool support. The main motivation for the research presented in this paper is to investigate the off-line detection of these conflicts, to guide the users of an ontology-based self-adaptive system in making sure that such conflicts can never arise.

To model the adaptation adequately, we follow the idea of DL actions as introduced in [1]. DL actions describe adaptations on models of a DL knowledge base based on some preconditions, which are defined as DL axioms. Actions can be aggregated into complex actions, which in this context are simply sequences of actions. performed. The authors of [1] consider two main reasoning problems: the *executability problem* is concerned with whether a sequence of actions can be executed on models of a given KB. The *projection problem* is to decide which entailments hold after a sequence of actions has been executed. We extend this setting in two ways: first, we consider actions in which preconditions can be formulated using *unions of conjunctive queries* (UCQs), to adhere to the setting described above. In addition, we consider additional reasoning problems that are concerned with the parallel execution of actions. 1) The *conflict problem* analyses whether, if two actions can always be executed in parallel if they can be executed on their own. 2) The *conditionalised conflict problem* attaches actions with a desired post-condition, which may also be described using queries, it analyses whether the parallel execution of two actions always ensures that these post-conditions remain satisfied. 3) Finally, we consider the *action interaction problem*, which analyses whether two actions can be triggered independent without causing non-deterministic behaviour, that is, whether exetuting two actions in parallel on the same model always results in the same result.

A more expressive framework for reasoning about DL actions is to embed them into *Golog programs* [2]. Golog is a simple programming language providing for common constructs such as loops and conditional execution, which can be used to describe the behaviour of more complex systems. For Golog programs, one is usually concerned with the *verification problem*, which is to verify whether executions of the program satisfy certain properties formulated in a temporal logic. Tasks 1) and 2) can be easily encoded into Golog programs with DL actions, provided they are expressive enough to use UCQs. Moreover, they allow to model the behaviour of self-adaptive problems more adequately due to their additional expressive power. However, usually, the verification problem is one exponential harder than the executability or the projection problem [14, Theorem, 4.24]. We show that this is not necessarily the case if actions can be equipped with UCQs: in fact, then all problems are not harder than query answering itself, at least for the DLs we consider in this paper, and for which the precise complexity of query answering is known. In contrast, if DL actions may only use DL axioms as preconditions, then the problems are not computationally harder than the executability and the projection problem.

2 Ontology-Mediated Query Answering

We recall the DLs discussed in the paper and ontology-mediated query answering for knowledge bases expressed in these.

Let $\mathsf{N_C}$, $\mathsf{N_R}$ and $\mathsf{N_I}$ be three pair-wise disjoint sets of respectively *concept names*, *role names* and *individual names*. A *role* R is either a role name $r \in \mathsf{N_R}$ or an *inverse role* r^-, $r \in \mathsf{N_R}$. Concepts are expressions of the following forms, where $A \in \mathsf{N_C}$, R is a role, $a \in \mathsf{N_I}$ an individual name, $n \in \mathbb{N} \setminus \{0\}$ and C, C_1 and C_2 are concepts:

- top: $\top$,
- conjunction: $C_1 \sqcap C_2$,
- complement: $\neg C$,
- existential role restriction: $\exists R.C$,
- nominal: $\{a\}$, and
- at-least restriction: $\geq nR.C$.

Further concepts are introduced as abbreviations: bottom $\bot = \neg\top$, disjunction $C_1 \sqcup C_2 = \neg(\neg C_1 \sqcap C_2)$, universal role restriction $\forall R.C = \neg(\exists R.\neg C)$, at-least restriction $\leq nR.C = \neg(\geq (n+1)R.C)$ and equals restriction $=nR.C = (\geq iR.C \sqcap \leq nR.C)$.

A concept is in the DL $\mathcal{ALC}$ iff it only uses top, conjunctions, complements and existential role restrictions and no inverse roles. More expressive DLs are denoted by appending $\mathcal{ALC}$ with one or more of the letters $\mathcal{I}$, $\mathcal{O}$ and $\mathcal{Q}$, which respectively denote that we can additionally use inverse roles, nominals and at-least restrictions. For example, $\mathcal{ALCIO}$ concepts may use top, conjunctions, complements, existential role restrictions, inverse roles and nominals, while $\mathcal{ALCQ}$ may use top, conjunctions, complements, existential role restrictions and at-least restrictions.

A *knowledge base* (KB) $\mathcal{K}$ is a tuple $\langle \mathcal{T}, \mathcal{A} \rangle$, where the TBox $\mathcal{T}$ contains *axioms* of the forms $C_1 \sqsubseteq C_2$ (general concept inclusions, GCIs) and $R_1 \sqsubseteq R_2$ (role inclusions), and the ABox $\mathcal{A}$ contains *assertions* of the forms $C_1(a)$ and $r(a,b)$, where C_1, C_2 are concepts, R_1, R_2 roles, $a, b \in \mathsf{N_I}$ and $r \in \mathsf{N_R}$. In the DLs introduced above, role inclusions are not permitted, and we use the letter $\mathcal{H}$ to denote that role inclusions are permitted: for example, a KB in $\mathcal{ALCIO}$ does not have role inclusions, while a KB in $\mathcal{ALCHIO}$ may additionally have role inclusions.

The semantics of KBs is defined in terms of *interpretations*, which are tuples $\mathcal{I} = \langle \Delta^\mathcal{I}, \cdot^\mathcal{I} \rangle$ with the *domain* $\Delta^\mathcal{I}$ a set of *domain elements* and the *interpretation* $\cdot^\mathcal{I}$ a function mapping each individual name $a \in \mathsf{N_I}$ to a domain element $a^\mathcal{I} \in \Delta^\mathcal{I}$, each concept name $A \in \mathsf{N_C}$ to a set $A^\mathcal{I} \subseteq \Delta^\mathcal{I}$ of domain elements, and each role name $r \in \mathsf{N_R}$ to a relation $r^\mathcal{I} = \Delta^\mathcal{I} \times \Delta^\mathcal{I}$ over the domain. The interpretation function is extended to inverse roles by $(r^-)^\mathcal{I} = (r^\mathcal{I})^-$, and to concepts by

$$\top^\mathcal{I} = \Delta^\mathcal{I} \qquad (C_1 \sqcap C_2)^\mathcal{I} = C_1^\mathcal{I} \cap C_2^\mathcal{I} \qquad (\neg C_1)^\mathcal{I} = \Delta^\mathcal{I} \setminus C_1^\mathcal{I}$$
$$(\exists R.C)^\mathcal{I} = \{d \in \Delta^\mathcal{I} \mid \exists \langle d, e\rangle \in R^\mathcal{I} : e \in C^\mathcal{I}\} \qquad \{a\}^\mathcal{I} = \{a^\mathcal{I}\}$$
$$(\geq nR.C)^\mathcal{I} = \{d \in \Delta^\mathcal{I} \mid \#\{\langle d, e\rangle \in R^\mathcal{I} : e \in C^\mathcal{I}\} \geq n\}.$$

An axiom $C_1 \sqsubseteq C_2$ $(R_1 \sqsubseteq R_2)$ is *satisfied* in an interpretation $\mathcal{I}$, in symbols $\mathcal{I} \models C_1 \sqsubseteq C_2$ $(\mathcal{I} \models R_1 \sqsubseteq R_2)$, iff $C_1^\mathcal{I} \subseteq C_2^\mathcal{I}$ $(R_1^\mathcal{I} \subseteq R_2^\mathcal{I})$. An assertion $C_1(a)$ $(r(a, b))$ is satisfied in an interpretation $\mathcal{I}$, in symbols $\mathcal{I} \models C_1(a)$ $(\mathcal{I} \models r(a, b))$, iff $a^\mathcal{I} \in C^\mathcal{I}$ $(\langle a, b\rangle \in r^\mathcal{I}$. An interpretation $\mathcal{I}$ is a *model* of a KB $\mathcal{K}$, in symbols $\mathcal{I} \models \mathcal{K}$, if every axiom and assertion in $\mathcal{K}$ is satisfied in $\mathcal{I}$. $\mathcal{K}$ is *satisfiable* iff it has a model, and otherwise *unsatisfiable*. Finally, a KB $\mathcal{K}$ entails an axiom/assertion α, iff $\mathcal{I} \models \alpha$ for all models $\mathcal{I}$ of $\mathcal{K}$.

Note that we do not enforce the uniquer name assumption (UNA), since inequality of individuals a,b can be easily expressed by adding the axiom $\{a\} \sqsubseteq \neg\{b\}$ to the KB. The UNA can thus be enforced for any KB using only a polynomial number of axioms.

We define satisfaction of first-order formulae from interpretations in the usual way, and say a first-order formula is entailed by a KB iff it is entailed by every model of it. A *Boolean conjunctive query* (CQ) is a first-order formula of the form $\exists\vec{x}.\phi$, where ϕ is a conjunction over atoms of the forms $A(t_1)$, $r(t_1, t_2)$, with $A \in \mathsf{N_C}$, $r \in \mathsf{N_R}$ and $t_1, t_2 \in \mathsf{N_I} \cup \vec{x}$. A *Boolean union of conjunctive queries* (UCQ) is a disjunction over Boolean conjunctive queries.

We have the following complexities for deciding whether a KB $\mathcal{K}$ is satisfiable, depending on the DL $\mathcal{L}$ in which the $\mathcal{K}$ is formulated:

1. it is ExpTime-hard for $\mathcal{L} = \mathcal{ALC}$ [12],
2. it is in ExpTime for $\mathcal{L} \in \{\mathcal{ALCHIO}, \mathcal{ALCHIQ}, \mathcal{ALCHOQ}\}$ [13], and
3. it is NExpTime-complete for $\mathcal{L} = \mathcal{ALCHIOQ}$ [13].

Deciding entailment of CQs and UCQs is

1. ExpTime-complete for $\mathcal{L} = \mathcal{ALCHQ}$ [9],
2. 2-ExpTime-hard for $\mathcal{L} \in \{\mathcal{ALCI}, \mathcal{ALCO}\}$ [9,10],
3. in 2-ExpTime for $\mathcal{L} \in \{\mathcal{ALCHOQ}, \mathcal{ALCHIQ}, \mathcal{ALCHIO}\}$ [4,6,7], and
4. decidable for $\mathcal{L} =\in \{\mathcal{ALCIOQ}, \mathcal{ALCHIOQ}\}$ [11].

3 Actions with Conjunctive Queries

We introduce our extended action framework, by defining DL actions with CQs. We distinguish between *atomic actions* and *composite actions*. A *basic* action consists of a precondition, which states when an action can be executed (in our context, this corresponds to a situation description), and a post-condition that descibes the effects of the action. Composite actions are then formed by combining atomic actions.

Preconditions are formulated using DL formulae, which in our case may involve conjunctive queries. The set of *DL formulae* is the smallest set of formulae satisfying the following conditions:

1. every axiom, assertion and CQ is a DL formula,
2. if ϕ is a DL formula, then so is $\neg\phi$, and
3. if ϕ_1 and ϕ_2 are DL formulae, then so is $\phi_1 \wedge \phi_2$.

We introduce disjunction as abbreviation: $\phi_1 \vee \phi_2 = \neg(\neg\phi_1 \wedge \neg\phi_2)$. For convenience, we identify KBs $\mathcal{K} = \{\mathcal{T}, \mathcal{A}\}$ with the corresponding DL formula $\bigwedge_{\alpha\in\mathcal{T}} \alpha \wedge \bigwedge_{\alpha\in\mathcal{A}} \alpha$. Note that, even though we did not use UCQs directly, they can be expressed in a DL formula using disjunction.

The semantics of DL formulae is defined intuitively, that is, given an interpretation $\mathcal{I}$ and a DL formula ϕ that is not an axiom, assertion or CQ, we say that ϕ is satisfied in $\mathcal{I}$, in symbols $\mathcal{I} \models \phi$, if

1. If ϕ is an axiom or a CQ, satisfaction is defined as in Section 2
2. $\phi = \neg\psi$ and $\mathcal{I} \not\models \psi$ or
3. $\phi = \psi_1 \wedge \psi_2$, $\mathcal{I} \models \psi_1$ and $\mathcal{I} \models \psi_2$.

