Query Answering over Contextualized RDF/OWL Knowledge with Forall-Existential Bridge Rules: Decidable Finite Extension Classes

Mathew Joseph a,b,*, Gabriel Kuper b, Till Mossakowski c, Luciano Serafini a

a DKM, FBK-IRST, Via Sommarive 18, 38050, Trento, Italy
b DISI, University Of Trento, Via Sommarive 5, 38123, Trento, Italy
c Faculty of Computer Science, Otto-von-Guericke University of Magdeburg, 39016, Magdeburg, Germany
E-mail: {joseph, kuper}@disi.unitn.it, {serafini}@fbk.eu, {mossakow}@iws.cs.uni-magdeburg.de

Abstract. The proliferation of contextualized knowledge in the Semantic Web (SW) has led to the popularity of knowledge formats such as quads in the SW community. A quad is an extension of RDF triple with the contextual information of the triple. We in this paper, study the problem of query answering over quads augmented with forall-existential bridge rules that enable interoperability of reasoning between the triples in various contexts. We call a set of quads together with such expressive bridge rules, a quad-system. Query answering over quad-systems is undecidable, in general. We derive decidable classes of quad-systems, for which query answering can be done using forward chaining. Sound, complete and terminating procedures, which are adaptations of the well known chase algorithm, are provided for these classes for deciding query entailment. Safe, msafe, and csafe class of quad-systems restrict the structure of blank nodes generated during the chase computation process to be directed acyclic graphs (DAGs) of bounded depth. RR and restricted RR classes do not allow the generation of blank nodes during the chase computation process. Both data and combined complexity of query entailment has been established for the classes derived. We further show that quad-systems are equivalent to forall-existential rules whose predicates are restricted to ternary arity, modulo polynomial time translations. We subsequently show that the technique of safety, strictly subsumes in expressivity, some of the well known and expressive techniques, such as joint acyclicity and model faithful acyclicity, used for decidability guarantees in the realm of forall-existential rules.

Keywords: Contextualized Query Answering, Contextualized RDF/OWL knowledge bases, Multi-Context Systems, Quads, Query answering, forall-existential rules, Knowledge Representation, Semantic Web

1. Introduction

As the Semantic Web (SW) is getting more and more ubiquitous and its constellation of interlinked ontologies, the web of data, is seamlessly proliferating at a steady rate, more and more applications have started using SW as a back end, providing their users manifold services, leveraging semantic technologies. One of the main reasons why SW enjoys such admirable hospitality from its mammoth geographically disparate users is its “simple” and “open” model. The model is simple, as the only intricacy that a creator/consumer of a SW
application needs to equipped with, is that of a (RDF) triple. A triple \( t = (s, p, o) \) represents the most basic piece of knowledge in SW, where \( s \) called the subject, is an identifier for a person, place, thing, value, or a resource in general, about which the creator of \( t \) intended to express his/her knowledge using \( t \). \( p \) called the predicate, is an identifier for a property, attribute, or in general a relation that relates \( s \) with the third component \( o \), called the object, that is also an identifier for a resource, similar to \( s \). The model is called open, as it allows anybody, anywhere around the world to freely create their RDF/OWL ontologies about a domain of their choice, and publish them in (embedded) RDF/OWL formats in their web portals, also linking via URLs to the concepts in other similarly published ontologies. Thus, the open model in order to promote reuse and freedom, imposes no arbitration mechanism for the ontologies users publish on the SW.

On the other hand, a problem caused by this open model is that an ontology which a person publishes is often his/her own perspective about a particular domain, which largely is relative to this person. As a consequence, the truth value of a piece of knowledge in the SW is context-dependent. Recently, as a solution to the aforementioned problem, SW community, adopts the use of a quadruple, an extension of a triple, as the primary carrier of knowledge. A quad \( c: (s, p, o, c) \), thus adds the fourth component of the context \( c \) to the triple \( (s, p, o) \), thus explicating the identifier of the context in which the triple holds. As a result, more and more triple stores are becoming quad-stores. Some of the popular quad-stores are 4store\(^1\), Openlink Virtuoso\(^2\), and some of the current popular triple stores like Sesame\(^3\), Allegrograph\(^4\) internally keep track of the contexts of triples. Some of the recent initiatives in this direction have also extended existing formats like N-Triples to N-Quads, which the RDF 1.1 has introduced as a W3C recommendation. The latest Billion triple challenge datasets have been all released in the N-Quads format.

Another benefit of quadruples over triples are that they allow knowledge creators to specify various attributes of meta-knowledge that further qualify knowledge \([2]\), and also allow users to query for this meta-knowledge \([3]\). These attributes, which explicate the various assumptions under which knowledge holds, are also called context dimensions \([4]\). Examples of context dimensions are provenance, creator, intended user, creation time, validity time, geo-location, and topic. Having defined knowledge that is contextualized, as in \( c_1: (\text{Renzi}, \text{primeMinisterOf}, \text{Italy}) \), one can now declare in a meta-context \( mc \), statements such as \( mc: (c_1, \text{creator}, \text{John}), mc: (c_1, \text{expiryTime}, \text{"jun-2016"}) \) that talk about the knowledge in context \( c_1 \), in this case its creator and expiry time. Another benefit of such a contextualized approach is that it opens possibilities of interesting ways for querying a contextualized knowledge base. For instance, if context \( c_1 \) contains knowledge about football world cup 2014 and context \( c_2 \) about football euro cup 2012. Then the query “who beat Italy in both world cup 2014 and euro cup 2012” can be formalized as the conjunctive query:

\[
c_1: (x, \text{beat}, \text{Italy}) \land c_2: (x, \text{beat}, \text{Italy}),
\]

where \( x \) is a variable.

While reasoning with knowledge in quad form, since knowledge can be grouped and divided context wise and simultaneously be fed to separate reasoning engines, this approach improves both efficiency and scalability. Besides the above flexibility, bridge rules \([5]\) can be provided for inter-operating the knowledge in different contexts. Such rules are primarily of the form:

\[
c : \phi \rightarrow c' : \phi'
\]

where \( \phi, \phi' \) are both atomic concept (role) symbols, \( c, c' \) are contexts. The semantics of such a rule is that if, for any \( \vec{a}, \phi(\vec{a}) \) holds in context \( c \), then \( \phi'(\vec{a}) \) should hold in context \( c' \), where \( \vec{a} \) is a unary/binary vector depending on whether \( \phi, \phi' \) are concept/role symbols. Although such bridge rules serve the purpose of specifying knowledge interoperability from a source context \( c \) to a target context \( c' \), in many practical situations there is the need of inter-operating multiple source contexts with multiple target targets, for which the bridge rules of the form (1) is inadequate. Besides, one would also want the ability of creating new values in target contexts for the bridge rules.

In this work, we study contextual reasoning and query answering over contextualized RDF/OWL knowledge in the presence of forall-existential bridge rules that allows conjunctions and existential quantifiers in them, and hence is more expressive than those, in DDL \([5]\) and McCarthy et al. \([6]\). We provide a basic semantics for contextual reasoning based on which we provide procedures for conjunctive query answering. For query answering, we use the notion of a
distributed chase, which is an extension of the standard chase [19,20] that is widely used in the knowledge representation (KR) and Database (DB) settings, for similar purposes. As far as the semantics for reasoning is concerned, we adopt the approach given in works such as Distributed Description Logics [5], E-connections [21], and two-dimensional logic of contexts [22], to use a set of interpretation structures as a model for contextualized knowledge. In this way, knowledge in each context is separately interpreted to a different interpretation structure. The main contributions of this work are:

1. We extend the standard RDF/OWL semantics to a context-based semantics that can be used for reasoning over contextualized RDF/OWL knowledge. Studying conjunctive query answering over quad-systems. It turns out that the entailment problem of conjunctive queries is undecidable for the most general class of quad-systems, called unrestricted quad-systems.

2. We derive decidable subclasses of unrestricted quad-systems, namely \(c\text{safe}, \text{m safe}\), and \(\text{safe}\) quad-systems, for which we detail both data and combined complexities of conjunctive query entailment. The classes are based on constrained DAG structure of Skolem blank nodes generated during the chase construction. We also provide decision procedures to decide whether an input quad-system is safe (\(c\text{safe}, \text{m safe}\)) or not.

3. We further derive less expressive classes, \(\text{RR}\) and restricted \(\text{RR}\) quad-systems, for which no Skolem blank nodes are generated during chase construction.

4. We show that the class of unrestricted quad-systems are equivalent to the class of ternary \(\forall\exists\) rule sets. We compare the derived classes of quad-systems with well known subclasses of \(\forall\exists\) rule sets, such as jointly acyclic and model faithful acyclic rule sets, and show that the technique of safety, we propose, subsumes these other techniques, in expressivity.

The paper is structured as follows. In section 2, we formalize the notion of contextualized quad-systems, giving various definitions and notations for setting the background. In section 3, we formalize the problem of query answering on quad-systems, define notions such as distributed chase that is further used for query answering, and give the undecidability results of query entailment on unrestricted quad-systems. In section 4, we present \(c\text{safe}, \text{m safe}, \text{safe}\) quad-systems and their computational properties. In section 5, the RR quad-systems and the restricted RR quad-systems. In section 6, we prove the equivalence of quad-systems with ternary \(\forall\exists\) rule sets, and formally compare a few well known decidable classes in the realm of \(\forall\exists\) rules to the classes of quad-systems, we presented in section 4. We provide a detailed discussion to other relevant related works in section 7, and conclude in section 8.

Note that parts of the contents of section 2 and section 3 has been taken from conference papers [10] and [11].

2. Contextualized Quad-Systems

In this section, we formalize the notion of a quad-system and its semantics. For any vector or sequence \(\vec{x}\), we denote by \(|\vec{x}|\) the number of symbols in \(\vec{x}\), and by \(\{\vec{x}\}\) the set of symbols in \(\vec{x}\). For any sets \(A\) and \(B\), \(A \rightarrow B\) denotes the set of all functions from set \(A\) to set \(B\). Given the set of URIs \(U\), the set of blank nodes \(B\), and the set of literals \(L\), the set \(C = U \uplus B \uplus L\) is called the set of (RDF) constants. Any \((s, p, o) \in C \times C \times C\) is called a generalized RDF triple (from now on, just triple). A graph is defined as a set of triples. A Quad is a tuple of the form \((c, (s, p, o))\), where \((s, p, o)\) is a triple and \(c\) is a URI\(^5\), called the context identifier that denotes the context of the RDF triple. A quad-graph is defined as a set of quads. For any quad-graph \(Q\) and any context identifier \(c\), we denote by \(\text{graph}_Q(c)\) the set \(\{(s, p, o) | c: (s, p, o) \in Q\}\). We denote by \(Q_C\) the quad-graph whose set of context identifiers is \(C\). The set of constants occurring in \(Q_C\), given as \(C(Q_C) = \{c, s, p, o | c: (s, p, o) \in Q_C\}\). The set of URIs in \(Q_C\), is given by \(U(Q_C) = C(Q_C) \cap U\). The set of blank nodes \(B(Q_C)\), the set of literals \(L(Q_C)\) are similarly defined. Let \(V\) be the set of variables, any element of the set \(V\) is a term. Any \((s, p, o) \in V \times V \times V\) is called a triple pattern, and an expression of the form \(c: (s, p, o)\), where \((s, p, o)\) is a triple pattern, \(c\) a context identifier, is called a quad pattern. A triple pattern \(t\), whose variables are elements of the vector \(\vec{x}\) or elements of the vector \(\vec{y}\) is written as \(t(\vec{x}, \vec{y})\). For any function \(f: A \rightarrow B\), the restriction of \(f\) to a set \(A'\), is the mapping \(f|_{A'}\) from \(A' \cap A\) to \(B\) s.t. \(f|_{A'}(a) = f(a)\), for each \(a \in A \cap A'\). For any triple pattern \(t = (s, p, o)\) and a function \(\mu\) from \(V\) to a set \(A\), \(t[\mu]\) denotes

\(^5\)Although, in general a context identifier can be a constant, for the ease of notation, we restrict them to be a URI
(µ'(s), µ'(p), µ'(o)), where µ' is an extension of µ to C s.t. µ'(c) is the identity function. For any set of triple patterns G, G'[|µ|] denotes ∪_{t∈G} t[|µ|]. For any vector of constants a = ⟨a_1,...,a_l⟩, and vector of variables x of the same length, x/a is the function µ s.t. µ(x_i) = a_i, for 1 ≤ i ≤ |a|. We use the notation t(a, y) to denote t(x, y)[x/a]. Similarly, the above notations are also extended to sets of quad-patterns. For instance Q(x, y) denotes a set of quad-patterns, whose variables are from x or y, and Q(a, y) is written for Q(x, y)[x/a].