We define actions as operations on interpretations. Specifically, an atomic action is of the form $\mathfrak{a} = \phi \rhd \mathfrak{E}$, with the *precondition* ϕ a DL formula, and the *post-condition* $\mathfrak{e}$, which is a set of *conditionalised effects* and *unconditionalised effects*. Here, unconditionalised effects are of the forms $\oplus\alpha$ and $\ominus\alpha$, with α an assertion of one of the forms $A(a)$, $r(a, b)$ with $A \in \mathsf{N_C}$, $r \in \mathsf{N_R}$ and $a, b \in \mathsf{N_I}$, and conditionalised effects are of the forms $\beta/\mathfrak{e}$, where β is an assertion and $\mathfrak{e}$ an unconditionalised effect. Intuitively, the precondition specifies when an action can be executed. The exection that applies all post-conditions at the same time, where each conditionalised effect $\beta/\oplus\alpha$ $(\beta/\ominus\alpha)$ is applied whenever β is satisfied.

A *composite action* is an expression of the form $\mathfrak{a}_1; \mathfrak{a}_2; \ldots \mathfrak{a}_n$, with $\mathfrak{a}_1, \ldots, \mathfrak{a}_n$ atomic actions. We also allow the *empty composite action* ϵ, which corresponds to an empty sequence of actions. Note that an atomic action can also be seen as a composite action with only one element. For this reason, we may for simplicity treat it as a composite action if this is convenient.

We define the semantics of actions. For this, we first collect the changes on each concept and role name for a set of effects. Specifically, for a set $\mathfrak{E}$ of effects and an interpretation $\mathcal{I}$, we first define for all $A \in \mathsf{N_C}$ and $r \in \mathsf{N_R}$:

$$\begin{aligned}
A^{+\mathfrak{E}} &= \{a^{\mathcal{I}} \mid \oplus(A(a)) \in \mathfrak{E}\} \cup \{a^{\mathcal{I}} \mid \beta/\oplus(A(a)), \mathcal{I} \models \beta\} \\
A^{-\mathfrak{E}} &= \{a^{\mathcal{I}} \mid \ominus(A(a)) \in \mathfrak{E}\} \cup \{a^{\mathcal{I}} \mid \beta/\mathit{Drop}(A(a)), \mathcal{I} \models \beta\} \\
r^{+\mathfrak{E}} &= \{\langle a^{\mathcal{I}}, b^{\mathcal{I}}\rangle \mid \oplus(r(a, b)) \in \mathfrak{E}\} \cup \{\langle a^{\mathcal{I}}, b^{\mathcal{I}}\rangle \mid \beta/\oplus(r(a, b)), \mathcal{I} \models \beta\} \\
r^{-\mathfrak{E}} &= \{\langle a^{\mathcal{I}}, b^{\mathcal{I}}\rangle \mid \ominus(r(a, b)) \in \mathfrak{E}\} \cup \{\langle a^{\mathcal{I}}, b^{\mathcal{I}}\rangle \mid \beta/\ominus(r(a, b)), \mathcal{I} \models \beta\}.
\end{aligned}$$

We say that an atomic action $\mathfrak{a} = \phi \rhd \mathfrak{E}$ is *executable* on an interpretation $\mathcal{I}$ if $\mathcal{I} \models \phi$.

The *execution of an atomic action* $\mathfrak{a}$ *on* $\mathcal{I}$ is defined as $\mathcal{I}^{\mathfrak{a}} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}^{\mathfrak{a}}}\rangle$, where for every $a \in \mathsf{N_I}$, $a^{\mathcal{I}^{\mathfrak{a}}} = a^{\mathcal{I}}$, and for every $X \in \mathsf{N_C} \cup \mathsf{N_R}$, $X^{\mathcal{I}^{\mathfrak{a}}} = (X^{\mathcal{I}} \cup X^{+\mathfrak{E}}) \setminus X^{-\mathfrak{E}}$. Note that we for convenience define executions independent on executability.

Executability and execution of composite actions is now defined as expected. Specifically, for a composite action $\mathfrak{a} = \mathfrak{a}_1; \ldots, \mathfrak{a}_n$, the execution $\mathcal{I}^{\mathfrak{a}}$ of $\mathfrak{a}$ on $\mathcal{I}$ is defined recursively as $\mathcal{I}^{\epsilon} = \mathcal{I}$ and $\mathcal{I}^{\mathfrak{a}_n} = (\mathcal{I}^{\mathfrak{a}_1;\ldots{}'\mathfrak{a}_{n-1}})^{\mathfrak{a}}_n$. $\mathfrak{a}$ is executable if $\mathfrak{a} = \epsilon$ or $\mathfrak{a} = \mathfrak{a}_1; \ldots; \mathfrak{a}_n$ and $\mathfrak{a}_n$ is executable on $\mathcal{I}^{\mathfrak{a}_1;\ldots;\mathfrak{a}_{n-1}}$.

In addition to an action, one usually considers a KB that restricts the space of interpretations, or alternatively, specifies the (incomplete knowledge) about the initial situation on which the action is executed. The obvious questions in this setting are 1) is the given action executable on the models of the KB, and 2) what is entailed after the action has been executed. These are the first two reasoning problems we consider.

Definition 1. Given a KB $\mathcal{K}$ and an action $\mathfrak{a}$, the *executability problem* is to decide whether ϕ is executable on every interpretation of $\mathcal{K}$. The *projection problem* is to decide whether, given a *projection problem* is to decide whether, given a DL formula, for every model $\mathcal{I}$ of $\mathcal{K}$, we have $\mathcal{I}^{\mathfrak{a}} \models \phi$. We then say that ϕ is an *answer* to the projection problem on $\mathcal{K}$ and $\mathfrak{a}$.

The executability problem is easily reducable to the projection problem, as for a composite action $\mathfrak{a}_1; \ldots; \mathfrak{a}_n$, we just need to check whether for each $i \in [\![1, n]\!]$, where $\mathfrak{a}_i = \phi_i \rhd \mathfrak{E}_i$, whether ϕ_i is an answer to the projection problem for $\mathcal{K}$ and $\mathfrak{a}_1; \ldots; \mathfrak{a}_{i-1}$.

3.1 Conflicting and interacting actions

To define when two actions are conflicting, we need to specify formally what it means for two actions to be applied concurrently. Let $\mathfrak{a}_1$ and $\mathfrak{a}_2$ be two actions. The set $\mathrm{Seq}(\mathfrak{a}_1 \| \mathfrak{a}_2)$ is defined by induction over $\mathfrak{a}_1$ and $\mathfrak{a}_2$:

- if $\mathfrak{a}_1 = \epsilon$, then $\mathrm{Seq}(\mathfrak{a}_1 \| \mathfrak{a}_2) = \{\mathfrak{a}_2\}$,
- if $\mathfrak{a}_2 = \epsilon$, then $\mathrm{Seq}(\mathfrak{a}_1 \| \mathfrak{a}_2) = \{\mathfrak{a}_1\}$,
- if $\mathfrak{a}_1 = \mathfrak{a}_1^h; \mathfrak{a}_1^t$ and $\mathfrak{a}_2 = \mathfrak{a}_2^h; \mathfrak{a}_2^t$, with $\mathfrak{a}_1^h$ and $\mathfrak{a}_2^h$ atomic actions, and $\mathfrak{a}_1^t$ and $\mathfrak{a}_2^t$ possibly empty composite actions, then
$$\mathrm{Seq}(\mathfrak{a}_1 \| \mathfrak{a}_2) = \{\mathfrak{a}_1^h; \mathfrak{a} \mid \mathfrak{a} \in \mathrm{Seq}(\mathfrak{a}_1^t \| \mathfrak{a}_2)\} \cup \{\mathfrak{a}_2^h; \mathfrak{a} \mid \mathfrak{a} \in \mathrm{Seq}(\mathfrak{a}_1 \| \mathfrak{a}_2^t)\}.$$

Intuitively, $\mathrm{Seq}(\mathfrak{a}_1 \| \mathfrak{a}_2)$ contains all possible execution sequences that can be obtained by interleaving $\mathfrak{a}_1$ and $\mathfrak{a}_2$. We note that we here take into account the intuition that all effects of an action are executed instantly and simultaneously. To obtain a more fine-grained analysis, one might want to split the atomic actions into several smaller actions. For example, for an atomic action $\phi \rhd \mathfrak{E}$ and a (possibly conditionalised) effect $\mathfrak{e} \in \mathfrak{E}$, the action can be split into a composite action $\phi \rhd (\mathfrak{E} \setminus \mathfrak{e}); \top \rhd \{\mathfrak{e}\}$.

As motivated in the introduction, our first interest is in deciding whether two actions are conflicting. This reasoning problem is defined as follows.

Definition 2. Let $\mathcal{K}$ be a KB, $\mathfrak{a}_1$ and $\mathfrak{a}_2$ be two actions and ϕ a DL formula. We say $\mathfrak{a}_1$ and $\mathfrak{a}_2$ are *conflicting in* $\mathcal{K}$ if there exists a model $\mathcal{I}$ of $\mathcal{K}$ on which $\mathfrak{a}$ and $\mathfrak{a}_2$ are executable, but also some action $\mathfrak{a} \in \mathrm{Seq}(\mathfrak{a}_1 \| \mathfrak{a}_2)$ that is not executable on $\mathcal{I}$. We say $\mathfrak{a}_1$ and $\mathfrak{a}_2$ are *conflicting in* $\mathcal{K}$ *with respect to* ϕ if there exists a model $\mathcal{I}$ of $\mathcal{K}$ on which $\mathcal{I}_1^{\mathfrak{a}} \models \phi$, $\mathcal{I}_2^{\mathfrak{a}} \models \phi$, but for some $\mathfrak{a} \in \mathrm{Seq}(\mathfrak{a}_1 \| \mathfrak{a}_2)$, $\mathcal{I}^{\mathfrak{a}} \not\models \phi$. The *conflict problem* is two decide whether two actions are conflicting in a given KB. The *conditionalised conflict problem* is to decide whether two actions are conflicting in a given KB with respect to a given DL formula.

Sometimes, we are interested not only in knowing whether to actions conflict with respect to some given invariant, but in whether they lead to some non-deterministic behavior. That is, we want to be sure that their outcome is irrelevant of the specific sequence picked from $\mathrm{Seq}(\mathfrak{a}_1 \| \mathfrak{a}_2)$. This is captured in the following definition.

Definition 3. Given a KB $\mathcal{K}$ and two actions $\mathfrak{a}_1$ and $\mathfrak{a}_2$, we say that $\mathfrak{a}_1$ and $\mathfrak{a}_2$ are *interacting in* $\mathcal{K}$ if there exist a model $\mathcal{I}$ of $\mathcal{K}$ and two actions $\mathfrak{a}, \mathfrak{a}' \in \mathrm{Seq}(\mathfrak{a}_1, \mathfrak{a}_2)$ s.t. $\mathcal{I}^{\mathfrak{a}} \neq \mathcal{I}^{\mathfrak{a}'}$. The *action interaction problem* is to decide whether two actions are interacting.

3.2 Golog programs over DL actions with conjunctive queries

As it turns out, most reasoning problems discussed in the previous subsections are instances of the verification problem of Golog programs. Moreover, Golog allows for a more detailed modelling of complex self-adaptive systems. However, Golog programs with DL actions with CQs has not been investigated before. As we later show, for these Golog programs, the verification problem is often not harder than the standard query entailment problem.

The set of Golog programs over DL actions with CQs is the smallest set satisfying the following:

1. every atomic action is a Golog program,
2. every empty composite action is a Golog program,
3. if $\mathfrak{p}_1$, $\mathfrak{p}_2$ are Golog program and ϕ a DL formula, then the following expressions are Golog programs:
   (a) $\mathfrak{p}_1; \mathfrak{p}_2$ (sequence),
   (b) $(\mathfrak{p}_1|\mathfrak{p}_2)$ (non-deterministic choice),
   (c) $(\mathfrak{p}_1\|\mathfrak{p}_2)$ (concurrent execution),
   (d) $\phi?$ (test),
   (e) $\mathfrak{p}*$ (non-deterministic iteration).