For the sake of interoperating knowledge in different contexts, bridge rules need to be provided:

Bridge rules (BRs) Formally, a BR is of the form:
\[ ∀x, y [ c_1 : t_1(x, z) ∧ ... ∧ c_n : t_n(x, z) \rightarrow ∃y \hat{c}_1 : t'_1(x, \hat{y}) ∧ ... ∧ \hat{c}_m : t'_m(x, \hat{y}) ] \] (2)

where c_1,...,c_n, \hat{c}_1,...,\hat{c}_m are context identifiers, x, y, z are vectors of variables s.t. \{x\}, \{y\}, and \{z\} are pairwise disjoint. t_1(x, z), ..., t_n(x, z) are triple patterns which do not contain blank-nodes, and whose set of variables are from x or y, and Q(x, y) is written for Q(x, y)[x/a].

For any BR r of the form (2), body(r) is the set of quadr patterns \{c_1 : t_1(x, z),...,c_n : t_n(x, z)\}, and head(r) is the set of quadr patterns \{\hat{c}_1 : t'_1(x, \hat{y}), ... ,\hat{c}_m : t'_m(x, \hat{y})\}, and the frontier of r, fr(r) = \{x\}. Occasionally, we also note the BR r above as body(r)(x, z) → head(r)(x, \hat{y}). The set of terms in a BR r is:
\[ CV(r) = \{c, s, p, o | c : (s, p, o) ∈ body(r)∪head(r)\} \]

The set of terms for a set of BRs R is \[ CV(R) = \bigcup_{r∈R} CV(r). \] The URIs, blank nodes, literals, variables of a BR r (resp. set of BRs R) are similarly defined, and are denoted as U(r), B(r), L(r), V(r) (resp. U(R), B(R), L(R), V(R)), respectively.

Definition 2.1 (Quad-System). A quad-system QS_C is defined as a pair \( QS_C = (Q_C, R) \), where Q_C is a quad-graph, whose set of context identifiers is C, and R is a set of BRs.

For any quad-system, QS_C = \( (Q_C, R) \), the set of constants in QS_C is given by C(QS_C) = C(Q_C) ∪ C(R). The sets U(QS_C), B(QS_C), L(QS_C), and V(QS_C) are similarly defined for any quad-system QS_C. For any quad-graph Q_C (BR r), its symbol size \|Q_C\| (\|r\|) is the number of symbols required to print Q_C (r). Hence, \|Q_C\| \approx 4 * |Q_C|, where |Q_C| denotes the cardinality of the set Q_C. Note that |Q_C| equals the number of quads in Q_C. For a BR r, \|r\| \approx 4 + k, where k is the number of quad-patterns in r. For a set of BRs R, \|R\| is given as \( Σ_{r∈R} \|r\| \). For any quad-system QS_C = \( (Q_C, R) \), its size \|QS_C\| = \|Q_C\| + \|R\|.

Semantics In order to provide a semantics for enabling reasoning over a quad-system, we need to use a local semantics for each context to interpret the knowledge pertaining to it. Since the primary goal of this paper is a decision procedure for query answering over quad-systems based on forward chaining, we consider the following desiderata for the choice of the local semantics and its deductive machinery:

- there exists a set LIR of inference rules and an operation \( lclosure() \) that computes the deductive closure of a graph w.r.t. to the local semantics using the inference rules in LIR,
- each inference rule in LIR is range restricted, i.e. non value-generating,
- given a finite graph as input, the \( lclosure() \) operation, terminates with a finite graph as output in polynomial time whose size is polynomial w.r.t. to the input set.

Some of the alternatives for the local semantics satisfying the above mentioned criterion are Simple, RDF, RDFS [30], OWL-Horst [26] etc. Assuming that a local semantics has been fixed, for any context c, we denote by \( I^c = (Ω^c_1, τ^c_1) \) an interpretation structure for the local semantics, where \( Ω^c \) is the interpretation domain, \( τ^c_1 \) the corresponding interpretation function. Also \( |=_\text{local} \) denotes the local satisfaction relation between a local interpretation structure and a graph. Given a quad graph Q_C, a distributed interpretation structure is an indexed set \( IC = \{I^c\}_{c∈C} \), where I^c is a local interpretation structure, for each c ∈ C. We define the satisfaction relation \( |= \) between a distributed interpretation structure \( IC \) and a quad-system QS_C as:

Definition 2.2 (Model of a Quad-System). A distributed interpretation structure \( IC = \{I^c\}_{c∈C} \) satisfies a quad-system QS_C = \( (Q_C, R) \), in symbols \( IC |= QS_C \), iff all the following conditions are satisfied:
1. \( I^c |=_\text{local} \text{graph}_C(c) \), for each c ∈ C;
2. a^c_α = a^c_β, for any a ∈ C, c_1, c_2 ∈ C;
3. for each BR r ∈ R of the form (2) and for each \( σ ∈ V \rightarrow Δ^c \), where \( Δ^c = \bigcup_{r∈R} Δ^r \), if
\[ I^c |=_\text{local} t_1(\bar{x}, \bar{z})[σ],...,I^c |=_\text{local} t_n(\bar{x}, \bar{z})[σ], \]
then there exists function \( σ' \supseteq σ \), s.t.
$I^c \models_{\text{local}} t^c_{i}(\vec{x}, \vec{y})[\sigma^c_i], \ldots, I^c_m \models_{\text{local}} t^c_m(\vec{x}, \vec{y})[\sigma^c_m]$

Condition 1 in the above definition ensures that for any model $I^c$ of a quad-graph, each $I^c \in I^c$ is a local model of the set of triples in context $c$. Condition 2 ensures that for any $I^c$ s.t. $I^c \models QS_C$, then $I^c \models QS_C$, and otherwise said to be inconsistent. For any quad-system $QS_C = (Q_C, R_S)$, it can be the case that $graph_{Q_C}(c)$ is locally consistent, i.e. there exists an $I^c$ s.t. $I^c \models \text{local graph}_{Q_C}(c)$, for each $c \in C$, whereas $QS_C$ is not consistent. This is because the set of BRs $R$ adds more knowledge to the quad-system, and restricts the set of models that satisfy the quad-system.

**Definition 2.3** (Quad-system entailment). (a) A quad-system $QS_C$ entails a quad $c$: $(s, p, o)$, in symbols $QS_C \models c: (s, p, o)$, iff for any distributed interpretation structure $I^c$, if $I^c \models QS_C$ then $I^c \models \langle \{c: (s, p, o)\} \cup \emptyset \rangle$. (b) A quad-system $QS_C$ entails a quad-graph $Q_C'$, in symbols $QS_C \models Q_C'$, iff $QS_C \models c: (s, p, o)$ for any $c: (s, p, o) \in Q_C'$. (c) A quad-system $QS_C$ entails a BR $r$ iff for any $I^c$, if $I^c \models QS_C$ then $I^c \models \emptyset \cup \{r\}$. (d) For a set of BRs $R$, $QS_C \models R$ iff $QS_C \models r$, for every $r \in R$. (e) Finally, a quad-system $QS_C$ entails another quad-system $QS_C'$, = $(Q_C', R')$, in symbols $QS_C \models QS_C'$, iff $QS_C \models Q_C'$, and $QS_C \models R'$.

We call the decision problems (DPs) corresponding to the entailment problems (EPs) in (a), (b), (c), (d), and (e) as quad EP, quad-graph EP, BR EP, BRs EP, and quad-system EP, respectively.

3. Query Answering on Quad-Systems

In the realm of quad-systems, the classical conjunctive queries or select-project-join queries are slightly extended to what we call Contextualized Conjunctive Queries (CCQs). A CCQ $CQ(\vec{x})$ is an expression of the form:

$$\exists \vec{y} \ q_1(\vec{x}, \vec{y}) \land \ldots \land q_p(\vec{x}, \vec{y})$$

where $q_i$, for $i = 1, \ldots, p$ are quad patterns over vectors of free variables $\vec{x}$ and quantified variables $\vec{y}$. A CCQ is called a boolean CCQ if it does not have any free variables. With some abuse, we sometimes discard the logical symbols in a CCQ and consider it as a set of quad-patterns. For any CCQ $CQ(\vec{x})$ and a vector $\vec{a}$ of constants s.t. $\|\vec{a}\| = \|\vec{y}\|$, $CQ(\vec{a})$ is boolean. A vector $\vec{a}$ is an answer for a CCQ $CQ(\vec{x})$ w.r.t. structure $I^c$, in symbols $I^c \models CQ(\vec{a})$, iff there exists assignment $\mu: \{\vec{y}\} \rightarrow B$ s.t. $I^c \models \bigcup_{i=1}^{p} q_i(\vec{a}, \vec{y})[\mu]$. A vector $\vec{a}$ is a certain answer for a CCQ $CQ(\vec{x})$ over a quad-system $QS_C$, iff $I^c \models CQ(\vec{a})$, for every model $I^c$ of $QS_C$. Given a quad-system $QS_C$, a CCQ $CQ(\vec{x})$, and a vector $\vec{a}$, DP of determining whether $QS_C \models CQ(\vec{a})$ is called the CCQ EP. It can be noted that the other DPs over quad-systems, namely Quad/Quad-graph EP, BR(s) EP, Quad-system EP, are reducible to the CCQ EP (See property 6.6). Hence, in this paper, we primarily focus on the CCQ EP.

3.1. dChase of a Quad-System

In order to build a procedure for query answering over a quad-system, we employ what has been called in the literature, a chase [19,20]. Specifically, we adopt notions of the restricted chase in Fagin et al. [23] (also called non-oblivious chase). In order to fit the framework of quad-systems, we extend the standard notion of chase to a distributed chase, abbreviated $dChase$. In the following, we show how the $dChase$ of a quad-system can be constructed.

For a set of quad-patterns $S$ and a set of terms $T$, we define the relation $T$-connectedness between quad-patterns in $S$ as follows:

- $q_1$ and $q_2$ are $T$-connected, if $C^V(q_1) \cap C^V(q_2) \cap T \neq \emptyset$, for any two quad-patterns $q_1, q_2 \in S$.
- if $q_1$ and $q_2$ are $T$-connected, and $q_3$ and $q_4$ are $T$-connected, then $q_1$ and $q_3$ are also $T$-connected, for any quad-patterns $q_1, q_2, q_3 \in S$.

It can be noted that $T$-connectedness is an equivalence relation and partitions $S$ into a set of $T$-components (similar notion is called a piece in Baget et al. [14]). Note that for two distinct $T$-components $P_1, P_2$ of $S$, $C^V(P_1) \cap C^V(P_2) \cap T = \emptyset$. For any BR $r = \text{body}(r)(\vec{x}, \vec{z}) \rightarrow \text{head}(r)(\vec{x}, \vec{y})$, suppose $P_1, P_2, \ldots, P_k$ are the pairwise distinct $\{\vec{y}\}$-components of $\text{head}(r)(\vec{x}, \vec{y})$, then $r$ can be replaced by the semantically equivalent set of BRs $\{\text{body}(r)(\vec{x}, \vec{z}) \rightarrow P_1, \ldots, \text{body}(r)(\vec{x}, \vec{z}) \rightarrow P_k\}$ whose symbol size is worst case quadratic w.r.t. the symbol size of $r$. Hence, w.l.o.g. we assume that for any BR $r$, the set of quad-patterns $\text{head}(r)$ is a single component w.r.t. the set of existentially quantified variables in $r$. 

Joseph et al. / Query Answering over Contextualized RDF/OWL Knowledge with Forall-Existential Bridge Rules ...
Considering the fact that the local semantics for contexts are fixed a priori (for instance RDFS), both the number of rules in the set of local inference rules LIR and the size of each rule in LIR can be assumed to be a constant. Note that each local inference rule is range restricted and do not contain existentially quantified variables in its head. Any $ir \in LIR$ is of the form:

$$\forall \vec{x}\vec{z} [t_1(\vec{x},\vec{z}) \land \ldots \land t_k(\vec{x},\vec{z}) \rightarrow t'_1(\vec{x})],$$

where $t_i(\vec{x},\vec{z})$, for $i = 1, \ldots, n$ are triple patterns, whose variables are from $\{\vec{x}\}$ or $\{\vec{z}\}$, and $t'_1(\vec{x})$ is a triple pattern, whose variables are from $\{\vec{x}\}$. Hence, for any quad-system $QS_C = \langle C, R \rangle$ in order to accomplish the effect of local inferencing in each context $c \in \mathcal{C}$, for each $ir \in LIR$ of the form (4), we could augment $R$ with the form:

$$\forall \vec{x}\vec{z} \exists c: t_1(\vec{x},\vec{z}) \land \ldots \land c: t_k(\vec{x},\vec{z}) \rightarrow c: t'_1(\vec{x})]$$

Since $||LIR||$ is a constant and the size of the augmentation is linear in $|\mathcal{C}|$, w.l.o.g we assume that the set $R$ contains a BR $ir_c$, for each $ir \in LIR, c \in \mathcal{C}$.

Given a quad-system $QS_C$, we denote by $B_{sk} \subseteq B$, a set of blank nodes with unique node ids called Skolem blank nodes, s.t. $B_{sk} \cap B(QS_C) = \emptyset$. For any BR $r = body(r)(\vec{x},\vec{z}) \rightarrow head(r)(\vec{x},\vec{y})$ and an assignment $\mu$: $\{\vec{x}\} \cup \{\vec{z}\} \rightarrow \mathcal{C}$, the application of $\mu$ on $r$ is defined as:

$$apply(r, \mu) = head(r)[\mu^{ext}(\vec{y})]$$

where $\mu^{ext}(\vec{y}) \supseteq \mu$ s.t. $\mu^{ext}(\vec{y})(y_i) = _i b$ is a fresh blank node from $B_{sk}$, for each $y_i \in \{\vec{y}\}$.

We assume that there exists an order $\prec_l$ (for instance, lexicographic order) on the set of constants. We extend $\prec_l$ to the set of quads s.t. for any two quads $c$: $(s, p, o)$ and $c': (s', p', o')$, $c: (s, p, o) \prec_l c': (s', p', o')$, iff $c \prec_l c'$, or $c = c'$, $s \prec_l s'$, or $c = c', s = s', p \prec_l p'$, or $c = c', s = s', p = p', o \prec_l o'$. It can be noted that $\prec_l$ is a strict linear order over the set of all quads. For any quad-graph $QC$, $\prec_l$-greatest quad of $QC$, denoted $\text{greatestQuad}_{\prec_l}(QC)$, is the quad $q \in QC$ s.t. $q' \prec_l q$, for every other $q' \in QC$. Also, the order $\prec_l$ is defined over the set of quad-graphs as follows: for any two quad-graphs $QC, QC'$:

$$QC \prec_l QC', \text{ if (i) } QC \subset QC';$$
$$QC \prec_l QC', \text{ if (i) does not hold and (ii) greatestQuad}_{\prec_l}(QC \setminus QC') \prec_l \text{greatestQuad}_{\prec_l}(QC' \setminus QC);$$
$$QC \not\prec_l QC', \text{ if both (i) and (ii) are not satisfied;}$$

A relation $R$ over a set $A$ is called a strict linear order iff $R$ is irreflexive, transitive, and $R(a, b) \lor R(b, a)$ holds, for every distinct $a, b \in A$.

**Property 3.1.** Let $Q$ be the set of all quad-graphs; $\prec_l$ is a strict linear order over $Q$.

Also, we define the level of a quad in dChase of a quad-system $QS_C = \langle C, R \rangle$ as follows: any quad in $QC$ is of level 0. The level of a set of quads is the largest among levels of quads in the set. Level of any quad that results from the application of a BR $r$ w.r.t. an assignment $\mu$ is one more than the level of the set $\text{body}(r)[\mu]$, if it has already not been assigned a level. Let $\prec$ be an ordering on the quad-graphs s.t. for any two quad-graphs $QC'$ and $QC''$ of the same level, $QC' \prec QC''$, iff $QC' \prec QC''$. For $QC'$ and $QC''$ of different levels, $QC' \prec QC''$, iff level of $QC'$ is less than level of $QC''$. It can easily be seen that $\prec$ is a strict linear order over the set of quad-graphs. For any BRs $r, r'$ and assignments $\mu, \mu'$ over $V(\text{body}(r)), V(\text{body}(r'))$, respectively, $(r, \mu) \prec (r', \mu')$ iff $\text{body}(r)[\mu] \prec \text{body}(r')[\mu']$.

For any quad-graph $QC$, a set of BRs $R$, a BR $r \in R$, an assignment $\mu \in V(\text{body}(r)) \rightarrow \mathcal{C}$, the boolean function $\text{applicable}(r, \mu, QC)$ is defined as:

$$\text{True, if (a) } body(r)[\mu] \subset QC', \text{head}(r)[\mu'] \not\subset QC', $$
$$\forall \mu'' \supseteq \mu, \text{ and (b) } \forall r' \in R, \exists \mu' \text{s.t. } r' \neq r \text{ or }$$
$$\mu' \not\equiv \mu \text{ with } (r', \mu') \prec (r, \mu) \text{ and applicable}_{R}(r', \mu', QC');$$
$$\text{False, otherwise;}$$

For any quad-system $QS_C = \langle C, R \rangle$, let

$$d\text{Chase}_i(QS_C) = QC;$$
$$d\text{Chase}_{i+1}(QC) = d\text{Chase}_i(QS_C) \cup \text{apply}(r, \mu), \text{ if there exists } r = body(r)(\vec{x},\vec{z}) \rightarrow head(r)(\vec{x},\vec{y}) \in R, \text{ assignment } \mu: \{\vec{x}\} \cup \{\vec{z}\} \rightarrow \mathcal{C} \text{ s.t. } \text{applicable}(r, \mu, QC);$$
$$d\text{Chase}_{i+1}(QS_C) = d\text{Chase}_i(QS_C), \text{ otherwise; for any } i \in \mathbb{N}.$$

The dChase of $QS_C$, noted $d\text{Chase}(QS_C)$, is given as:

$$d\text{Chase}(QS_C) = \bigcup_{i \in \mathbb{N}} d\text{Chase}_i(QS_C)$$

Intuitively, $d\text{Chase}_i(QS_C)$ can be thought of as the state of $d\text{Chase}(QS_C)$ at the end of iteration $i$. It can be noted that, if there exists $i$ s.t. $d\text{Chase}_i(QS_C) = d\text{Chase}_{i+1}(QS_C)$, then $d\text{Chase}(QS_C) = d\text{Chase}_i(QS_C)$. The dChase $d\text{Chase}(QS_C)$ of a consistent quad-system $QS_C$ is a universal model [29] of the quad-system, i.e. it is a model of $QS_C$, and for any model $\mathcal{I}$ of $QS_C$, there is a homomorphism from $d\text{Chase}(QS_C)$ to $\mathcal{I}$. Hence, for any boolean CCQ $CQ()$, $QS_C \models CQ()$ iff there exists a map
\( \mu: V(CQ) \rightarrow C \) s.t. \( \{CQ\} \{\mu\} \subseteq d\text{Chase}(QS_C) \). We call the sequence \( d\text{Chase}_0(QS_C), d\text{Chase}_1(QS_C) \), ..., the \( d\text{Chase} \) sequence of \( QS_C \). The following lemma shows that in a \( d\text{Chase} \) sequence of a quad-system, any \( d\text{Chase} \) iteration can be performed in time exponential w.r.t. the size of the largest BR.

**Lemma 3.2.** For a quad-system \( QS_C = (Q_C, R) \), for any \( i \in \mathbb{N}^+ \), the following holds: (i) \( d\text{Chase}_i(QS_C) \) can be computed in time \( O(\|R\|^{|d\text{Chase}_{i-1}(QS_C)|}^{\|s\|}) \), where \( s = \max_{r \in R}[r] \). (ii) \|d\text{Chase}_i(QS_C)\| = O(\|d\text{Chase}_{i-1}(QS_C)\| + ||R||).

**Proof.** (i) We can first find, if there exists an \( r \in R \), assignment \( \mu \) s.t. \( \text{applicable}_R(r, \mu, R, d\text{Chase}_{i-1}(QS_C)) \) holds, in the following naive way: (1) bind the set of variables in all rules in \( R \) with the set of constants in \( d\text{Chase}_{i-1}(QS_C) \). Let this set be called \( S \). Note that \( |S| = O(|R| \times |d\text{Chase}_{i-1}(QS_C)|^{||s||}) \), where \( s = \max_{r \in R}[r] \). Also, note that each of the binding in \( S \) is of the form \( \text{body}_r(\vec{x}, \vec{y})(\mu) \rightarrow \text{head}_r(\vec{x}, \vec{y})(\mu) \) (\( C \)), where \( r \in R \). (2) From the set \( S \) we filter out every binding of the form (\( C \)) in which \( \vec{x}[\mu] \neq \vec{x}[\mu'] \). Let \( S' \) be the resulting set after the above filtering operation. (3) From the set \( S' \), we now filter out all the bindings of the form (\( C \)) with \( \text{head}_r(\vec{x}, \vec{y})(\mu') \subseteq d\text{Chase}_{i-1}(QS_C) \), with resulting set \( S'' \). (4) If \( S'' = \emptyset \), then there is no \( r \in R \), assignment \( \mu \) s.t. \( \text{applicable}_R(r, \mu, d\text{Chase}_{i-1}(QS_C)) \) is True. Otherwise if \( S'' \neq \emptyset \), then note that each binding of the form (\( C \)) in \( S'' \) is s.t. condition (a) of the true \( \text{applicable}_R(r, \mu, d\text{Chase}_{i-1}(QS_C)) \) is satisfied. Now, we can sort \( S'' \) w.r.t. \( \prec \) and select the least binding \( b \) of the form (\( C \)), so that condition (b) in True condition of \( \text{applicable}_R \) is satisfied for \( b \). It can easily seen that \( \text{applicable}_R(r, \mu, d\text{Chase}_{i-1}(QS_C)) \) holds for the \( r, \mu \) extracted from \( b \). Since, the size of each binding is at most \( ||s|| \), the operations (1)-(4) can be performed in time \( O(|R| \times |d\text{Chase}_{i-1}(QS_C)|^{||s||}) \). Since \( d\text{Chase}_i(QS_C) = d\text{Chase}_{i-1}(QS_C) \cup \text{head}_r(\mu) \), for \( r, \mu \) with \( \text{applicable}_R(r, \mu, d\text{Chase}_{i-1}(QS_C)) \), \( d\text{Chase}_i(QS_C) \) can be computed in time \( O(|d\text{Chase}_{i-1}(QS_C)|^{||s||}) \).