Note that, even though we do not use composite actions in this definition, they can still be expressed using the sequence-operator, which has the same syntax, but is also used with the same semantics, which we will see shortly. To express deliberate failing, we use the expression Fail as an abbreviation for a failing test: $\mathsf{Fail} = \neg\exists x.\top(x)$. We can express common programming constructs easily in Golog: if-then-else using $(\phi?; \mathfrak{p}_1)|(\neg\phi?; \mathfrak{p}_2)$, and while loops using $\mathfrak{p}*; \phi?$.

The semantics of Golog programs is defined using transition systems. Every Golog program induces an transition system over interpretations, which represents all possible executions of the program from an interpretation.

To identify the states of the program, we denote by $\text{sub}(\mathfrak{p})$ the set of *sub-programs of* $\mathfrak{p}$, which is defined as the smallest set satisfying

- $\mathfrak{p} \in \text{sub}(\mathfrak{p})$,
- if $\mathfrak{p}'* \in \text{sub}(\mathfrak{p})$, then also $\mathfrak{p}'; \mathfrak{p}'* \in \text{sub}(\mathfrak{p})$,
- if $(\mathfrak{p}_1; \mathfrak{p}_2) \in \text{sub}(\mathfrak{p})$, $(\mathfrak{p}_1|\mathfrak{p}_2) \in \text{sub}(\mathfrak{p})$ or $(\mathfrak{p}_1\|\mathfrak{p}_2) \in \text{sub}(\mathfrak{p})$, then $\mathfrak{p}_1, \mathfrak{p}_2 \in \text{sub}(\mathfrak{p})$, and
- if $(\mathfrak{p}_1; \mathfrak{p}_2)/(\mathfrak{p}_1|\mathfrak{p}_2)/(\mathfrak{p}_1\|\mathfrak{p}_2) \in \text{sub}(\mathfrak{p})$, then $(\mathfrak{p}_1'; \mathfrak{p}_2')/(\mathfrak{p}_1'|\mathfrak{p}_2')/(\mathfrak{p}_1'\|\mathfrak{p}_2') \in \text{sub}(\mathfrak{p})$ for every $\mathfrak{p}_1' \in \text{sub}(\mathfrak{p}_1)$ and $\mathfrak{p}_2' \in \text{sub}(\mathfrak{p}_2)$.

It is not hard to see that $\text{sub}(\mathfrak{p})$ is exponentially bounded in the size of $\mathfrak{p}$.

Let $\mathfrak{p}$ a Golog program. Then *the transition system induced by* $\mathfrak{p}$ is the tuple $\mathrm{T} = \langle Q, I, \hookrightarrow, Q_{\text{final}}, \lambda\rangle$, where

1. the set Q of states contains every tuple $\langle \mathcal{I}, \mathfrak{p}' \rangle$, where $\mathcal{I}$ is an interpretation and $\mathfrak{p}' \in \text{sub}(\mathfrak{p})$,
2. the set $I \subseteq Q$ of initial states contains all states of the form $\langle \mathcal{I}, \mathfrak{p} \rangle$,
3. the labeling function λ maps each state $\langle \mathcal{I}, \mathfrak{p} \rangle$ to the set of DL formulae ϕ s.t. $\mathcal{I} \models \phi$,

4. the transition relation $\hookrightarrow$ and the set Q_{final} of accepting states are defined next.

Before we define the transition relation $\hookrightarrow$, it is convenient to define the set of states where a program may terminate. Specifically, Q_{final} is the smallest set of states that contains every state $\langle \mathcal{I}, \mathfrak{p}' \rangle \in Q$ s.t.

1. $\mathfrak{p}' = \epsilon$,
2. $\mathfrak{p}' = \phi?$ and $\mathcal{I} \models \phi$,
3. $\mathfrak{p}' = (\mathfrak{p}_1)*$,
4. $\mathfrak{p}' = (\mathfrak{p}_1; \mathfrak{p}_2)$, $\mathfrak{p}_1 \in Q_{\text{final}}$ and $\mathfrak{p}_2 \in Q_{\text{final}}$,
5. $\mathfrak{p}' = (\mathfrak{p}_1 | \mathfrak{p}_2)$ and $\mathfrak{p}_1 \in Q_{\text{final}}$ or $\mathfrak{p}_2 \in Q_{\text{final}}$, and
6. $\mathfrak{p}' = (\mathfrak{p}_1 \| \mathfrak{p}_2)$ and $\mathfrak{p}_1 \in Q_{\text{final}}$ and $\mathfrak{p}_2 \in Q_{\text{final}}$.

Now the relation $\hookrightarrow$ in the transition system $\mathrm{T} = \langle Q, I, \hookrightarrow, Q_{\text{final}} \rangle$ is defined as the smallest relation s.t. for every $q \in Q$:

1. if $q = \langle \mathcal{I}, \mathfrak{a} \rangle$, where $\mathfrak{a}$ is an atomic action that is executable on $\mathcal{I}$, then $q \hookrightarrow \langle \mathcal{I}^{\mathfrak{a}}, \epsilon \rangle$,
2. if $q = \langle \mathcal{I}, \mathfrak{p}_1; \mathfrak{p}_2 \rangle$ and $\langle \mathcal{I}, \mathfrak{p}_1 \rangle \hookrightarrow \langle \mathcal{I}', \mathfrak{p}_1' \rangle$, then $q \hookrightarrow \langle \mathcal{I}', \mathfrak{p}_1'; \mathfrak{p}_2 \rangle$,
3. if $q = \langle \mathcal{I}, \mathfrak{p}_1; \mathfrak{p}_2 \rangle$, $\langle \mathfrak{p}_1, \mathcal{I} \rangle \in Q_{\text{final}}$ and $\langle \mathcal{I}, \mathfrak{p}_2 \rangle \hookrightarrow \langle \mathcal{I}', \mathfrak{p}_2' \rangle$, then $q \hookrightarrow \langle \mathcal{I}', \mathfrak{p}_2' \rangle$,
4. if $q = \langle \mathcal{I}, (\mathfrak{p}_1 | \mathfrak{p}_2) \rangle$, and $\langle \mathcal{I}, \mathfrak{p}_1 \rangle \hookrightarrow \langle \mathcal{I}', \mathfrak{p}_3 \rangle$ or $\langle \mathcal{I}, \mathfrak{p}_2 \rangle \hookrightarrow \langle \mathcal{I}', \mathfrak{p}_3 \rangle$, then $q \hookrightarrow \langle \mathcal{I}', \mathfrak{p}_3 \rangle$,
5. if $q = \langle \mathcal{I}, (\mathfrak{p}_1 \| \mathfrak{p}_2) \rangle$ and $\langle \mathcal{I}, \mathfrak{p}_1 \rangle \hookrightarrow \langle \mathcal{I}', \mathfrak{p}_1' \rangle$, then $q \hookrightarrow \langle \mathcal{I}', (\mathfrak{p}_1' \| \mathfrak{p}_2) \rangle$
6. if $q = \langle \mathcal{I}, (\mathfrak{p}_1 \| \mathfrak{p}_2) \rangle$ and $\langle \mathcal{I}, \mathfrak{p}_2 \rangle \hookrightarrow \langle \mathcal{I}', \mathfrak{p}_2' \rangle$, then $q \hookrightarrow \langle \mathcal{I}', (\mathfrak{p}_1 \| \mathfrak{p}_2') \rangle$
7. if $q = \langle \mathcal{I}, \mathfrak{p}_1 * \rangle$ and $\langle \mathcal{I}, \mathfrak{p}_1 \rangle \hookrightarrow \langle \mathcal{I}', \mathfrak{p}_2 \rangle$, then $q \hookrightarrow \langle \mathcal{I}', \mathfrak{p}_2; \mathfrak{p}_1 * \rangle$,

The definition, which might seem a bit cumbersome at first sight, ensures that in every step in the transition system consumes an action, while treating all operators as expected. The last condition makes sure that accepting paths in the transition system can always be followed, with the intuitive behavior that the state remains the same if the program finished its execution.

3.3 DL-CTL* formulae with conjunctive queries

Properties of Golog progams are specified using CTL*-formulae over DL axioms, assertions and CQs, similarly to how we defined DL formulae. While there are popular fragments of CTL, such as CTLand LTL, in which properties could be expressed, we only consider the most expressive logic here. As our complexity bounds later show, the complexity is not reduced if we restrict the logic. *DL-CTL* state formulae* Φ and *DL-CTL* path formulae* Ψ are built according to the following syntax rules, where ϕ is an axiom, assertion or CQ:

$$\begin{aligned}\Phi ::=& \phi \mid \neg\Phi \mid \Phi \wedge \Phi \mid \mathsf{E}\Psi\\ \Psi ::=& \Phi \mid \Psi \wedge \Psi \mid \mathsf{X}\Psi \mid \Psi \mathsf{U} \Psi.\end{aligned}$$

As usual, further operators are defined as abbreviations. For state formulae, we abbreviate $\top = \exists x.\top(x)$, $\Phi_1 \vee \Phi_2 = \neg(\neg\Phi_1 \wedge \neg\Phi_2)$ and $\mathsf{A}\Psi = \neg(\mathsf{E}\neg\Psi)$, and for path formulae, we abbreviate $\Psi_1 \vee \Psi_2 = \neg(\neg\Psi_1 \wedge \neg\Psi_2)$, $\Diamond\Psi = \top\mathsf{U}\Psi$ and $\Box\Psi = \neg\Diamond\neg\Psi$.

Entailment of CTL*-formulae from Golog programs is then defined based on the induced transition systems. An path in a transition system $\mathrm{T} = \langle Q, I, \hookrightarrow, Q_{\text{final}}, \lambda\rangle$ is a possibly unbounded sequence $q_1, \ldots$ of states s.t. for $i \geq 1$, $q_i \hookrightarrow q_{i+1}$ or q_i is the last state in the path. For a path $\pi = q_1, \ldots$, we denote by $\pi[i] = q_i$ the ith state in π, and by $\pi[i..] = q_i, \ldots$ the prefix of π beginning from the ith state. We first define entailment of CTL* formulae with respect to a state/path in the transition system. Let $\mathrm{T} = \langle Q, I, \hookrightarrow, Q_{\text{final}}, \lambda\rangle$ be a labelled transition system. A path is *accepting* iff it is unbounded or we have $\pi[n] \in Q_{\text{final}}$ for n the length of π. Entailment of a CTL* state formula Φ in T a state $q \in Q$, in symbols $T, q \models \Phi$, is then defined as follows based on the syntactical shape of Φ:

1. if $\Phi \in \lambda(q)$, then $T, q \models \Phi$,
2. if $\Phi = \neg\Psi$, then $T, q \models \Phi$ iff $T, q \not\models \neg\Phi$,
3. if $\Phi = \Phi_1 \wedge \Phi_2$, then $T, q \models \Phi$ iff $T, q \models \Phi_1$ and $T, q \models \Phi_2$, and
4. if $\Phi = \mathsf{E}\Psi$, then $T, q \models \Phi$ iff there exists an accepting path π in T s.t. $\pi[1] = q$ and $T, \pi \models \Psi$,

where entailment of a path formula Ψ from a path π in T, in symbols $T, \pi \models \Psi$, is defined as follows:

1. if Ψ is a state formula, then $T, \pi \models \Psi$ iff $T, \pi[1] \models \Psi$,
2. if $\Psi = \neg\Psi'$, then $T, \pi \models \Psi$ iff $T, \pi \not\models \Psi'$,
3. if $\Psi = \Psi_1 \wedge \Psi_2$, then $T, \pi \models \Psi$ iff $T, \pi \models \Psi_1$ and $T, \pi \models \Psi_2$,
4. if $\Psi = \mathsf{X}\Psi'$, then $T, \pi \models \Psi$ iff $T, \pi[2..] \models \Psi'$, and
5. if $\Psi = \Psi_1 \mathsf{U} \Psi_2$, then $T, \pi \models \Psi$ iff there exists some $i \geq 1$ s.t. $T, \pi[i..] \models \Psi_2$ and for all $j \in [\![1, i]\!]$, $T, \pi[j..] \models \Psi_1$.