(ii) Trivially holds, since in the worst case \( d\text{Chase}_i(QS_C) = d\text{Chase}_{i-1}(QS_C) \cup \text{head}_r(\mu) \), for \( r \in R \).

**Lemma 3.3.** For any quad-system \( QS_C \), If \( \_ : b \) is a Skolem blank node in \( d\text{Chase}(QS_C) \), generated by the application of assignment \( \mu \) on \( r = \text{body}_r(\vec{x}, \vec{z}) \rightarrow \text{head}_r(\vec{x}, \vec{y}) \), with \( \mu^{\text{ext}(\vec{y})}(y_j) = \_ : b, y_j \in \{\vec{y}\} \), then \( \_ : b \) is unique for \( (r, y_j, x[\mu^{\text{ext}(\vec{y})}]) \).

**Proof.** By contradiction, suppose if \( \_ : b \) is not unique for \( (r, y_j, x[\mu^{\text{ext}(\vec{y})}]) \), i.e. there exists \( \_ : b' \neq \_ : b \) in \( d\text{Chase}(QS_C) \), with \( \_ : b' \) generated by \( r \) s.t. \( \_ : b' = \mu^{\text{ext}(\vec{y})}(y_j) \) and \( x[\mu^{\text{ext}(\vec{y})}] = x[\mu^{\text{ext}(\vec{y})}] \). W.l.o.g. suppose \( \_ : b \) was generated in an iteration \( l \in \mathbb{N} \) and \( \_ : b' \) in an iteration \( m > l \). This means that \( \text{head}(r)(\vec{x}, \vec{y})[\mu^{\text{ext}(\vec{y})}] \subseteq d\text{Chase}_l(QS_C) \) and hence, \( \text{head}(r)(\vec{x}, \vec{y})[\mu^{\text{ext}(\vec{y})}] \subseteq d\text{Chase}_{m-1}(QS_C) \). This means that \( \text{applicable}_R(r, \mu', d\text{Chase}_{m-1}(QS_C)) \) is false, as \( \mu' = \mu \), and our assumption that \( \_ : b' = y_j[\mu^{\text{ext}(\vec{y})}] \) is false. Hence, \( \_ : b \) is unique for \( (r, y_j, x[\mu^{\text{ext}(\vec{y})}]) \).

Although, we now know how to compute the \( d\text{Chase} \) of a quad-system, which can be used for deciding CCQ EP, the following proposition reveals that for the class of quad-systems whose BRs are of the form (2), which we call unrestricted quad-systems, the \( d\text{Chase} \) can be infinite.

**Proposition 3.4.** There exists unrestricted quad-systems whose \( d\text{Chase} \) is infinite.

**Proof.** Consider an example of a quad-system \( QS_C = (Q_C, r) \), where \( Q_C = \{c: (a, \text{rdf:type, } C), \} \), and the BR \( r = \{x, \text{ rdf:type, } C \} \rightarrow \exists y c: (x, P, y), c: (y, \text{ rdf:type, } C) \). The \( d\text{Chase} \) computation starts with \( d\text{Chase}_0(QS_C) = \{c: (a, \text{ rdf:type, } C), \} \), now the rule \( r \) is applicable, and its application leads to \( d\text{Chase}_1(QS_C) = \{c: (a, \text{ rdf:type, } C), c: (a, P: _ : b_1), c: (_ : b_1, \text{ rdf:type, } C) \} \), where \( _ : b_1 \) is a fresh Skolem blank node. It can be noted that \( r \) is yet again applicable on \( d\text{Chase}_1(QS_C) \), for \( c: (_ : b_1, \text{ rdf:type, } C) \), which leads to the generation of another Skolem blank node, and so on. Hence, \( d\text{Chase}(QS_C) \) does not have a finite fix-point, and \( d\text{Chase}(QS_C) \) is infinite.

A class \( \mathcal{C} \) of quad-systems is called a finite extension class (FEC), iff every member \( QS_C \in \mathcal{C} \), \( d\text{Chase}(QS_C) \) is a finite set. Therefore, the class of unrestricted quad-systems is not a FEC. This raises the question if there are other approaches that can be used, for instance, a similar problem of non-finite chase is manifested in description logics (DLs) with value creation, due to the presence of existential quantifiers, whereas the approaches like the one in Glimm et al. [27] provides an algorithm for CQ entailment based on query rewriting. The theorem 3.5 below establishes the fact that the CCQ EP for unrestricted

**Theorem 3.5.**
quad-systems is undecidable. Despite this, the reader should note that the following undecidability result and its proof is only provided for the sake of self containedness, and we do not claim the undecidability theorem nor its proof to be a novel contribution, as we will show in section 6, ternary ∀∃ rule sets are polynomially reducible to unrestricted quad-systems. Hence, the undecidability results provided in Baget et al. [14] can trivially be applied in our setting to obtain the undecidability result for unrestricted quad-systems.

**Theorem 3.5.** The CCQ entailment problem over unrestricted quad-systems is undecidable.

**Proof.** (sketch) We show that the well known undecidable problem of non-emptiness of intersection of context-free grammars (CFGs) is reducible to the CCQ entailment problem. Given two CFGs, \( G_1 = \langle V_1, T, S_1, P_1 \rangle \) and \( G_2 = \langle V_2, T, S_2, P_2 \rangle \), where \( V_1, V_2 \) are the set of variables, \( T \subseteq T \cap (V_1 \cup V_2) = \emptyset \) is the set of terminals, \( S_1 \in V_1 \) is the start symbol of \( G_1 \), and \( P_1 \) are the set of PRs of the form \( v \rightarrow \bar{w} \), where \( v \in V, \bar{w} \) is a sequence of the form \( w_1...w_n \), where \( w_i \in V_1 \cup T \). Similarly \( S_2, P_2 \) is defined. Deciding whether the language generated by the grammars \( L(G_1) \) and \( L(G_2) \) have non-empty intersection is known to be undecidable [32].

Given two CFGs \( G_1 = \langle V_1, T, S_1, P_1 \rangle \) and \( G_2 = \langle V_2, T, S_2, P_2 \rangle \), we encode grammars \( G_1, G_2 \) into a quad-system \( QS_c = \langle Q_c, R \rangle \), with only a single context identifier \( c \). Each PR \( r = v \rightarrow \bar{w} \in P_1 \cup P_2 \), with \( \bar{w} = w_1w_2...w_n \), is encoded as a BR of the form: \( c: (x_1, w_1, x_2), c: (x_2, w_2, x_3), ..., c: (x_{n-1}, w_n, x_{n+1}) \rightarrow c: (x_1, v, x_{n+1}) \), where \( x_1, ..., x_{n+1} \) are variables. For each terminal symbol \( t_i \in T \), \( R \) contains a BR of the form: \( c: (x, rdf:type, C) \rightarrow \exists y \ c: (x, t_i, y), c: (y, rdf:type, C) \) and \( Q_c \) is the singleton: \( \{ c: (a, rdf:type, C) \} \). It can be observed that:

\[
QS_c \models \exists y \ c: (a, S_1, y) \land c: (a, S_2, y) \iff L(G_1) \cap L(G_2) \neq \emptyset
\]

We refer the reader to Appendix for the complete proof. 

---

4. **Safe, Msafe and Csafe Quad-Systems: Decidable FECs**

In the previous section, we saw that the query answering problem over unrestricted quad-systems is undecidable, in general. We will also see in section 6 that any quad-system is polynomially translatable to a ∀∃ rule set, which is also a first order logic theory. Hence, a possible solution approach is to translate to these more expressive languages, and apply well known tests (see related work for details on such tests) available in these languages to check if query answering is decidable. If the translated quad-system passes one of these tests, then query answering can be performed on this translation using available algorithms in these expressive languages. But, such an approach is often discouraged, because of the non-applicability of the already available tools and techniques available for reasoning over quads. Instead, we in the following define three classes of quad-systems, namely SAFE, MSafe and CSAFE, that are FECs and for which query entailment is decidable. Finiteness/Decidability is achieved by putting certain restrictions (explained below) on the blank nodes generated in the dChase.

Recall that, for any quad-system \( QS_c \) the set of blank-nodes \( \mathbf{B}(dChase(QS_c)) \) in its dChase\((QS_c)\) not only contain blank nodes present in \( QS_c \), i.e. \( \mathbf{B}(QS_c) \), but also contain Skolem blank nodes that are generated during the dChase construction process. Note that the following relation holds: \( \mathbf{B}_{sk}(dChase(QS_c)) = \mathbf{B}(dChase(QS_c)) \setminus \mathbf{B}(QS_c) \). We assume w.l.o.g. that for any set of BRs \( R \), any BR in \( R \) has a unique rule identifier, and we often write \( r_i \) for the BR in \( R \), whose identifier is \( i \).

**Definition 4.1 (Origin RuleId/Vector).** For any Skolem blank node \( _:: b \), generated in the dChase by the application of a BR \( r_i = body(r_i)(\bar{x}, \bar{y}) \rightarrow head(r_i)(\bar{x}, \bar{y}) \) using assignment \( \mu: \{ \bar{x} \} \rightarrow C \), i.e. \( :: b = \mu^{\text{ext}}(\bar{y}) \), for some \( y_j \in \bar{y} \), with \( \bar{x} | \mu^{\text{ext}}(\bar{y}) = \bar{w} \), we say the origin ruleId (resp. vector) of \( :: b \) is \( i \) (resp. \( \bar{w} \)), noted originRuleId(_:: b) = i (resp. originVector(_:: b) = \bar{w}).

As we saw in lemma 3.3, any such Skolem blank node \( _:: b \), generated in the dChase can uniquely be represented by the expression \( (i, j, \bar{w}) \), where \( i \) is rule id, \( j \) is identifier of the existentially quantified variable \( y_j \) in \( r_i \), substituted by \( _:: b \) during the application of \( \mu \) on \( r_i \). Also in the above case, we denote relation between each constant \( k = \mu^{\text{ext}}(\bar{y})(x_h) \), \( x_h \in \{ \bar{x} \} \), to \( _:: b \) with the relation childOf. Moreover, since children of a Skolem blank node can be Skolem blank nodes, which themselves can have children, one can naturally define relation descendantOf=childOf as the transitive closure of childOf. Note that according to the above definition, ‘descendantOf’ is not reflexive. In addition, we could keep track of the set of contexts in which...
a blank-node was first generated, using the following notion:

**Definition 4.2** (Origin-contexts). For any quad-system \( QS_C \) and for any Skolem blank node \( _b : b \in B_{\text{sk}}(d\text{Chase}(QS_C)) \), the set of origin-contexts for \( _b : b \) is given by \( \text{originContexts}( _b : b ) = \{ c \mid \exists i. \ c : (s, p, o) \in d\text{Chase}_i(QS_C), s = _b \wedge p = _b \wedge o = _b ; b \text{ and } \not\exists j < i \text{ with } c' : (s', p', o') \in d\text{Chase}_j(QS_C), s' = _b \wedge p' = _b \wedge o' = _b ; b \text{ for any } c' \in C \} \).