Finally, we say that a transition system T entails a CTL* state formula Φ, in symbols $T \models \Phi$, iff $T, q \models \Phi$ for all $q \in I$. For a Golog program $\mathfrak{p}$ and a CTL* state formula, we say $\mathfrak{p}$ entails Φ iff for the transition system T induced by $\mathfrak{p}$, we have that $T \models \Phi$. The *Golog verification problem* is to decide, given a Golog program $\mathfrak{p}$ and a CTL* state formula Φ, whether $\mathfrak{p} \models \Phi$.

4 Verification of Golog programs With CQs

We show that, for the DLs $\mathcal{ALCHIO}$ and $\mathcal{ALCHOQ}$, verification of Golog programs is not harder as conjunctive query entailment. For convenience, we focus on the complementary problem of *satisfiability of CTL*-formulae* defined as follows. Given a program $\mathfrak{p}$ and a CTL* state formula Φ, we say that $\mathfrak{p}$ is satisfiable in $\mathfrak{p}$ if there exists an initial state $q \in I$ in the induced transition system $T = \langle Q, I, \hookrightarrow, Q_{\text{final}}, \lambda\rangle$ s.t. $\mathfrak{p}, q \models \Phi$. It is not hard to see that Φ is satisfiable in $\mathfrak{p}$ iff $\mathfrak{p} \not\models \neg\Phi$, so that the problems are indeed complementary.

In order to decide whether a CTL* state formula is satisfiable by a Golog program, we make use of a technique from [14] for Golog programs over DL actions without conjunctive queries. The idea is to guess a finite representation of the transition system, restricted to the states reachable from one selected initial state. While the CTL* formula can be directly verified on this abstract transition system, in order to decide whether it has a correspondence in the transition system of the program, they construct an exponentially sized *reduction KB*, which is tested for satisfiability. As a result, they obtain complexity bounds that are one exponential higher than those for satisfiability. We use a similar reduction. However, since our Golog program contains CQs, the reduction results in an exponentially sized reduction KB together with an exponentially sized UCQ, which have to be checked for query entailment. While satisfiability checking is 2-ExpTime-complete already for $\mathcal{ALCO}$, we show by inspection of the algorithms presented in [6] and [4] showing inclusion of UCQ entailment in 2-ExpTime for respectively $\mathcal{SHOQ}$ and $\mathcal{ZOI}$, that for these logics, this particular UCQ entailment problem can be decided in double exponential time with respect to the size of the program and the CTL* formula.

4.1 Abstract transition systems

For a given program $\mathfrak{p}$, we denote by $\mathrm{T}(\mathfrak{p}, \mathcal{I})$ the transition system induced by $\mathfrak{p}$ restricted to the states reachable from the state $\langle \mathcal{I}, \mathfrak{p} \rangle$. Formally, for $\langle Q, I, \hookrightarrow, Q_{\text{final}}, \lambda \rangle$ the transition system induced by $\mathfrak{p}$, we define

$$\mathrm{T}(\mathfrak{p}, \mathcal{I}) = \langle Q_{\mathcal{I}}, I_{\mathcal{I}}, \hookrightarrow_{\mathcal{I}}, Q_{\text{final},\mathcal{I}}, \lambda_{\mathcal{I}} \rangle,$$

where

- $I_{\mathcal{I}} = \{\langle \mathcal{I}, \mathfrak{p} \rangle\}$,
- $Q_{\mathcal{I}} = \{q \in Q \mid \langle \mathcal{I}, \mathfrak{p} \rangle \hookrightarrow^* q\}$, where $\hookrightarrow^*$ denotes the transitive-reflexive closure of $\hookrightarrow$,
- $\hookrightarrow_{\mathcal{I}} = (\hookrightarrow) \cap Q_{\mathcal{I}} \times Q_{\mathcal{I}}$,
- $Q_{\text{final}}\mathcal{I} = Q_{\text{final}} \cap Q_{\mathcal{I}}$,
- $\lambda_{\mathcal{I}}$ is obtained from λ by restricting the domain to $Q_{\mathcal{I}}$.

The satisfiability problem can then be formulated as follows: a CTL* state formula Φ is satisfiable by a program $\mathfrak{p}$ iff there exists an interpretation $\mathcal{I}$ s.t. $\mathrm{T}(\mathfrak{p}, \mathcal{I}) \models \Phi$.

We first characterize the interpretations that occur in $\mathrm{T}(\mathfrak{p}, \mathcal{I})$. In the following, let $\mathfrak{p}$ be a fixed program. Denote by $\mathbf{L}$ the set of *relevant literals* l of the forms $A(a)$, $r(a, b)$, where $A \in \mathsf{N_C}$, $r \in \mathsf{N_R}$, $a, b \in \mathsf{N_I}$, and A, r, a and b occur in some effect in some action in $\mathfrak{p}$. Note that, since the number of effects occuring in $\mathfrak{p}$ is linear, $\mathbf{L}$ is at most cubic in size. For every subset $L \subseteq \mathbf{L}$ and interpretation $\mathcal{I}$, define the interpretation $\mathcal{I}^L = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}^L} \rangle$ by setting

- $a^{\mathcal{I}^L} = a^{\mathcal{I}}$ for all $a \in \mathsf{N_I}$,
- $A^{\mathcal{I}^L} = (A^{\mathcal{I}} \cup \{a^{\mathcal{I}} \mid A(a) \in L\}) \setminus \{a^{\mathcal{I}} \mid A(a) \in \mathbf{L} \setminus L\}$, and
- $r^{\mathcal{I}^L} = (r^{\mathcal{I}} \cup \{\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \mid r(a, b) \in L\}) \setminus \{\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \mid r(a, b) \in \mathbf{L} \setminus L\}$.

Intuitively, $\mathcal{I}^L$ is the smallest modification of $\mathcal{I}$ such that every literal in L is satisfied, and none of the literals in $\mathbf{L} \setminus L$ is satisfied.

In the restricted transition system $\mathrm{T}(\mathfrak{p}, \mathcal{I})$, every interpretation occurring in $Q_{\mathcal{I}}$ is obtained from $\mathcal{I}$ by applications of actions in $\mathfrak{p}$. As a consequence, for every state $\langle \mathcal{I}', \mathfrak{p}' \rangle \in Q_{\mathcal{I}}$, there exists some subset $L \subseteq \mathbf{L}$ s.t. $\mathcal{I}' = \mathcal{I}^L$. For this reason, every state in $\mathrm{T}(\mathfrak{p}, \mathcal{I})$ can be identified by a sub-program $\mathfrak{p}' \in \mathrm{sub}(\mathfrak{p})$ and a subset $L \subseteq \mathbf{L}$. As a second observation, we notice that for the verification of the transitions in $\mathrm{T}(\mathfrak{p}, \mathcal{I})$, as well as for the verification of a CTL* formula, the only information relevant about an interpretation $\mathcal{I}'$ in a state $\langle \mathcal{I}', \mathfrak{p}' \rangle \in Q_{\mathcal{I}}$ is which axioms, assertions and CQs they entail. To capture this, we collect all relevant entailments in the set $\mathcal{E}(\mathfrak{p}, \Phi)$, which contains for every axiom, assertion and CQ α occurring in $\mathfrak{p}$ or Φ the two elements α and $\neg\alpha$.

An *abstract transition systems* for $\mathfrak{p}$ and Φ is now a labeled transition system

$$\langle Q_a, I_a, Q_{\text{final,a}}, \hookrightarrow_a, \lambda_a \rangle,$$

where every state $q \in Q_a$ is of the form $\langle L, \mathfrak{p}' \rangle$, where $L \subseteq \mathbf{L}$ and $\mathfrak{p}' \in \text{sub}(\mathfrak{p})$,and the labeling function $\lambda : Q_a \to 2^{\mathcal{E}(\mathfrak{p},\Phi)}$ is such that for every two states $q_1 = \langle L_1, \mathfrak{p}_1 \rangle, q_2 = \langle L_2, \mathfrak{p}_2 \rangle \in Q_a$, $\lambda(q_1) = \lambda(q_2)$ if $L_1 = L_2$. The intuition is that for $q = \langle L, \mathfrak{p}' \rangle \in Q_a$, $\lambda(q)$ contains all relevant entailments of $\mathcal{I}^L$. The transition function $\hookrightarrow_a$ is then defined correspondingly to how it is done for transition systems induced by $\mathfrak{p}$ (see Section 3.2), and entailment of DL-CTL* formulae is defined as usual for labeled transition systems.

To make sure that abstract transition systems correspond to transition systems induced by Golog programs, we make use of the notion of *validity.*

Definition 4. An abstract transition system $\langle Q_a, I_a, Q_{\text{final,a}} \hookrightarrow_a, \lambda_a \rangle$ is *valid* if there exists an interpretation $\mathcal{I}$ s.t. for every state $q = \langle L, \mathfrak{p}' \rangle \in Q_a$ and $\alpha \in \mathcal{E}(\mathfrak{p}, \Phi)$, $\alpha \in \lambda_a(q)$ iff $\mathcal{I}^L \models \alpha$.

The following is an easy consequence of the definition of valid abstract transition systems (see also [14]).

Lemma 5. *Given a program $\mathfrak{p}$ and a DL-CTL* state formula Φ, Φ is satisfiable by $\mathfrak{p}$ iff there exists a valid abstract transition system T for $\mathfrak{p}$ and Φ s.t. $T \models \Phi$.*

Lemma 5 motivates the following procedure for deciding satisfiability of DL-CTL* state formulae: i) guess an abstract transition system T for $\mathfrak{p}$ and Φ, ii) check whether T $\models \Phi$, and iii) check whether T is valid. Since for an abstract transition system $\langle Q_a, I_a, \hookrightarrow_a, \lambda_a \rangle$, we have $Q_a \subseteq 2^{\mathbf{L}} \times \text{sub}(\mathfrak{p})$, the number of states in an abstract transition system is exponentially bounded in the size of $\mathfrak{p}$, while each label in λ_a is linearly bounded in the size of $\mathfrak{p}$ and Φ, each abstract transition system is exponentially bounded in size. Entailment of propositional CTL* formulae from labeled transition systems can be performed in time polynomial in the size of the transition system, which means that it can be done in time exponential in the size of $\mathfrak{p}$ and Φ for abstract transition systems for $\mathfrak{p}$ and Φ. It follows that Step i) can be performed in non-deterministic exponential time, while Step ii) can be performed in exponential time. It thus only remains to establish the complexity of Step iii), which we do in the next subsection.

4.2 Deciding validity of abstract transition systems

To decide whether an abstract transition system T is valid, we construct a KB $\mathcal{K}_\text{T}$ and a positive FO-formula ϕ_T s.t. $\mathcal{K}_\text{T} \models \phi_\text{T}$ iff T is not valid.

Let $\text{T} = \langle Q_a, I_a, \hookrightarrow_a, \lambda_a \rangle$ be an abstract transaction system. Since we require by definition, that for two states $q_1 = \langle L_1, \mathfrak{p}_1 \rangle$, $q_2 = \langle L_2, \mathfrak{p}_2 \rangle \in Q_a$, $\lambda_a(q_1) = \lambda_a(q_2)$ if $L_1 = L_2$, we can abuse notation and treat λ_a as a function over subsets of $L \subseteq \mathbf{L}$ s.t. $\lambda_a(L) = \lambda_a(\langle L, \mathfrak{p}' \rangle)$ for $\langle L, \mathfrak{p}' \rangle \in Q_a$.

Since actions only affect the named part of an interpretation, that is, the assignment of named individuals to concepts and roles, our reduction KB needs to make sure that the interpretation of the unnamed part remains fixed. We use two concepts N and M to distinguish between named and unnamed individuals. Let $\mathsf{ind}(\mathfrak{p})$ be the individual names occurring in $\mathfrak{p}$. $\mathcal{K}_\text{T}$ contains the following axioms to define N and M:

$$N \equiv \bigsqcup_{a \in \mathsf{ind}(\mathfrak{p})} \{a\}$$
$$M \equiv \neg N.$$

For every subset $L \subseteq \mathbf{L}$, and for every name $X \in \mathsf{N_C} \cup \mathsf{N_R}$ occurring in $\mathfrak{p}$ and Φ, we use a fresh name X^L which represents the interpretation of X in $\mathcal{I}^L$ for the individuals occurring in $\mathfrak{p}$. For every $r \in \mathsf{N_R}$, we further define $(r^-)^L = (r^L)^-$.