Intuitively, origin-contexts for a Skolem blank node \( _b : b \) are the set of contexts in which triples containing \( _b : b \) are first generated, during the dChase construction. Note that there can be multiple contexts to which \( _b : b \) can simultaneously be generated. By setting \( \text{originRuleId}(k) = \text{n.d.} \), (resp. \( \text{originVector}(k) = \text{n.d.} \), resp. \( \text{originContexts}(k) = \text{n.d.} \)) where n.d. is an ad hoc constant, for every \( k \in B_{\text{sk}}(d\text{Chase}(QS_C)) \), we extend the definition of origin ruleId, (resp. origin vector, resp. origin-contexts) to all the constants in the dChase of a quad-system.

**Example 4.3.** Consider the quad-system \( < Q_C, R > \), where \( Q_C = \{ c_1 : (a, b, c) \} \). Suppose \( R \) is the following set:

\[
R = \left\{ c_1 : (x_{11}, x_{12}, x_1) \rightarrow c_2 : (x_{11}, x_{12}, y_1) (r_1) \right\} \\
\left\{ c_2 : (y_{21}, y_{22}, x_2) \rightarrow c_3 : (y_{21}, y_{22}, x_2) (r_2) \right\} \\
\left\{ c_3 : (x_{31}, x_{32}, x_3) \rightarrow c_2 : (y_{31}, x_{31}, x_3) (r_3) \right\}
\]

Suppose that for brevity quantifiers have been omitted, and variables of the form \( y_i \) or \( y_j \) are implicitly existentially quantified. Iterations during dChase construction are:

\[
d\text{Chase}_0(QS_C) = \{ c_1 : (a, b, c) \} \\
d\text{Chase}_1(QS_C) = \{ c_1 : (a, b, c), c_2 : (a, b, _b : b_1) \} \\
d\text{Chase}_2(QS_C) = \{ c_1 : (a, b, c), c_2 : (a, b, _b : b_1), c_3 : (_b : b_2, _b : b_3, _b : b_1) \} \\
d\text{Chase}_3(QS_C) = \{ c_1 : (a, b, c), c_2 : (a, b, _b : b_1), c_3 : (_b : b_2, _b : b_3, _b : b_1), c_3 : (_b : b_4, _b : b_1) \}
\]

Also note:

\[
\text{originRuleId}( _b : b_1 ) = 1, \text{originRuleId}( _b : b_2 ) = \text{originRuleId}( _b : b_3 ) = 2, \text{originRuleId}( _b : b_4 ) = 3,
\]

\[
\text{originVector}( _b : b_1 ) = (a, b), \text{originVector}( _b : b_2 ) = \text{originVector}( _b : b_3 ) = ( _b : b_1 ), \text{originVector}( _b : b_4 ) = ( _b : b_3, _b : b_1 )
\]

Fig. 1. descendance graph of \( _b : b_4 \) in example 4.3. Note: n.d. labels note shown

\[
\text{originContexts}( _b : b_1 ) = \{ c_2 \}, \text{originContexts}( _b : b_2 ) = \{ c_2 \}, \text{originContexts}( _b : b_3 ) = \{ c_2 \}, \text{originContexts}( _b : b_4 ) = \{ c_2 \}
\]

Also \( _b : b_1 \) descendantOf \( _b : b_3 ; _b : b_1 \) descendantOf \( _b : b_2 ; _b : b_3 \) descendantOf \( _b : b_4 ; _b : b_1 \) descendantOf \( _b : b_4 \).

For any Skolem blank node \( _b : b \) (in dChase), its descendent hierarchy can be analyzed using a descendance graph \( (V, E, \lambda_r, \lambda_v, \lambda_c) \), which is a labeled graph rooted at \( _b : b \), whose set of nodes \( V \) are constants in the dChase, the set of edges \( E \) is s.t. \( (k, k') \in E \), iff \( k' \) is a descendant of \( k \). \( \lambda_r, \lambda_v, \lambda_c \) are node labeling functions \( \lambda_r(k) = \text{originRuleId}(k), \lambda_v(k) = \text{originVector}(k) \), s.t. \( \lambda_c(k) = \text{originContexts}(k) \), for any \( k \in V \). Descendance graph for \( _b : b_4 \) of example 4.3 is shown in Fig.1. For any two vectors of constants \( \vec{v}, \vec{w} \), we note \( \vec{v} \equiv \vec{w} \), iff there exists a bijection \( \mu : B(\vec{v}) \rightarrow B(\vec{w}) \) s.t. \( \vec{w} = \vec{v}[\mu] \).

**Definition 4.4** (safe, msafe, csafe quad-systems). A quad-system \( QS_S \) is said to be unsafe (resp. unsafe, resp. unsafe) if it exists Skolem blank nodes \( _b : b \neq _b : b' \) in dChase(QS_C) s.t. \( _b : b \) is a descendant of \( _b : b' \), with originRuleId(_b : b) = originRuleId(_b : b') and originVector(_b : b) \n d\text{Chase}_i(QS_C) \). A quad-system is safe (msafe, csafe) if it is not unsafe (resp. unsafe, resp. unsafe).

Intuitively, safe, msafe and csafe quad-systems, do not allow repetitive generation of Skolem blank-nodes with a certain set of attributes in its dChase. The containment relation between the class of safe, msafe, and
c-safe quad-systems are established by the following theorem:

**Theorem 4.5.** Let $\text{SAFE}$, $\text{MSAFE}$, and $\text{CSAFE}$ denote the class of safe, m-safe, and c-safe quad-systems, respectively, then the following holds:

$$
\text{CSAFE} \subseteq \text{MSAFE} \subseteq \text{SAFE}
$$

**Proof.** We first show $\text{MSAFE} \subseteq \text{SAFE}$, by showing the inverse inclusion of their compliments, i.e. $\text{UNSAFE} \subseteq \text{UNMSAFE}$. Suppose a given quad-system $QS_C$ is unsafe, then by definition its dChase contains two distinct Skolem blank nodes $\_: b, \_: b'$ s.t. $b$ is a descendant of $\_: b'$, with $\text{originRuleId}(\_: b) = \text{originRuleId}(\_: b')$ and $\text{originVector}(\_: b) \equiv \text{originVector}(\_: b')$. But this implies that $\text{originRuleId}(\_: b) = \text{originRuleId}(\_: b')$. Hence, by definition $QS_C$ is unsafe. Hence, $\text{UNSAFE} \subseteq \text{UNMSAFE}$ ($\dagger$)

Now, we show that $\text{CSAFE} \subseteq \text{MSAFE}$, by showing that $\text{UNMSAFE} \subseteq \text{UNCSAFE}$. Suppose a given quad-system $QS_C = (Q_C, R)$ is unsafe, then by definition its dChase contains two distinct Skolem blank nodes $\_: b, \_: b'$ s.t. $b$ is a descendant of $\_: b'$, with $\text{originRuleId}(\_: b) = \text{originRuleId}(\_: b')$. But this implies that there exists a BR $r_i = \text{body}(r_i)(\vec{x}, \vec{y}) \rightarrow \text{head}(r_i)(\vec{x}, \vec{y})$, assignment $\mu$, (resp. $\mu'$), s.t. $\_: b$ (resp. $\_: b'$) was generated in $d\text{Chase}(QS_C)$ as result of application of $\mu$ (resp. $\mu'$) on $r_i$. That is $\_: b = y_j[\mu^{\text{ext}}(\vec{y})]$, and $\_: b' = y_k[\mu'^{\text{ext}}(\vec{y})]$, where $y_j, y_k \in \vec{y}$. We have the following two subcases (i) $j = k$, (ii) $j \neq k$:

Suppose $j = k$, then it immediately follows that $\text{originContexts}(\_: b) = \text{originContexts}(\_: b')$. Hence, $QS_C$ is unsafe. Suppose $j \neq k$, then by construction of dChase, on application of $\mu'$ on $r_i$, along with $\_: b'$, there gets also generated a Skolem blank node $\_: b'' = y_j[\mu'^{\text{ext}}(\vec{y})]$, with $y_j \in \vec{y}$. Since, $\_: b$ and $\_: b''$ are generated by substitutions of the same variable $y_j \in \vec{y}$ of BR $r_i$, $\text{originContexts}(\_: b) = \text{originContexts}(\_: b'')$. Also since $\text{childOf}(\_: b') = \text{childOf}(\_: b'') = \{\vec{x}[\mu'^{\text{ext}}(\vec{y})]\}$, $\_: b$ is a descendant of $\_: b''$. Hence, by definition, it holds that $QS_C$ is unsafe. Hence, $\text{UNMSAFE} \subseteq \text{UNCSAFE}$ ($\dagger$).

From $\dagger$ and $\ddagger$, it follows that $\text{CSAFE} \subseteq \text{CSAFE} \subseteq \text{SAFE}$. To show that the containments are strict, consider the quad-system $QS_C$ in example 4.3. By definition, $QS_C$ is m-safe, however unsafe, as the Skolem blank nodes $\_: b_1, \_: b_4$, which have the same origin contexts are s.t. $\_: b_1$ is a descendant of $\_: b_4$. Hence, $\text{CSAFE} \subseteq \text{MSAFE}$. For $\text{MSAFE} \subseteq \text{SAFE}$, the following example shows an instance of a quad-system that is unsafe, yet is safe.
It can be seen that \( _{-} : b_1, _{-} : b_2, _{-} : b_3, _{-} : b_4, _{-} : b_5 \) form a descendant chain, since \( _{-} : b_i \) descendantOf \( _{-} : b_{i+1} \), for each \( i = 1, \ldots, 4 \). Also, \( \text{originRuleId}(_{-} : b_i) = \text{originRuleId}(_{-} : b_{i+1}) \), for each \( i = 1, \ldots, 4 \). Hence, it turns out that \( QS_C \) is ununsafe. However, it can be seen that \( \text{originVector}(_{-} : b_i) \neq \text{originVector}(_{-} : b_j) \), for \( 1 \leq i \neq j \leq 5 \), and hence, by definition, \( QS_C \) is safe with a terminating dChase. It can be noticed that during each distinct application of \( r_1 \), the vector of constants bound to the vector of variables \((x_{11}, \ldots, x_{14})\) are different w.r.t. \( \bowtie \). Safe quad-systems in this way are capable of recognizing such positive cases of finite dChases, which are classified as negative cases by msafe quad-systems, by also keeping track of the origin vectors of Skolem blank-nodes in its dChase.

The following property shows that for a safe quad-system, the descending graph of any Skolem blank node in its dChase is a directed acyclic graph (DAG):

**Property 4.7 (DAG property).** For a safe (\( \text{csafe}, \text{msafe} \)) quad-system \( QS_C \), and for any blank node \( b \in B_{sk}(d\text{Chase}(QS_C)) \), its descending graph is a DAG.