We use concept names T^L_C to represent the interpretation of concepts C in $\mathcal{I}^L$. Denote by $\mathrm{sub}_c(\mathfrak{p}, \Phi)$ the (sub-)concepts occurring in $\mathfrak{p}$ and Φ. We fix some subset $L^* \subseteq \mathbf{L}$ which serves as a reference point to the interpretation of the unnamed individuals.

For each $C \in \mathrm{sub}(\mathfrak{p}, \Phi)$, we add the following axioms to $\mathcal{K}_\mathrm{T}$ based on the syntactical shape of C:

1. $T^L_A \equiv (N \sqcap A^L) \sqcup (M \sqcap A^{L^*})$ if $C = A$ for $A \in \mathsf{N_C}$,
2. $T^L_C \equiv C$ if C is of one of the forms $\{a\}$, $\top$ or $\bot$, where $a \in \mathsf{N_I}$.
3. $T^L_{\neg D} \equiv \neg T^L_D$ if $C = \neg D$,
4. $T^L_{C_1 \sqcap C_2} \equiv T^L_{C_1} \sqcap T^L_{C_2}$ if $C = C_1 \sqcap C_2$,
5. $T^L_{\exists R.D} \equiv \big(N \sqcap \big(\exists R^{L^*}.(M \sqcap T^L_D) \sqcup \exists R^L.(N \sqcap T^L_D)\big)\big) \sqcup (M \sqcap \exists R^{L^*}.T^L_D)$ if $C = \exists R.D$, and
6. if $C = {\geq} nR.D$:

   $$T_{\geq nR.D} \equiv \left(N \sqcap \bigsqcup_{0 \leq i \leq m} \big({\geq} iR^L.(N \sqcap T^L_D) \sqcap {\geq}(n-i)R^L.(M \sqcap T^L_D)\big) \right) \sqcup (M \sqcap \geq mR^{L^*}.T^L_C),$$

   where $m = \min(n, |\mathsf{ind}(\mathfrak{p})|)$.

Note that, different to the reduction defined in [14], we make sure that the size of the axioms generated by 6 is polynomially bounded by the size of the program, even if we use binary encoding for number restrictions. This will be relevant in Section 5, where we also make use of this reduction.

To encode the actual effects described by $L \subseteq \mathbf{L}$, we add the following assertions for every $L \subseteq \mathbf{L}$ for which there exists a state $\langle L, \mathfrak{p}' \rangle \in Q_a$:

- $A^L(a)$ for every $A(a) \in L$,
- $\neg A^L(a)$ for every $A(a) \notin L$,
- $r^L(a, b)$ for every $r(a, b) \in L$, and
- $(\forall r^L.\neg\{b\})(a)$ for every $r(a, b) \notin L$.

It remains to encode the assertions, axioms and CQs assigned to the effect sets. For the assertions and axioms in $\lambda_a(L)$, this is straightforwardly done by adding

- $T^L_C(a)$ for each $C(a) \in \lambda_a(L)$,
- $\neg T^L_C(a)$ for each $C(a) \in \mathrm{sub}(\mathfrak{p}, \Phi) \setminus \lambda_a(L)$,
- $r^L(a, b)$ for each $r(a, b) \in \lambda_a(L)$,
- $(\forall r^L.\neg\{b\})(a)$ for each $r(a, b) \in \mathrm{sub}(\mathfrak{p}, \Phi) \setminus \lambda_a(L)$,
- $T^L_C \sqsubseteq T^L_D$ for each $C \sqsubseteq D \in \lambda_a(L)$,
- $T^L_C(a^*)$, $\neg T^L_D(a^*)$, where a^* is fresh, for every $C \sqsubseteq D \in \mathrm{sub}(\mathfrak{p}, \Phi) \setminus \lambda_a(L)$,

- $R^L \sqsubseteq S^L$ for each $R \sqsubseteq S \in \lambda_a(L)$, and
- for each $R \sqsubseteq S \in \mathrm{sub}(\mathfrak{p}, \Phi) \setminus \lambda_a(L)$:

$$\begin{aligned}
&\{a^*\} \sqcap M \sqsubseteq \exists R^{L^*}.\{b^*\} \\
&\{a^*\} \sqcap N \sqsubseteq \exists R^{L}.(\{b^*\} \sqcap N) \sqcup \exists R^{L^*}.(\{b^*\} \sqcap M) \\
&\{a^*\} \sqcap M \sqsubseteq \forall S^{L^*}.\neg\{b^*\} \\
&\{a^*\} \sqcap N \sqsubseteq \forall S^{L}.\neg(\{b^*\} \sqcap N) \sqcap \forall S^{L^*}.\neg(\{b^*\} \sqcap M),
\end{aligned}$$

  where a^* and b^* are fresh.

Up to here, the reduction is mostly as in [14] for DLs without CQs. It remains to deal with the CQs in $\mathcal{E}(\mathfrak{p}, \Phi)$. We first introduce a transformation on CQs. For a CQ q and $L \subseteq \mathbf{L}$, denote by q^L the result of replacing every atom of the form $A(t)$ by

$$\Big(\big(A^L(t) \wedge N(t)\big) \vee \Big(A^{L^*}(t) \wedge M(t)\Big) \Big),$$

and every atom of the form $r(t_1, t_2)$ by

$$\Big(\big(N(t_1) \wedge r^L(t_1, t_2) \wedge N(t_2)\big) \vee \Big(M(t_1) \wedge r^{L^*}(t_1, t_2)\Big) \vee \Big(r^{L^*}(t_1, t_2) \wedge M(t_2)\Big) \Big).$$

Furthermore, we define a set $\mathcal{A}_q^L$ of assertions and axioms representing positive entailments of q in $\mathcal{I}^L$. Without loss of generality , we assume that different CQs use different variables. We assign to every variable x occurring in q a fresh individual name a_x^L, and define a function $\cdot^L$ on terms by $a^L = a$ for $a \in \mathsf{N_I}$ and $x^L = a_x^L$ for x a variable. $\mathcal{A}_q^L$ now contains the assertion $T_A^L(t^L)$ for every atom of the form $A(t)$ in q, and the axioms

$$\begin{aligned}
&\{t_1^L\} \sqcap N \sqsubseteq \exists R^L.(\{t_2^L\} \sqcap N) \sqcup \exists R^{L^*}.(\{t_2^L\} \sqcap M) \\
&\{t_1^L\} \sqcap M \sqsubseteq \exists R^{L^*}.\{t_2^L\}
\end{aligned}$$

for every atom of the form $r(t_1, t_2)$ in q. The reduction KB $\mathcal{K}_\mathrm{T}$ now contains for every $L \subseteq \mathbf{L}$ for which there exists a state $\langle L, \mathfrak{p}\rangle \in Q_a$, and for every CQ $q \in \lambda_a(L)$, all assertions in $\mathcal{A}_q^L$. This completes the construction of the reduction KB $\mathcal{K}_\mathrm{T}$. The negated CQs in $\lambda_a(L)$ are represented in the reduction query ϕ_T, which is defined by

$$\phi_\mathrm{T} = \bigvee_{\substack{\langle L, \mathfrak{p}'\rangle \in Q_a \\ q \in \mathcal{E}(\mathfrak{p}, \Phi) \setminus L}} q^L.$$

Note that the construction is polynomial in the number of states in T, which limits the number of literal sets $L \subseteq \mathbf{L}$ that we have to consider. We keep this result in the following corollary, which will be relevant later on.

Corollary 6. *The reducton KB $\mathcal{K}_T$ and the reduction query ϕ_T are both polynomial in the size of T.*

It is now standard to verify that the abstract transition system T is valid if $\mathcal{K}_\mathrm{T} \not\models \phi_\mathrm{T}$.

Lemma 7. *T is valid iff $\mathcal{K}_T \not\models \phi_T$.*

Proof (Sketch). If T is valid, we take the corresponding interpretation $\mathcal{I}$, which we extend to an interpretation $\mathcal{I}_T$ by setting for the fresh names in $\mathcal{K}_\mathrm{T}$:

1. $N^{\mathcal{I}_T} = \{a^{\mathcal{I}} \mid a \in \mathsf{ind}(\mathfrak{p})\}$,
2. $M^{\mathcal{I}_T} = \Delta^{\mathcal{I}} \setminus M^{\mathcal{I}_T}$,
3. $(A^L)^{\mathcal{I}_T} = \{d \mid d \in A^{\mathcal{I}^L}\}$,
4. $(r^L)^{\mathcal{I}_T} = \{\langle d, e\rangle \mid \langle d, e\rangle \in A^{\mathcal{I}^L}\}$
5. $(T_C^L)^{\mathcal{I}_T} = \{d \mid d \in C^{\mathcal{I}^L}\}$.

It is standard to verify that $\mathcal{I}_T$ is a model of $\mathcal{K}_\mathrm{T}$. Furthermore, $\mathcal{I}_T \not\models \phi_\mathrm{T}$, since otherwise, there exists a state $\langle L, \mathfrak{p}'\rangle \in Q_a$ and a CQ $q \in \mathrm{sub}(\mathfrak{p}, \Phi) \setminus \lambda_a(\langle L, \mathfrak{p}'\rangle)$ s.t. $\mathcal{I}^L \models q$. We obtain that $\mathcal{K}_\mathrm{T} \not\models \phi_\mathrm{T}$.

For the other direction, assume $\mathcal{K}_\mathrm{T} \not\models \phi_\mathrm{T}$. Then, there exists a model $\mathcal{I}_T$ of $\mathcal{K}_\mathrm{T}$ s.t. $\mathcal{I}_T \not\models \phi_\mathrm{T}$. We construct an interpretation $\mathcal{I}$ based on $\mathcal{I}_T$ and the initial state $\langle L, \mathfrak{p}\rangle$ as follows:

- for every $A \in \mathsf{N_C}$: $A^{\mathcal{I}} = (A^L)^{\mathcal{I}_T}$,
- for every $r \in \mathsf{N_R}$: $r^{\mathcal{I}} = (r^L)^{\mathcal{I}_T}$.

It is now standard to verify that for every state $\langle L, \mathfrak{p}\rangle \in Q_a$, and every assertion, axiom and CQ $\alpha \in \mathrm{sub}(\mathfrak{p}, \Phi)$, $\mathcal{I}^L \models \alpha$ iff $\alpha \in \lambda_a(\langle L, \mathfrak{p}\rangle)$. It follows that the abstract transition system T is valid. □

Note that both the reduction KB $\mathcal{K}_\mathrm{T}$ and the reduction query ϕ_T are in size polynomial in the size of the abstract transition system T, and thus exponential with respect to the size of $\mathfrak{p}$ and Φ. For Golog Programs and DL-CTL* formulae without conjunctive queries, the test in Lemma 7 actually corresponds to a pure satisfiability test. KB-satisfiability can be decided in ExpTime for $\mathcal{ALCHOQ}$ and $\mathcal{ALCHOIQ}$, and in NExpTime for $\mathcal{ALCHOIQ}$, which is the reason why validity of abstract transaction systems without conjunctive queries can be decided in 2-ExpTime for $\mathcal{ALCHOQ}$ and $\mathcal{ALCHOI}$, and in N2-ExpTime for $\mathcal{ALCHOIQ}$ [14, Lemma 4.17].

A straight-forward application of the known complexities of query entailment however does not lead to the desired complexity bounds: query entailment is 2-ExpTime-hard already for $\mathcal{ALCO}$, so that this would give us only a 3-ExpTime-bound. However, a close inspection of the decidability procedures for UCQ entailment in $\mathcal{ALCHOQ}$ from [6], and for UCQ entailment in $\mathcal{ALCHOI}$ in [4] shows that the entailment in Lemma 7 can actually be performed in 2-ExpTime for these logics. For $\mathcal{ALCHOIQ}$, so far only decidability of query entailment is known, but no tight complexity bounds nor even elementary upper bounds. We therefore leave the precise bound for this logic open, and only note that Lemma 7 implies decidability of the validity problem.

Lemma 8. *For $\mathcal{L} \in \{\mathcal{ALCHOQ}, \mathcal{ALCHOI}\}$ and a natural number n, entailment of UCQs from $\mathcal{L}$-KBs can be decided in time double exponential in n provided that*

- *the size of the KB is at most exponential in n,*
- *the number of disjuncts in the CQ is at most exponential in n, and*
- *the size of each CQ is polynomial in n.*

Proof. We first show the case for $\mathcal{L} = \mathcal{ALCHOQ}$. The result follows from a close analysis of the 2-ExpTime decision procedure for entailment of UCQs from $\mathcal{SHOQ}$ KBs presented in [6].

The decision procedure also applies to the extension $\mathcal{SHOQ}^{\sqcap}$ of $\mathcal{SHOQ}$ with role conjunctions. Specifically, to decide entailment of a CQ, a set $\mathsf{con}_{\mathcal{K}}(q)$ of CQs of the form $C_1(x_1) \vee \ldots \vee C_n(x_n)$ is defined, where each C_i, $i \in [\![1,n]\!]$ is a $\mathcal{SHOQ}^{\sqcap}$-concept and $x_i \neq x_j$ for $i \neq j \in [\![1,n]\!]$. For UCQs q, $\mathsf{con}_{\mathcal{K}}(q)$ is defined as the union of all sets $\mathsf{con}_{\mathcal{K}}(q')$ where $q' \in q$. Intuitively, each $\in \mathsf{con}_{\mathcal{K}}(q)$ corresponds to a forest-shaped match in some model of $\mathcal{K}$, and every such match is represented by some CQ in $\mathsf{con}_{\mathcal{K}}(q)$. A consequence of this is that $\mathcal{K} \models \bigvee_{q_i \in \mathsf{con}_{\mathcal{K}}(q)} q_i$ iff $\mathcal{K} \models q$ [6, Theorem 6]. The size bounds on $\mathsf{con}_{\mathcal{K}}(q)$ are given in [6, Lemma 7]: its size is at most i) polynomial in the size of $\mathcal{K}$ and ii) exponential in the size of q. For a UCQ whose number of CQs is exponentially bounded in n and the size of each CQ is polynomially bounded in n, from the fact that $\mathsf{con}_{\mathcal{K}}(q)$ corresponds to the union of all sets $\mathsf{con}_{\mathcal{K}}(q')$ where $q' \in q$, it follows that $\mathsf{con}_{\mathcal{K}}(q)$ is exponential in both n and the size of $\mathcal{K}$.

Now to decide $\mathcal{K} \models \bigvee_{q_i \in \mathsf{con}_{\mathcal{K}}(q)} q_i$, we use the fact that the atoms in each query in $\mathsf{con}_{\mathcal{K}}(q)$ are variable-disjoint, and built a sequence of *reduction KBs*. Each reduction KB is obtained from selecting from each CQ in $q_i\mathsf{con}_{\mathcal{K}}(q)$ one atom $C_i(x_i)$, and adding $\top \sqsubseteq \neg C_i$ to $\mathcal{K}$. If one of them is unsatifiable, $\mathcal{K} \not\models \bigvee_{q_i \in \mathsf{con}_{\mathcal{K}}(q)} q_i$ and $\mathcal{K} \not\models q$.

The size of each KB is linear in $\mathsf{con}_{\mathcal{K}}(q)$, and there are exponentially many possible choices wrt. the size of $\mathsf{con}_{\mathcal{K}}(q)$. If q is shaped as in the lemma, this amounts to a number of satisfiability tests that is double exponential in n, where each KB is exponential in the size of n. As shown in the remainder of [6], each such satisfiabiltiy test can be performed in time exponential to the size of the KB, so that we obtain that the overall decision procedure runs in time double exponential in n.

We now consider the case where $\mathcal{L} = \mathcal{ALCHOI}$, where we inspect the procedure presented in [4] deciding entailment of *regular path queries* (*RPQs*) in $\mathcal{ZOI}$-KBs. Regular path queries are a generalisation and UCQs, and $\mathcal{ZOI}$ is an extension of $\mathcal{ALCHOI}$, so that this procedure can also be used to decide UCQ entailment for $\mathcal{ALCHOI}$-KBs. The authors in [4] reduce UCQ-entailment to the emptiness problem for *one-way non-deterministic parity tree automata* (*1NPAs*). Specifically, given a $\mathcal{ZOI}$ KB $\mathcal{K}$ and a RPQ q, they construct a 1NPA $\mathbf{A}_{\mathcal{K}\not\models q}$ which accepts the empty language iff $\mathcal{K} \models q$. We first give an overview over the main ideas, before we argue why this approach can decide bounded CQs as in the Lemma in double-exponential time. Specifically, the authors exploit the fact that $\mathcal{ZOI}$ has the forest model property for query-entailment, that is, query entailment from $\mathcal{ZOI}$-KBs $\mathcal{K}$ can be completely characterized by restricting to models of $\mathcal{K}$ that can be mapped to a labeled forest where every node represents a domain element, edges correspond to role-connections and the roots of the forest to the named individuals, and every role-connection that does not go to a named individual has a corresponding edge in the forest. To characterise these forest models by means of automata, they represent forest-interpretations directly in labeled trees of fixed branching degree, where the branching degree is bounded by the KB and nodes are labeled with sets of individual, concept and role names. Here, the role names refer to the incoming edge, and individual names only occur on the direct successors of the root [4, Definition 3.7 and beginning of Section 4]. They then construct a *fully enriched automaton* (*FEA*) with a polynomial number of states and a constant index, which they step-wise translate into an 1NPA of $\mathbf{A}_{\mathcal{K}}$ accepting exactly those labeled trees that correspond to a forest-shaped model of $\mathcal{K}$. $\mathbf{A}_{\mathcal{K}}$ has a double-exponential number of states and a constant index[1].

To capture query entailment, they define another 1NPA $\mathbf{A}_{\neg q}$ which accepts exactly those labelled trees that correspond to a forest-shaped interpretation in which the query q is not entailed, and

[1]The authors only explicitly spell out the size of the two-way alternating parity tree (2APA) which they construct in the second-last step of their transformation and then translate to the 1NPA. The number of states of the 2APA is polynomially bounded in the size of $\mathcal{K}$ and has a constant index. However, according to [5, Proposition 2.12], the transformation comes with an exponential blow-up in the number of states, while it keeps the index. Therefore, the final 1NPA $\mathbf{A}_{\mathcal{K}}$ constructed here has a double-exponential number of states and a constant index.

build the intersection of the automata $\mathbf{A}_{\mathcal{K}}$ and $\mathbf{A}_{\neg q}$, which is the final 1NPA $\mathbf{A}_{\mathcal{K}\not\models q}$. If $\mathbf{A}_{\mathcal{K}\not\models q}$ is empty, that is, the language of accepted trees is empty, there cannot be a forest-shaped model of $\mathcal{K}$ in which q is not entailed, and correspondingly, $\mathcal{K} \models q$.

While [4] only briefly sketch the automaton $\mathbf{A}_{\neg q}$, a construction of an automaton accepting the same language is described in detail in [5]: this automaton has an exponential number of states while its index is exponential. The relevant construction in [5, Section 5] can be easily adapted so that number of states and index depend only on the size of the query, as the only relevant factors are the variables, concept and role names occurring in it.

It is not hard to see that for a UCQ $q = q_1 \vee \ldots \vee q_m$ the automaton $\mathcal{A}_{\neg q}$ is equivalent to the intersection of the automata $\mathcal{A}_{\neg q_1}, \ldots, \mathcal{A}_{\neg q_m}$, as the disjuncts can be tested independently on a given interpretation. Now the number of states and the index of the intersection of two 1NPAs is determined as follows [5, Proposition 2.15]: Let $Q(\mathbf{A})$ denote the states of $\mathbf{A}$ and $\mathrm{ind}(\mathbf{A})$ its index, then

$$\mathrm{ind}(\mathbf{A}_1 \cap \mathbf{A}_2) = O(f(\mathbf{A}_1, \mathbf{A}_2))$$

and

$$|Q(\mathbf{A}_1 \cap \mathbf{A}_2)| \leq 2^{O(f(\mathbf{A}_1,\mathbf{A}_2)^2)} \cdot f(\mathbf{A}_1, \mathbf{A}_2) \cdot |Q(\mathbf{A}_1)| \cdot |Q(\mathbf{A}_2)|,$$

where $f(\mathbf{A}_1, \mathbf{A}_2) = \mathbf{A}_1 + \mathbf{A}_2 + 1$. Let $\mathbf{A}'_{\neg q} \equiv \mathbf{A}_{\neg q}$ denote the intersection of $\mathbf{A}_{q_1}, \ldots, \mathbf{A}_{q_n}$. We obtain that

$$\mathrm{ind}(\mathbf{A}'_{\neg q}) = O\left(\sum_{1\leq i\leq m} \mathrm{ind}(\mathbf{A}_{\neg q_i}) + 1\right),$$

and

$$|Q(\mathbf{A}'_{\neg q})| \leq 2^{O\left(\left(\sum_{1\leq i\leq n} \mathrm{ind}(\mathbf{A}_{\neg q_i})+1\right)^{2m}\right)} \cdot \left(\sum_{1\leq i\leq n} \mathrm{ind}(\mathbf{A}_{\neg q_i}) + 1\right) \cdot \Pi_{1\leq i\leq n}|Q(\mathbf{A}_{\neg q_i})|$$

Since for each $\mathbf{A}_{\neg q_i}$, the $\mathrm{ind}(\mathbf{A}_{\neg q_i})$ is single exponential in the size of q_i, and $|Q(\mathbf{A}_{\neg q_i})|$ is double exponential in the size of q_i, we obtain that, provided that q is shaped as in the lemma, $\mathrm{ind}(\mathbf{A}'_{\neg q})$ is single exponential in n and $|Q(\mathbf{A}'_{\neg q})|$ is double exponential in n, and the same holds for $\mathbf{A}'_{\mathcal{K}\not\models q} = \mathbf{A}_{\mathcal{K}} \cap \mathbf{A}'_{\neg q}$. Emptiness of 1NDAs $\mathbf{A}$ can be decided in time $O(|Q(\mathbf{A})|^{\mathrm{ind}(\mathbf{A})})$, which is double exponential in n for $\mathbf{A}'_{\mathcal{K}\not\models q}$. We obtain that for $\mathcal{ALCHOI}$-KBs $\mathcal{K}$, entailment of UCQs q can be decided in time double exponential in n, provided that the size of $\mathcal{K}$ and the number of CQs in q is exponentially bounded in n and the size of each CQ is polynomially bounded in n. □

Lemma 9. *Validity of abstract transition systems for $\mathcal{L}$-programs $\mathfrak{p}$ and $\mathcal{L}$-CTL* state formulae Φ with CQs*

- *is decidable for Golog programs and CTL* formulae for $\mathcal{L} = \mathcal{ALCHOIQ}$ concepts,*
- *is decidable in time double exponential in the size of $\mathfrak{p}$ and Φ for $\mathcal{L} \in \{\mathcal{ALCHOQ}, \mathcal{ALCHOI}\}$.*

Proof. For $\mathcal{ALCHOIQ}$, it suffices to consider Lemma 7 observe that query entailment for $\mathcal{ALCHOIQ}$ KBs is decidable. For $\mathcal{ALCHOQ}$ and $\mathcal{ALCHOI}$, we observe that the reduction query ϕ_{T} can be transformed into an equivalent UCQ ϕ'_{T} by distributing in each disjunct over conjunction. In ϕ'_{T}, each CQ is polynomially bounded in the size of the largest CQ occurring in $\mathfrak{p}$ and Φ, and ϕ_{T} has an exponential number of these CQs. The size of $\mathcal{K}_{\mathrm{T}}$ is exponential in the size of $\mathfrak{p}$ and Φ. By Lemma 8, we obtain that $\mathcal{K}_{\mathrm{T}} \models \phi'_{\mathrm{T}}$ can be decided in time double exponential in the size of $\mathfrak{p}$ and Φ, and by Lemma 7, that validity of abstract transition systems with CQs can be decided in 2-ExpTime in the size of $\mathfrak{p}$ and Φ for $\mathcal{L} \in \{\mathcal{ALCHOQ}, \mathcal{ALCHOI}\}$. □

Theorem 10. *The projection and the program verification problem for DL actions with CQs is decidable for actions and properties formulated over $\mathcal{ALCHOIQ}$, and* 2-ExpTime*-complete for actions and properties formulated over $\mathcal{ALCHOQ}$ or $\mathcal{ALCHOI}$.*

Proof. We note that a 2-ExpTime-hardness follows from the corresponding complexity of CQ entailment for $\mathcal{ALCO}$-KBs [10]. We also note that the projection problem can be straightforwardly encoded into a verification problem, by defining a program that just executes a sequence of actions and then marks the end of the execution by making some fresh assertion true, and a CTL*-property that checks whether the property to be checked is always true at the end of this execution.