**Proof.** By construction, as there exists no descendant for any constant \( k \in C(QS_C) \), there cannot be any out-going edge from any such \( k \). Hence, any member of \( C(QS_C) \) cannot be involved in cycles. Therefore, the only members that can be involved can be the members of \( C(d\text{Chase}(QS_C)) \). But if \( \text{originVector}(_{-} : b_i) \neq \text{originVector}(_{-} : b_j) \), for \( 1 \leq i \neq j \leq 5 \), then this implies that \( _{-} : b_i \) is a descendant of \( _{-} : b_j \). Since this would violate the prerequisites of being safe (resp. \( \text{csafe}, \text{msafe} \)), and imply that \( QS_C \) is unsafe (resp. \( \text{unsafe}, \text{unmsafe} \)), which is a contradiction.

Since the descendence graph \( G \) of any Skolem blank node \( _{-} : b \in B_{sk}(d\text{Chase}(QS_C)) \) is s.t. \( G \) is rooted at \( _{-} : b \) and is acyclic, any directed path from \( _{-} : b \) terminates at some node. Hence, one can use a tree traversal technique, such as preorder (visit a node first and then its children) to sequentially traverse nodes in \( G \). The algorithm 1 takes a descendence graph \( G \) and unravels it into a tree. The algorithm first removes all the transitive edges from \( G \), i.e. if there are \( v, v', v'' \in V \), with \( (v, v') \in E, (v', v'') \in E \), then it removes \( (v, v'') \). Note that, in the resulting graph, the presence of a path from \( v \) to \( v'' \) still gives us the information that \( v'' \) is a descendant of \( v \). The algorithm then traverses the graph in preorder fashion, as it encounters a node \( v \), if \( v \) has an indegree \( k \) greater than one, it replaces \( v \) with \( k \) fresh nodes \( v_1, \ldots, v_k \), and distributes the set of edges incident to \( v \) across \( v_1, \ldots, v_k \) s.t. (i) each \( v_i \) has at-most one incoming edge (ii) all the edges incident to \( v \) are incident to some \( v_i, i \in \{1, \ldots, k\} \). Whereas, if \( v \) has an indegree 1, whereas outdegree of \( v_i \) is same as the outdegree of \( v \), \( i \in \{1, \ldots, k\} \). Hence, after the above operation each \( v_i \) has an indegree 1, whereas outdegree of \( v_i \) is same as the outdegree of \( v \), \( i \in \{1, \ldots, k\} \). Hence, after all the nodes are visited, every node except the root in the new graph \( G \) has an indegree 1. \( G \) is still rooted, connected, acyclic, and is hence a tree. The algorithm terminates as there are no cycles in graph, and at some point reaches a node with no children. For instance, the unraveling of the descendence graph of \( _{-} : b_3 \) in Fig. 1 is shown in Fig. 2. The following property holds for any Skolem blank node of a safe quad-system.

**Property 4.8.** For a safe quad-system \( QS_C = (Q_C, R) \), and any Skolem blank node in dChase\((QS_C)\), the unraveling (Algorithm 1) of its descendence graph results in a tree \( t = (V, E, \lambda_r, \lambda_o, \lambda_c) \) s.t.:

```plaintext
Algorithm 1:
UnRavel (Descendance Graph G)

/* procedure to unravel, a descendance graph into a tree */
Input : descendance graph G = (V, E, \lambda_r, \lambda_o, \lambda_c)
Output: A labeled Tree G

begin
G = (V, E, \lambda_r, \lambda_o, \lambda_c) := RemoveTransitiveEdges(G);
foreach Node v_o \in preOrder(G) do
    if (k = indegree(v_o)) > 1 then
        \{v_1, \ldots, v_k\} := getFreshNodes();
        foreach v_i \notin V is fresh
            /* replace old node v_o by the fresh nodes in V */
            removeNodeFrom(v_o, V);
            addNodesTo(\{v_1, \ldots, v_k\}, V);
        foreach (v_o, v_j) \in E do
            /* replace each outgoing edge from v_o with a fresh outgoing edges from each fresh node v_1, v_2, v_3 */
            removeEdgeFrom((v_o, v_j), E);
            addEdgesTo((v_i, v_j), E);
        i := 1;
    foreach (v_o, v_j) \in E do
        /* replace each incoming edge of v_o with an incoming edge for a unique v_i */
        removeEdgeFrom((v_o, v_j), E);
        addEdgeTo((v_i, v_j), E);
    ++i;
end
/* restrict node labels to the updated set of nodes in V */
\lambda_r := \lambda_r|V, \lambda_o := \lambda_o|V, \lambda_c := \lambda_c|V;
return G;
```
1. Since any node \( t \) is from the set \( C(QS_C) \), any leaf node of \( t \) is from the set \( C(QS_C) \).
2. Since \( t \) is from the set \( C(QS_C) \), any non-leaf node of \( t \) is from the set \( B_{sk}(dChase(QS_C)) \).
3. Since \( t \) is from the set \( C(QS_C) \), the order of \( t \) is \( \leq w \), where \( w = \max_{r \in R}[fr(r)] \).
4. Since \( t \) is from the set \( C(QS_C) \), there cannot be a path between \( b \neq b' \in V \), with \( \lambda_r(b) = \lambda_r(b') \) and \( \lambda_r(b) \equiv \lambda_r(b') \).
5. Since \( t \) is from the set \( C(QS_C) \), there cannot be a path between \( b \neq b' \in V \), with \( \lambda_r(b) = \lambda_r(b') \), if \( QS_C \) is also msafe.
6. Since \( t \) is from the set \( C(QS_C) \), there cannot be a path between \( b \neq b' \in V \), with \( \lambda_r(b) = \lambda_r(b') \), if \( QS_C \) is also csafe.

**Proof.**
1. Since any node \( n \) in the descendance graph is s.t. \( n \in C(dChase(QS_C)) \), and since \( C(dChase(QS_C)) = C(QS_C) \ominus B_{sk}(dChase(QS_C)) \), Since any member \( m \in B_{sk}(dChase(QS_C)) \) is generated from an application of a BR with an assignment \( \mu \) s.t. its frontier variables are assigned by \( \mu \) with a set of constants, \( m \) has at-least one child. But, since \( n \) is a leaf node, \( n \in C(QS_C) \).
2. Since any member \( m \in C(QS_C) \) cannot have descendants and since any non-leaf node has children, \( m \) cannot be a non-leaf node. Hence, non-leaf nodes should be from \( B_{sk}(dChase(QS_C)) \).
3. The order of \( t \) is the maximal outdegree among the nodes of \( t \), and outdegree of a node is the number of children it has. Since any node in \( t \) with non-zero outdegree is a Skolem blank-node

\[ \exists b \text{ generated by application of an assignment } \mu \text{ on } r = body(r)(\vec{x}, \vec{z}) \rightarrow head(r)(\vec{x}, \vec{y}) \in R, \text{ the number of children } \exists b \text{ has equals } ||\vec{x}||. \] Hence, order of \( t \) is bounded by \( w \).
4. Since any path from \( b \) to \( b' \) implies that \( b' \) is a descendant of \( b \), it should be the case that \( \lambda_r(b) \neq \lambda_r(b') \) or \( \lambda_r(b) \neq \lambda_r(b') \) otherwise safety condition would be violated.
5. Similar to above, immediate by definition.
6. Similar to above, immediate by definition.

The property above is exploited to show that there exists a finite bound in the dChase size and its computation time.

**Lemma 4.9.** For any safe/msafe/csafe quad-system \( QS_C = (Q_c, R) \), the following holds: (i) the dChase size \( ||dChase(QS_C)|| = O(2^{||QS_c||}) \), (ii) \( dChase(QS_C) \) can be computed in 2EXPTIME, (iii) if \( |R| \) and the set of schema triples in \( Q_c \) is fixed to a constant, then \( ||dChase(QS_C)|| \) is a polynomial in \( ||QS_C|| \) and can be computed in PTIME.

**Proof.** The proofs are provided for safe quad-systems, but since \( CSafe \subseteq MSafe \subseteq Safe \) and since we are giving upper bounds, they also propagate trivially to msafe and csafe quad-systems.

(i) For any blank node in \( dChase(QS_C) \), the size of its originVector is upper bounded by \( w = \max_{r \in R}[fr(r)] \). If \( S \) is the set of all origin vectors of blank-nodes in \( dChase(QS_C) \), then cardinality of the set \( S' = S \setminus |S| \) is upper bounded by \( (|U(QS_c)| + |L(QS_c)| + w)w \), which means that \( S' = O(2^{||QS_c||}) \). Also, since the set of origin ruleId labels, \( Rids \), can at most be \( |R| \), hence the cardinality of the set \( Rids \times S' = O(2^{||QS_c||}) \). For the descendance tree \( t \) of any Skolem blank node of \( dChase(QS_C) \), since there cannot be paths in \( t \) between distinct \( b \) and \( b' \), s.t. \( originRuleId(b) = originRuleId(b') \) and \( originVector(b) \neq originVector(b') \), the length of any such path is upper bounded by \( Rids \times S' = O(2^{||QS_c||}) \). However, it turns out that this above upper bound provided is loose, as there is the need of additional filter BRs to transform/back-propagate vectors of constants associated with Skolem blank nodes generated by repetitive application of the same BR. For instance, consider the set of BRs in eg: 4.6. The BR \( r_1 \) transforms the origin vector to a new vector each time during its application. BRs \( r_2 \cdot r_3 \) deals with back propagation of these vectors back to input origin vectors of BR \( r_1 \). Hence, such filter BRs rule out the case.
of a BR being applied to a quad that contains a Skolem blank node that was generated using the same BR on an isomorphic origin vector, ensuring that the safety criteria for Skolem blank-nodes generated is not violated. It turns out that the number of such filter BRs required is polynomial w.r.t. to the number of descendants with the same rule id, for a node in \( t \). Hence, it turns out the depth of \( t \) is polynomially bounded by \( |R| \). (Note that depth of \( t \) is bounded by \(|R|\) for msafe quad-systems. Also since, the set of origin context labels are bounded by the set of existential variables in \( R \), depth of \( t \) is bounded by \(|R|\) for csafe quad-systems.) Also order of the tree is bounded by \( w \). Hence, any such tree can have at most \( O(2^{|QS_C|}) \) leaf nodes, \( O(2^{|QS_C|}) \) inner nodes, and \( O(2^{|QS_C|}) \) nodes. Since each of the leaf nodes can only be from \( C(QS_C) \) and each of the inner nodes correspond to an existential variable in \( R \), the number of such possible trees are clearly bounded double exponentially in \(|QS_C|\), hence bounds the number of Skolem blank nodes generated in the dChase.

(ii) From (i) \( ||dChase(QS_C)|| \) is double exponential in \(|QS_C|\), and since each iteration add at least one quad to its dChase, the number of iterations are bounded double exponentially in \(|QS_C|\). Also, by lemma 3.2 any iteration \( i \) can be done in time \( O(||dChase_{i-1}(QS_C)|| |R|) \). Hence, by using (i), we get \( ||dChase_{i-1}(QS_C)|| = O(2^{2^{|QS_C|}}) \). Hence, we can infer that each iteration \( i \) can be done in time \( O(2^{|R|}2^{2^{|QS_C|}}) \). Also since the number of iterations is double exponential, computing \( dChase(QS_C) \) is in 2EXPTIME.

(iii) Since \(|R|\) is fixed to a constant, the set of existential variables is also a constant. Also in this case, since the size of the frontier of any \( r \in R \) is also a constant, the order and depth of any descendant tree \( t \) of a Skolem blank node is a constant. Hence, the number of (leaf) nodes of \( t \) is bounded by a constant. Also in this setting, the label of inner nodes of \( t \), which correspond to existential variables, is also a constant, and the leaf nodes of \( t \) can only be a constant in \( C(QS_C) \). Hence, the number of descendant trees and consequently, the number of Skolem blank nodes generated is bounded by \( O(|C(QS_C)|^z) \), where \( z \) is a constant. Hence, the set of constants generated in \( dChase(QS_C) \) is a polynomial in \(|QS_C|\), and so is \( ||dChase(QS_C)|| \).