To get the upper bounds for the verification problem, we describe an algorithm that proceeds as follows. Since we aim for a deterministic complexity class, it suffices to focus on the complementary problem of CTL* state formulae satisfiability by Golog programs. Let $\mathfrak{p}$ be the Golog program and Φ the CTL* state formula. By Lemma 5, to show that Φ is satisfiable by $\mathfrak{p}$, we need to find a valid abstract transition system T for $\mathfrak{p}$ and Φ s.t. $\mathrm{T} \models \Phi$. The number of possible states in an abstract transition system is exponentially bounded, so that there are at most double exponentially many abstract transition systems to consider. We construct these one by one, and check for each whether they are valid and whether they entail Φ. Both tasks are decidable, so that be obtain decidability for $\mathcal{ALCHOIQ}$. For $\mathcal{ALCHOQ}$ and $\mathcal{ALCHOI}$, we observe that we require at most double exponentially many iterations, in each of which we perform validity in 2-ExpTime, and check for entailment of CTL* in time polynomial in the number of states, and thus in ExpTime. Thus, the resulting procedure runs in 2-ExpTime, so that we obtain the desired upper bound. □

5 Conflicts and Interactions

5.1 Detecting conflicting actions

As the the conflict problem, the conditionalised conflict problem, as well as the action interaction problem are new, we investigate their complexity in our setting, but also their respective variants for DL actions and DL formulae without CQs.

Since query entailment is already 2-ExpTime-hard for $\mathcal{ALCO}$, it suffices to show that the conflict problem and the conditionalised conflict problem can be decided in 2-ExpTime. For this, we encode the conflict problem into a Golog verification problem, which by Theorem 10, is 2-ExpTime-complete for $\mathcal{L} \in \{\mathcal{ALCHOQ}, \mathcal{ALCHOI}\}$. For the case without CQs, we will need to be a bit more careful, since here we can obtain complexity bounds that are lower than the respective verification problem, which is still 2-ExpTime-complete for $\mathcal{L} \in \{\mathcal{ALCHOQ}, \mathcal{ALCHOI}\}$, and coN2ExpTime-complete for $\mathcal{L} = \mathcal{ALCHOIQ}$.

Note that every composite can also be seen as a Golog program that uses the sequence-operator to connect atomic actions. In the following, it will be convenient to enforce that every branch in the transition system ends in an accepting state, and to mark paths that correspond to non-accepting states in a different manner. Let Fail be a fresh concept name. We inductively define the program $\mathsf{acc}(\mathfrak{a})$ on composite actions $\mathfrak{a}$ by setting $\mathsf{acc}(\epsilon) = \epsilon$ and $\mathsf{acc}(\phi \rhd \mathfrak{E}; \mathfrak{a}) = \big((\neg\phi?; \top \rhd \mathsf{Fail}(a))|\top \rhd \mathfrak{E}; \mathsf{acc}(\mathfrak{a})\big)$, where a is any individual name. If $\mathfrak{a}$ is executable on an interpretation $\mathcal{I}$, then $\mathsf{acc}(\mathfrak{a})$ will perform the same adaptations on $\mathcal{I}$ than $\mathfrak{a}$. If $\mathfrak{a}$ is not executable on $\mathcal{I}$, then $\mathsf{acc}(\mathfrak{a})$ will have an accepting path then terminates in a state satisfying $\mathsf{Fail}(a)$.

Let $\mathcal{K}$ be a KB, $\mathfrak{a}_1$ and $\mathfrak{a}_2$ two actions and ϕ a DL formula. We define the program $\mathfrak{p}$ as follows, where A_1, A_2 and A_3 are fresh concept names and a is just any individual name.

$$
\begin{aligned}
\top \rhd \{\ominus A_1(a), \ominus A_2(a), \ominus A_3(a)\}; (& \\
& (\mathfrak{a}_1; \top \rhd \oplus A_1(a)) \\
| \; & (\mathfrak{a}_2; \top \rhd \oplus A_2(a)) \\
| \; & ((\mathsf{acc}(\mathfrak{a}_1) \| \mathsf{acc}(\mathfrak{a}_2)); (\top \rhd \oplus A_3(a)) \\
&).
\end{aligned}
$$

The program non-deterministically picks one of three options 1) execute $\mathfrak{a}_1$, 2) execute $\mathfrak{a}_2$, and 3) execute both in parallel (which corresponds to: non-deterministically execute some action $\mathfrak{a} \in \mathrm{Seq}(\mathsf{acc}(\mathfrak{a}_1) \| \mathsf{acc}(\mathfrak{a}_2)))$. The assertions $A_1(a)$, $A_2(a)$ and $A_3(a)$ are used to mark the respective choices. Note that every action $\mathfrak{a} \in \mathrm{Seq}(\mathsf{acc}(\mathfrak{a}_1) \| \mathsf{acc}(\mathfrak{a}_2))$ corresponds to an action in $\mathrm{Seq}(\mathfrak{a}_1 \| \mathfrak{a}_2)$, but so that it has an accepting path in the transition system induced by $\mathfrak{p}$. However, if $\mathfrak{a}$ is not executable, then the corresponding path in the transition system will not lead to $A_3(a)$ to be added to the interpretation. For an interpretation in which $\mathfrak{a}_1$ and $\mathfrak{a}_2$ are exetable, we can thus check whether in this interpretation also every action $\mathfrak{a} \in \mathrm{Seq}(\mathfrak{a}_1 \| \mathfrak{a}_2)$ is executable, by checking whether every path in $\mathfrak{p}$ at some point makes one of $A_1(a)$, $A_2(a)$ or $A_3(a)$ true.

For the conflict problem, we thus have to verify the following CTL* formula, which is entailed by the program iff $\mathfrak{a}_1$ and $\mathfrak{a}_2$ are not conflicting.

$$
\Phi_1 = \Big(\Big((\mathcal{K} \wedge \mathsf{E}\Diamond A_1(a) \wedge \mathsf{E}\Diamond A_2(a)\Big) \rightarrow \mathsf{A}\Big(\Diamond A_1(a) \vee \Diamond A_2(a) \vee \Diamond A_3(a)\Big)\Big)
$$

The left hand side of the formula restricts to those initial interpretations of the transition system which are models of $\mathcal{K}$ on which both $\mathfrak{a}_1$ and $\mathfrak{a}_2$ are executable. The right-hand side then checks whether on these interpretations, also every element $\mathfrak{a} \in (\mathfrak{a}_1 \| \mathfrak{a}_2)$ is executable.

Similarly, the conditionalised conflict problem for a given invariant ϕ is captured by the following CTL* formula, which is entailed iff $\mathfrak{a}_1$ and $\mathfrak{a}_2$ are not conflicting with respect to the DL formula ϕ.

$$
\begin{aligned}
\Phi_2 = \Big(\Big((\mathcal{K} \wedge \mathsf{E}\Diamond(A_1(a) \wedge \phi) \wedge \mathsf{E}\Diamond(A_2(a) \wedge \phi)\Big) & \\
\rightarrow \mathsf{A}\Big(\Diamond(A_1(a) \wedge \phi) \vee \Diamond(A_2(a) \wedge \phi) \vee \Diamond(A_3(a) \wedge \phi)\Big)\Big)
\end{aligned}
$$

It follows that both problems are not harder than the verification problem, which indeed already corresponds to the complexity of query entailment (Theorem 10).

Theorem 11. *For DL actions* with CQs *over $\mathcal{L}$, both the conflict problem and the conditionalised conflict problem are*

- 2-ExpTime-*complete for $\mathcal{L} \in \{\mathcal{ALCHIO}, \mathcal{ALCHOQ}\}$, and*
- *decidable for $\mathcal{L} = \mathcal{ALCHIOQ}$.*

For DL actions and DL formulae without CQs, they are

- ExpTime-*complete for $\mathcal{L} \in \{\mathcal{ALCHIO}, \mathcal{ALCHOQ}\}$, and*
- NExpTime-*complete for $\mathcal{L} = \mathcal{ALCHIOQ}$.*

Proof. All hardness results follow from classical entailment problems, which can be straightforwardly reduced to action executability, which is a necessary condition for both conflict problems (if one of the actions is not executable, then the actions are trivially not conflicting). For

DL actions with CQs, the upper bound follows directly from the above construction and the complexity of Golog program verification (Theorem 10). For DL actions without CQs, we have a closer look on how to decide the problems more clever. In order for two formulae are conflicting (with respect to the DL-formulae ϕ), we have to show the non-entailment of Φ_1 (Φ_2), which corresponds to the satisfiability of the following formulae.

$$\Phi_1' = \mathcal{K} \wedge \mathsf{E}\Diamond A_1(a) \wedge \mathsf{E}\Diamond A_2(a) \wedge \mathsf{E}\Big(\neg\Diamond A_1(a) \wedge \Diamond A_2(a) \wedge \neg\Diamond A_3(a)\Big)$$

$$\begin{aligned}\Phi_2' = \mathcal{K} \wedge \mathsf{E}\Diamond(A_1(a) \wedge \phi) \wedge \mathsf{E}\Diamond(A_2(a) \wedge \phi) \\ \wedge \mathsf{E}\Big(\neg\Diamond\big(A_1(a) \wedge \phi\big) \wedge \neg\Diamond\big(A_2(a) \wedge \phi\big) \wedge \neg\Diamond\big(A_3(a) \wedge \phi\big)\Big)\end{aligned}$$

Note that both formulae only rely on the existance of three paths, namely one corresponding to $\mathfrak{a}_1$ and satisfying $A_1(a)$ at the end, one corresponding to $\mathfrak{a}_2$ and satisfying $A_2(a)$ at the end, and one corresponding to some interleaved action $\mathfrak{a} \in \mathrm{Seq}(\mathfrak{a}_1 \| \mathfrak{a}_2)$, which necessarily has to satisfy $A_3(a)$ at the end, and thus has to satisfy $\neg\phi$ at this point. Intuitively, that last path corresponds to the action that witnesses the conflict between $\mathfrak{a}_1$ and $\mathfrak{a}_2$. As $\mathrm{Seq}(\mathfrak{a}_1 \| \mathfrak{a}_2)$ contains at most exponentially elements, we can *guess* this witness $\mathfrak{a} \in \mathrm{Seq}(\mathfrak{a}_1 \| \mathfrak{a}_2)$, and then test for satisfiability in the following program $\mathfrak{p}_{\mathfrak{a}}$.

$$\begin{aligned}\top \rhd \{\ominus A_1(a), \ominus A_2(a), \ominus A_3(a)\}; (& \\ &(\mathfrak{a}_1; \top \rhd \oplus A_1(a)) \\ &\mid (\mathfrak{a}_2; \top \rhd \oplus A_2(a)) \\ &\mid (\mathfrak{a}; (\top \rhd \oplus A_3(a)) \\ &).\end{aligned}$$