Since in any dChase iteration except the final one, at least one quad should be added, and also since the final dChase can have at most \( O(|QS_C|^z) \) triples, the total number of iterations are bounded by \( O(|QS_C|^z) \). By lemma 3.2, since any iteration \( i \) can be computed in \( O(||dChase_{i-1}(QS_C)|| |R|) \) time, and since \(|R|\) is a constant, the time required for each iteration is a polynomial in \( ||dChase_{i-1}(QS_C)|| \), which is at most a polynomial in \(|QS_C|\). Hence, any dChase iteration can be performed in polynomial time in size of \( QS_C \) (\( \dagger \)). From (\( \dagger \)) and (\( \dagger \)), it can be concluded that dChase can be computed in PTIME.

Lemma 4.10. For any safe/msafe/csafe quad-system, the following holds: (i) data complexity of CCQ entailment is in PTIME, (ii) combined complexity of CCQ entailment is in 2EXPTIME.

Proof. Note that the proofs are provided for safe quad-systems, but since \( CSAFE \subset MSAFE \subset SAFE \) and since we are giving upper bounds, they also propagate trivially to msafe and csafe quad-systems.

Given a safe quad-system \( QS_C = \langle QC, R \rangle \), since \( dChase(QS_C) \) is finite, a boolean CCQ \( CQ() \) can naively be evaluated by binding the set of constants in the dChase to the variables in the \( CQ() \), and then checking if any of these bindings are contained in \( dChase(QS_C) \). The number of such bindings can at most be \( ||dChase(QS_C)||^{|CQ|} \) (\( \dagger \)).

(i) Since for data complexity, the size of the BRs \(|R|\), the set of schema triples, and \(|CQ()|\) is fixed to constant. From lemma 4.9 (iii), we know that under the above mentioned settings the dChase can be computed in PTIME and is polynomial in the size of \( QS_C \). Since \(|CQ()|\) is fixed to a constant, and from (\( \dagger \)), binding the set of constants in \( dChase(QS_C) \) on \( CQ() \) still gives a number of bindings that is worst case polynomial in the size of \(|QS_C|\). Since membership of these bindings can checked in the polynomially sized dChase in PTIME, the time required for CCQ entailment is in PTIME.

(ii) Since in this case \( ||dChase(QS_C)|| = O(2^{2^{2^{|QS_C|}}} |CQ()|) \) (\( \dagger \)), from (\( \dagger \)) and (\( \dagger \)), binding the set of constants in \( dChase(QS_C) \) to \( CQ() \) amounts to \( O(2^{|CQ()||2^{2^{|QS_C|}}} \) number of bindings. Since the dChase is double exponential in \(|QS_C|\), checking the membership of each of these bindings can be done in 2EXPTIME. Hence, the combined complexity is in 2EXPTIME.

Theorem 4.11. For any safe/msafe/csafe quad-system, the following holds: (i) The data complexity of CCQ entailment is PTIME-complete (ii) The combined complexity of CCQ entailment is 2EXPTIME-complete.
Proof. (i) (Membership) See lemma 4.10 for the membership in PTIME.
(Hardness) Follows from the PTIME-hardness of data complexity of CCQ entailment for Range-Restricted quad-systems (Theorem 5.2), which are contained in safe/msafe/csafe quad-systems.
(ii) (Membership) See lemma 4.10.
(Hardness) See following heading.

4.1. 2EXPSPACE-Hardness of CCQ Entailment

In this subsection, we show that the combined complexity of the decision problem of CCQ entailment for context acyclic quad-systems is 2EXPSPACE-hard. We show this by reduction of the word-problem of a 2EXPSPACE deterministic turing machine (DTM) to the CCQ entailment problem. A DTM $M$ is a tuple $M = (Q, \Sigma, \Delta, q_0, q_A)$, where
- $Q$ is a set of states,
- $\Sigma$ is a finite set of letters that includes the blank symbol $\square$,
- $\Delta$: $(Q \times \Sigma) \rightarrow (Q \times \Sigma \times \{+1, -1\})$ is the transition function,
- $q_0 \in Q$ is the initial state,
- $q_A \in Q$ is the accepting state.

W.l.o.g. we assume that there exists exactly one accepting state, which is also the lone halting state. A configuration is a word $\vec{\alpha} \in \Sigma^*Q\Sigma^*$. A configuration $\vec{\alpha}_2$ is a successor of the configuration $\vec{\alpha}_1$, iff one of the following holds:

1. $\vec{\alpha}_1 = \vec{w}_1q\sigma_1\vec{w}_r$, and $\vec{\alpha}_2 = \vec{w}_1\sigma'q'\sigma\vec{w}_r$, if $\Delta(q, \sigma) = (q', \sigma', R)$, or
2. $\vec{\alpha}_1 = \vec{w}_1q\sigma$ and $\vec{\alpha}_2 = \vec{w}_1\sigma'q', if \Delta(q, \sigma) = (q', \sigma', R)$, or
3. $\vec{\alpha}_1 = \vec{w}_1\sigma_1q\sigma\vec{w}_r$, and $\vec{\alpha}_2 = \vec{w}_1\sigma_1\sigma'\vec{w}_r$, if $\Delta(q, \sigma) = (q', \sigma', L)$.

where $q, q' \in Q$, $\sigma, \sigma' \in \Sigma$, and $\vec{w}_1, \vec{w}_r \in \Sigma^*$. Since number of configurations can at most be doubly exponential in the size of the input string, and since 2EXPSPACE $\subseteq$ 2EXPSPACE, the number of tape cells traversed by the DTM tape head is also bounded doubly exponentially. A configuration $\vec{c} = \vec{w}_1q\vec{w}_r$ is an accepting configuration iff $q = q_A$. A language $L \subseteq \Sigma^*$ is accepted by a 2EXPSPACE bounded DTM $M$, iff for every $\vec{w} \in L$, $M$ accepts $\vec{w}$ in time $O(2^n|\vec{w}|)$.

Simulating DTMs using Safe Quad-Systems Consider a double exponential time bounded DTM $M = (Q, \Sigma, \Delta, q_0, q_A)$, and a string $\vec{w}$, with $||\vec{w}|| = n$.

In order to simulate $M$, we construct a quad-system $QS^M_C = (Q, C, R)$, where $C = \{c_1, \ldots, c_n\}$, whose various elements represents the constructs of $M$. We follow the technique in works such as [34,36] to iteratively generate a doubly exponential number of objects that represent the cells of the tape of the DTM.

Let $Q^M_C$ be initialized with the following quads:
- $c_0: (k_0, rdf:type, R), c_0: (k_1, rdf:type, R), c_0: (k_0, rdf:type, min_0), c_0: (k_1, rdf:type, max_0), c_0: (k_0, succ_0, k_1)$

Now for each pair of elements of type $R$ in $c_i$, a Skolem blank-node is generated in $c_{i+1}$, and hence follows the recurrence relation $r(m+1) = [r(m)]^2$, with seed $r(1) = 2$, which after $n$ iterations yields $2^{2^n}$. In this way, a doubly exponential long chain of elements is created in $c_n$, using the following set of rules:

$c_i: (x_0, rdf:type, R), c_i: (x_1, rdf:type, R) \rightarrow \exists y c_{i+1}: (x_0, x_1, y, rdf:type, R) (eBr)$

The combination of minimal element with the minimal element (elements of type $min_i$) in $c_i$ create the minimal element in $c_{i+1}$, and similarly the combination of maximal element with the maximal element (elements of type $max_i$) in $c_i$ create the maximal element of $c_{i+1}$

$c_{i+1}: (x_0, x_0, x_1), c_{i+1}: (x_0, rdf:type, min_i) \rightarrow c_{i+1}: (x_1, rdf:type, min_{i+1})$
$c_{i+1}: (x_0, x_0, x_1), c_{i+1}: (x_0, rdf:type, max_i) \rightarrow c_{i+1}: (x_1, rdf:type, max_{i+1})$

Successor relation $succ_{i+1}$ is created in $c_{i+1}$ using the following set of rules, using the well-known, integer counting technique:

$c_i: (x_1, succ_i, x_2), c_{i+1}: (x_0, x_1, x_3)$
$c_{i+1}: (x_0, x_2, x_4) \rightarrow c_{i+1}: (x_3, succ_{i+1}, x_4)$
$c_i: (x_1, succ_i, x_2), c_{i+1}: (x_1, x_3, x_5)$
$c_{i+1}: (x_2, x_4, x_6), c_{i+1}: (x_3, rdf:type, max_i), c_{i+1}: (x_4, rdf:type, min_i) \rightarrow c_{i+1}: (x_5, succ_{i+1}, x_6)$

Each of the above set rules are instantiated for $0 \leq i < n$, and in this way after $n$ generating dChase iterations, $c_n$ has doubly exponential number of elements of type $R$, that are ordered linearly using the relation $succ_n$.

By virtue of the first rule below, each of the objects
Since in our construction, each letter is on the same cell using the following axiom:

$$\sigma$$

we could constrain that no two letters $$\sigma \neq \sigma'$$ are on the same cell. Also the transitive closure of $$succ$$ is defined as the relation $$succ$$

\[
c_n : (x_0, succ, x_1) \rightarrow c_n : (x_0, succ, x_1)
\]

Also using a similar construction, we could reuse the $$2^{2^n-1}$$ linearly ordered elements in $$c_n-1$$ to create another linearly ordered chain of double exponential number of objects in $$c_n$$ that represents configurations of $$M$$, whose minimal element is of type $$conInit$$, and the linear order relation being $$conSucc$$.

Various triple patterns that are used to encode the possible configurations, runs and their relations in $$M$$ are:

\[
(x_0, head, x_1) \text{ denotes the fact that in configuration } x_0, \text{ the DTM is at cell } x_1.
\]

\[
(x_0, state, x_1) \text{ denotes the fact that in configuration } x_0, \text{ the DTM is in state } x_1.
\]

\[
(x_0, \sigma, x_1) \text{ where } \sigma \in \Sigma, \text{ denotes the fact that in configuration } x_0, \text{ the cell } x_1 \text{ contains } \sigma.
\]

\[
(x_0, succ, x_1) \text{ denotes the linear order between cells of the tape.}
\]

\[
(x_0, succ \rightarrow succ, x_1) \text{ denotes the transitive closure of } succ.
\]

\[
(x_0, rdf : type, Accept) \text{ denotes the fact that the configuration } x_0 \text{ is an accepting configuration.}
\]

Since in our construction, each $$\sigma \in \Sigma$$ is represented as relation, we could constrain that no two letters $$\sigma \neq \sigma'$$ are on the same cell using the following axiom:

\[
c_n : (z_1, \sigma, z_2), c_n : (z_1, \sigma', z_2) \rightarrow
\]

for each $$\sigma \neq \sigma' \in \Sigma$$. Note that the above BR has an empty head, is equivalent to asserting the negation of its body.