If there exists some $\mathfrak{a} \in \mathrm{Seq}(\mathfrak{a}_1 \| \mathfrak{a}_2)$ s.t. Φ_1' (respectively Φ_2') is satisfiable in $\mathfrak{p}_{\mathfrak{a}}$, then $\mathfrak{p} \not\models \Phi_1$ ($\mathfrak{p} \not\models \Phi_2$), and consequently, $\mathfrak{a}_1$ and $\mathfrak{a}_2$ are conflicting in $\mathcal{K}$ (with respect to ϕ). As there are only three paths in each $\mathfrak{p}_{\mathfrak{a}}$, it is not hard to see that every abstract transition system T for $\mathfrak{p}_{\mathfrak{a}}$ has a number of states that is linear in $\mathfrak{p}_{\mathfrak{a}}$, and thus linear in $\mathcal{K}$, $\mathfrak{a}_1$ and $\mathfrak{a}_2$. By Corollary 6, satisfiability of Φ_1' and Φ_2' in $\mathfrak{p}_{\mathfrak{a}}$ can be decided using a polynomially sized reduction KB. Note that a reduction CQ is not required, since we assume our input does not contain any CQs. We thus obtain an NExpTime-procedure for $\mathcal{L} = \mathcal{ALCHOIQ}$ that works as follows: 1) guess an action $\mathfrak{a} \in \mathrm{Seq}(\mathfrak{a}_1 \| \mathfrak{a}_2)$ in polynomial time, 2) guess an abstract transition system for $\mathfrak{p}_{\mathfrak{a}}$ in polynomial time, 3) verify that $T \models \Phi_1'$ ($T \models \Phi_2'$) in polynomial time, and 4) verify validity of T by checking satisfiability of the reduction KB, which can be done in non-deterministic exponential time [13]. We obtain that for both conflict problems are NExpTime-complete for the case without CQs and with $\mathcal{L} = \mathcal{ALCHOIQ}$. For $\mathcal{L} \in \{\mathcal{ALCHOQ}, \mathcal{ALCHOI}\}$, Step 1) and 2) can be performed by deterministically iterating over all possible choices, and Step 4) can be performed in exponential time [13]. We obtain that for $\mathcal{L} \in \{\mathcal{ALCHOQ}, \mathcal{ALCHOI}\}$, both conflicy problems can be decided in ExpTime. □

5.2 Detecting interacting actions

We investigate the complexity of determining whether two actions are interacting in a given KB. For this, we focus on a simpler problem stated as follows.

Definition 12. Given a KB $\mathcal{K}$ and two actions $\mathfrak{a}_1$ and $\mathfrak{a}_2$, $\mathfrak{a}_1$ and $\mathfrak{a}_2$ are equivalent in $\mathcal{K}$, in symbols $\mathfrak{a}_1 \equiv_{\mathcal{K}} \mathfrak{a}_2$, iff for every model $\mathcal{I}$ of $\mathcal{K}$, $\mathcal{I}^{\mathfrak{a}_1} = \mathcal{I}^{\mathfrak{a}_2}$.

Two actions $\mathfrak{a}_1$ and $\mathfrak{a}_2$ are interacting in a KB $\mathcal{K}$ iff there exists two actions $\mathfrak{a}_1', \mathfrak{a}_2' \in \mathrm{Seq}(\mathfrak{a}_1 \| \mathfrak{a}_2)$ s.t. $\mathfrak{a}_1' \not\equiv_{\mathcal{K}} \mathfrak{a}_2'$. As the number of elements in $\mathrm{Seq}(\mathfrak{a}_1 \| \mathfrak{a}_2)$ is exponential in the size of $\mathfrak{a}_1$ and $\mathfrak{a}_2$, and reasoning is already at least ExpTime-hard for all the DLs considered here, it thus suffices to focus on the problem of action non-equivalence.

Theorem 13. *Deciding whether two actions over $\mathcal{L}$ are not equivalent, as well as whether they are interacting, in an $\mathcal{L}$ KB is*

1. *decidable for $\mathcal{L} = \mathcal{ALCHOIQ}$, and*
2. 2-ExpTime-*complete for $\mathcal{L} \in \{\mathcal{ALCHOQ}, \mathcal{ALCHOI}\}$.*

For actions without CQs, it is

1. NExpTime-*complete for $\mathcal{L} = \mathcal{ALCHOIQ}$ provided that the actions do not contain CQs, and*
2. ExpTime-*complete for $\mathcal{L} \in \{\mathcal{ALCHOQ}, \mathcal{ALCHOI}\}$ provided that the actions do not contain CQs.*

Proof. We start with the non-equivalence problem. Let $\mathfrak{a}_1 = \mathfrak{a}_1^1; \ldots; \mathfrak{a}_1^n$ and $\mathfrak{a}_2 = \mathfrak{a}_2^1; \ldots; \mathfrak{a}_2^m$, and let $\mathcal{K}$ be a KB. $\mathfrak{a}_1$ and $\mathfrak{a}_2$ are not equivalent if there exists a model $\mathcal{I}$ of $\mathcal{K}$ s.t. $\mathcal{I}^{\mathfrak{a}_1} \neq \mathcal{I}^{\mathfrak{a}_2}$.

We first proceed similar as for the conflicts, that is, by specifying a program $\mathfrak{p}$ that generates the two interpretations $\mathcal{I}^{\mathfrak{a}_1}$ and $\mathcal{I}^{\mathfrak{a}_1}$ from an initial interpretation $\mathcal{I}$. This program is simply the following.

$$\mathfrak{p} = \mathcal{K}?(\mathfrak{a}_1|\mathfrak{a}_2); \mathsf{Fail}$$

We guess an abstract transition system T for $\mathfrak{p}$ and construct the corresponding reduction KB $\mathcal{K}_T$ and reduction query ϕ_T. Similar as in the proof for Theorem 11, we can argue that T, $\mathcal{K}_T$ and ϕ_T are polynomial in size with respect to to $\mathcal{K}$, $\mathfrak{a}_1$ and $\mathfrak{a}_2$.

If for every $\mathcal{K}_T \not\models \phi_T$, we know that T is valid, that is, it witnesses a model of $\mathcal{I}$ on which $\mathfrak{a}_1$ and $\mathfrak{a}_2$ are executable, and we only have to verify that furthermore, $\mathcal{I}^{\mathfrak{a}_1} \neq \mathcal{I}^{\mathfrak{a}_2}$. T has two states $\langle L_1, \epsilon\rangle$, $\langle L_1, \epsilon\rangle$ which describe the situation after respectively $\mathfrak{a}_1$ and $\mathfrak{a}_2$ have been executed, that is, for which $\mathcal{I}^{\mathfrak{a}_1} \neq \mathcal{I}^{L_1}$ and $\mathcal{I}^{\mathfrak{a}_2} \neq \mathcal{I}^{L_2}$. Clearly, $\mathcal{I}^{\mathfrak{a}_1} \neq \mathcal{I}^{\mathfrak{a}_2}$ iff $L_1 \neq L_2$, which can be trivially decided in polynomial time.

By guessing T or iterating over the possible choices, we obtain that non-equivalence is not harder than query non-entailment for the logics considered, and not harder than satisfiability if the actions do not contain CQs. We thus obtain the complexities in the Theorem for this reasoning problem. To decide whether two actions $\mathfrak{a}_1$ and $\mathfrak{a}_2$ are interacting, we guess two actions $\mathfrak{a}_1', \mathfrak{a}_2' \in \mathrm{Seq}(\mathfrak{a}_1 \| \mathfrak{a}_2)$, and decide whether they are not equivalent, which can be performed by an exponential number of iterations if we target a deterministic complexity class. Thus, also for the action interaction problem, we obtain the complexities stated in the theorem. □

6 Conclusion

We presented a variant of DL actions and Golog programs over these actions, where in addition to earlier work, also conjunctive queries can be used as tests. This was motivated by our use case in self-adaptive systems, where CQs are used to detect system adaptations. For the DLs we considered in our study, our results show that executability and projection for these actions, as well as verification of temporal properties in the extended Golog programs, is not harder than checking whether a conjunctive query is entailed by a DL knowledge base, provided that this complexity is known. For executability and the projection, this corresponds to the situation as for DL actions without CQs, for which executability and projection is not harder than axiom entailment. On the other hand, for the verification problem, the result comes at a surprise, since in the absense of CQs, verification is an exponential harder than the projection problem [14].

In addition to the traditional problems executability, projection and verification of Golog programs, we also considered new reasoning tasks that are concerned with the interactions between actions executed in parallel. In particular, we considered the *conflict problem*, which determines whether for any model of the KB, parallel execution of two actions always preserves executability, the *conditionalised conflict problem*, which determines whether parallel execution of two actions always preserves a given post-condition, and the *action interaction problem*, which asks whether the outcome of two actions executed in parallel always yields the same result, provided that an interleaving semantics of parallel execution is used. Our results show that these reasoning problems are not harder than the projection problem, even for actions without conjunctive queries.

References

[1] Franz Baader, Carsten Lutz, Maja Milicic, Ulrike Sattler, and Frank Wolter. Integrating description logics and action formalisms: First results. In Manuela M. Veloso and Subbarao Kambhampati, editors, *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pages 572–577. AAAI Press / The MIT Press, 2005.

[2] Franz Baader and Benjamin Zarrieß. Verification of golog programs over description logic actions. In Pascal Fontaine, Christophe Ringeissen, and Renate A. Schmidt, editors, *Frontiers of Combining Systems - 9th International Symposium, FroCoS 2013, Nancy, France, September 18-20, 2013. Proceedings*, volume 8152 of *Lecture Notes in Computer Science*, pages 181–196. Springer, 2013.

[3] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, and Riccardo Rosati. Ontology-based database access. In Michelangelo Ceci, Donato Malerba, and Letizia Tanca, editors, *Proceedings of the Fifteenth Italian Symposium on Advanced Database Systems, SEBD 2007, 17-20 June 2007, Torre Canne, Fasano, BR, Italy*, pages 324–331, 2007.

[4] Diego Calvanese, Thomas Eiter, and Magdalena Ortiz. Regular path queries in expressive description logics with nominals. In *Proceedings of IJCAI*, pages 714–720, 2009.

[5] Diego Calvanese, Thomas Eiter, and Magdalena Ortiz. Answering regular path queries in expressive description logics via alternating tree-automata. *Inf. Comput.*, 237:12–55, 2014.

[6] Birte Glimm, Ian Horrocks, and Ulrike Sattler. Unions of conjunctive queries in $\mathcal{SHOQ}$. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference, KR 2008*, pages 252–262, 2008.

[7] Birte Glimm, Carsten Lutz, Ian Horrocks, and Ulrike Sattler. Conjunctive query answering for the description logic SHIQ. *J. Artif. Intell. Res.*, 31:157–204, 2008.

[8] Marcus Hähnel, Julian Mendez, Veronika Thost, and Anni-Yasmin Turhan. Bridging the application knowledge gap: using ontology-based situation recognition to support energy-aware resource scheduling. In Fábio M. Costa and Anders Andersen, editors, *Proceedings of the 13th Workshop on Adaptive and Reflective Middleware, ARM@Middleware 2014, Bordeaux, France, December 8-12, 2014*, pages 3:1–3:6. ACM, 2014.

[9] Carsten Lutz. The complexity of conjunctive query answering in expressive description logics. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings*, volume 5195 of *Lecture Notes in Computer Science*, pages 179–193. Springer, 2008.

[10] Nhung Ngo, Magdalena Ortiz, and Mantas Simkus. Closed predicates in description logics: Results on combined complexity. In Chitta Baral, James P. Delgrande, and Frank Wolter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016.*, pages 237–246. AAAI Press, 2016.

[11] Sebastian Rudolph and Birte Glimm. Nominals, inverses, counting, and conjunctive queries or: Why infinity is your friend! *CoRR*, abs/1401.3849, 2014.

[12] Klaus Schild. A correspondence theory for terminological logics: Preliminary report. In John Mylopoulos and Raymond Reiter, editors, *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI 1991)*, pages 466–471. Morgan Kaufmann, 1991.

[13] Stephan Tobies. *Complexity results and practical algorithms for logics in knowledge representation*. PhD thesis, RWTH Aachen University, Germany, 2001.

[14] Benjamin Zarrieß. *Verification of Golog programs over description logic actions*. PhD thesis, Dresden University of Technology, Germany, 2018.