**Initialization** Suppose the initial configuration is $$q_0 \vec{w} \Box$$, where $$\vec{w} = \sigma_0...\sigma_{n-1}$$, then we enforce this using the following BRs in our quad-system $$QS^M_c$$ as:

\[
c_n : (x_0, rdf : type, conInit), c_n : (x_1, rdf : type, min_n) \rightarrow c_n : (x_1, head, x_1), c_n : (x_1, state, q_0)
\]

\[
c_n : (x_0, rdf : type, min_n) \land \bigwedge_{i=0}^{n-1} c_n : (x_i, succ, x_{i+1}) \land c_n : (x_j, rdf : type, conInit) \rightarrow \\
\bigwedge_{i=0}^{n-1} c_n : (x_j, \sigma_i, x_i) \land c_n : (x_j, \Box, x_n)
\]

\[
c_n : (x_j, rdf : type, conInit), c_n : (x_j, \Box, x_0), c_n : (x_0, succ, x_1) \rightarrow c_n : (x_j, \Box, x_1)
\]

The last BR copies the $$\Box$$ to every succeeding cell in the initial configuration.

**Transitions** For every left transition $$\Delta(q, \sigma) = (q_j, \sigma', -1)$$, the following BR:

\[
c_n : (x_0, head, x_1), c_n : (x_0, \sigma, x_i), c_n : (x_0, state, q), c_n : (x_j, succ, x_i), c_n : (x_0, conSucc, x_1) \rightarrow c_n : (x_1, head, x_j), c_n : (x_1, \sigma', x_i), c_n : (x_1, state, q_j)
\]

For every right transition $$\Delta(q, \sigma) = (q_j, \sigma', +1)$$, the following BR:

\[
c_n : (x_0, head, x_1), c_n : (x_0, \sigma, x_i), c_n : (x_0, state, q), c_n : (x_i, succ, x_j), c_n : (x_0, conSucc, x_1) \rightarrow c_n : (x_1, head, x_j), c_n : (x_1, \sigma', x_i), c_n : (x_1, state, q_j)
\]

**Inertia** If in any configuration the head is at cell $$i$$ of the tape, then in every successor configuration, elements in preceding and following cells of $$i$$ in the tape are retained. The following two BRs ensures this:

\[
c_n : (x_0, head, x_1), c_n : (x_0, conSucc, x_1), c_n : (x_j, succ, x_j), c_n : (x_0, succ, x_i), c_n : (x_1, succ, x_j), c_n : (x_0, conSucc, x_1) \rightarrow c_n : (x_1, head, x_j), c_n : (x_1, \sigma', x_i), c_n : (x_1, state, q_j)
\]

The rules above are instantiated for every $$\sigma \in \Sigma$$.

**Acceptance** A configuration whose state is $$q_A$$ is accepting:

\[
c_n : (x_0, state, q_A) \rightarrow c_n : (x_0, rdf : type, Accept)
\]

If a configuration of accepting type is reached, then it can be back propagated to the initial configuration,
using the following BR:

\[ c_n: (x_0, \text{conSucc}, x_1), c_n: (x_1, \text{rdf}\text{::type}, \text{Accept}) \]

\[ \rightarrow c_n: (x_0, \text{rdf}\text{::type}, \text{Accept}) \]

Finally since \( M \) accepts \( \vec{w} \) iff the initial configuration is an accepting configuration. Let \( CQ^M \) be CCQ:

\[ \exists y \, c_n(y, \text{rdf}\text{::type}, \text{conInit}), c_n(y, \text{rdf}\text{::type}, \text{Accept}) \]

It can easily be verified that \( QS_{CQ}^M \models CQ^M \) iff the initial configuration is an accepting configuration. In order to prove the soundness and completeness of our simulation, we prove the following claims:

**Claim (1)** The quad-system \( QS_{CQ}^M \) in the aforementioned simulation is a c-safe quad-system

It can be noted that only BRs in which existentials are present are the BRs used to generate the double exponential chain of tape cells and configurations, and are of the form (eBR). Note that in each application of such a BR, a blank-node \( _b \) generated in a context \( c_i \), for any \( i = 1, \ldots, n \), is s.t. \( \text{originContexts}(\_b) = \{ c_i \} \) and has exactly two child blank-nodes, each of whose origin contexts is \( \{ c_{i-1} \} \). Hence, any Skolem blank-node generated in any \( c_i \), for \( i = 1, \ldots, n \) is s.t. its child blank-nodes has origin contexts \( c_{i-1} \). Thanks to the above property, it turns out there exists no two blank-nodes \( _b : _b' \) in the dChase of \( QS_{CQ}^M \) s.t. \( _b \) is a descendant of \( _b' \) and \( \text{originContexts}(\_b) = \text{originContexts}(\_b') \). Therefore \( QS_{CQ}^M \) is c-safe.

**Claim (2)** \( QS_{CQ}^M \models CQ^M \) iff \( M \) accepts \( \vec{w} \).

Suppose that \( QS_{CQ}^M \models CQ^M \), then it holds that in any model \( \mathcal{I}^C = \{ I^n \}_{i=1}^n \) of \( QS_{CQ}^M \), \( \mathcal{I}^C \models CQ^M \), which implies that \( I^n \) has an object \( o \) in its domain s.t. \( o \in \text{conInit}^n \) and \( o \in \text{Accept}^n \). But thanks to the acceptance axiom it follows that there exists an object \( o' \) s.t. \( (o, o') \) in the reflexive-transitive closure of \( \text{conSucc}^n \) s.t. \( o' \in \text{Accept}^n \). Also thanks to the initialization axioms, it can be seen that \( o \) represents the initial configuration of \( M \) i.e. it represents the configuration in which the initial state is \( q_0 \), and the left end of the read-write tape contains \( \vec{w} \) followed by trailing \( \square \)'s, with the read-write head positioned at the first cell of the tape. Also the transition axioms makes sure that if \( (o, o'') \in \text{conSucc}^n \), then \( o'' \) represents a successor configuration of \( o \). That is, if \( o \) represents the configuration in which \( M \) is at state \( q \) with read-write head at position \( pos \) of the tape that contains a letter \( \sigma \in \Sigma \), and if \( \Delta(q, \sigma) = (q', o', D) \), then \( o'' \) represents the configuration in which \( M \) is at state \( q' \), which read-write head at the position \( pos - 1/pos + 1 \) depending on whether \( D = -1/1 + 1 \), and \( \sigma' \) at the position \( pos - 1/pos + 1 \) of the tape. As a consequence of the above arguments, it follows that \( o' \) represents an accepting configuration of \( M \), i.e. a configuration in which the state is \( q_A \), the lone accepting, halting state. This means that \( M \) accepts the string \( \vec{w} \).

For the converse, we briefly show that if \( QS_{CQ}^M \not\models CQ^M \) then \( M \) does not accept \( \vec{w} \). Suppose that \( QS_{CQ}^M \not\models CQ^M \), then this implies that there exists a model \( \mathcal{I}^C = \{ I^n \}_{i=1}^n \) of \( QS_{CQ}^M \), s.t. \( \mathcal{I}^C \not\models CQ^M \). This means that no object in the domain of \( I^n \) exists that is a member of both \( \text{conInit}^n \) and \( \text{Accept}^n \). By the initialization axioms, we know that there exists an object \( o \) in the domain of \( I^n \) with \( o \in \text{conInit}^n \) and by preceding discussion, we know that \( o \) represents the initial configuration of \( M \). Also by the initial construction axioms of \( QS_{CQ}^M \), we know that \( o \) is the initial element of a double exponential chain of objects that are linearly ordered by property symbol \( \text{conSucc} \). From transition axioms we know that for any \( o'' \) s.t. \( (o, o'') \in \text{conSucc}^n \), then \( o'' \) represents a valid successor configuration of \( o \), which itself holds for \( o''' \), and so on. This means that for none of the succeeding double exponential configurations of \( M \), the accepting state \( q_A \) holds. This means that \( M \) does not reach an accepting configuration with string \( \vec{w} \), and hence rejects it.

Since the construction above shows the existence of a polynomial time reduction of the word problem of a 2EXPTIME DTM, which is a 2EXPTIME-hard problem, to the CCQ entailment problem over c-safe quad-systems, it immediately follows that CCQ entailment over c-safe/msafe/safe quad-systems is 2EXPTIME-hard.

### 4.2. Procedure for detecting safe/msafe/c-safe quad-systems

In this subsection, we present a procedure for deciding whether a given quad-system is safe (resp. msafe, resp. c-safe) or not. If the quad-system is safe (resp. msafe, resp. c-safe), the result of the procedure is a safe dChase (resp. msafe dChase, c-safe dChase) that contains the standard dChase, and can be used for query answering. Since safety (resp. msafety, resp. c-safety) property of a quad-system is attributed to the dChase of the quad-system, the procedure nevertheless performs the standard operations for computing the dChase, but also generate quads that indicate origin ruleIds and origin vectors (resp. origin ruleIds, resp. origin-contexts) of each Skolem blank node generated. In each iteration, a
Definition 4.12 (Context Scope). The context scope of a term \( t \) in a set of quad-patterns \( Q \), denoted by \( \text{cScope}(t,Q) \), is given as: \( \text{cScope}(t,Q) = \{ c \mid \text{c}: (s,p,o) \in Q, s = t \lor p = t \lor o = t \} \).

For any quad-system \( QS_c = (Q_c, R) \), let \( c_c \) be an ad hoc context identifier s.t. \( c_c \notin C \), then for \( r_i = \text{body}(r_i)(\vec{x}, \vec{z}) \rightarrow \text{head}(r_i)(\vec{x}, \vec{y}) \in R \), we define transformations \( \text{augS}(r_i) \), \( \text{augM}(r_i) \), \( \text{augC}(r_i) \) as follows:

\[
\text{augS}(r_i) = \text{body}(r_i)(\vec{x}, \vec{z}) \rightarrow \text{head}(r_i)(\vec{x}, \vec{y}) \land \forall y_j \in \{ \vec{y} \} \Big[ \bigwedge_{x_k \in \{ \vec{x} \}} c_c : (x_k, \text{descendantOf}, y_j) \land c_c : (y_j, \text{descendantOf}, y_j) \land c_c : (y_j, \text{originRuleId}, i) \land c_c : (y_j, \text{originVector}, \vec{x}) \Big]
\]

It should be noted that \( c_c : (y_j, \text{originVector}, \vec{x}) \) is not a valid quad pattern, and is only used for notation brevity. In the actual implementation, vectors can be stored using an rdf container data structure such as rdf:List, rdf:Seq or by typecasting it as a string.

\[
\text{augM}(r_i) = \text{body}(r_i)(\vec{x}, \vec{z}) \rightarrow \text{head}(r_i)(\vec{x}, \vec{y}) \land \forall y_j \in \{ \vec{y} \} \Big[ \bigwedge_{x_k \in \{ \vec{x} \}} c_c : (x_k, \text{descendantOf}, y_j) \land c_c : (y_j, \text{descendantOf}, y_j) \land c_c : (y_j, \text{originRuleId}, i) \Big]
\]

\[
\text{augC}(r_i) = \text{body}(r_i)(\vec{x}, \vec{z}) \rightarrow \text{head}(r_i)(\vec{x}, \vec{y}) \land \forall y_j \in \{ \vec{y} \} \forall c \in \text{cScope}(y_j, \text{head}(r_i)) \Big[ \bigwedge_{x_k \in \{ \vec{x} \}} c_c : (x_k, \text{descendantOf}, y_j) \land c_c : (y_j, \text{descendantOf}, y_j) \land c_c : (y_j, \text{originContext}, c) \Big]
\]

Intuitively, the transformation \( \text{augS/ augM/augC} \) on a BR \( r_i \), augments the head part of \( r_i \) with additional types of quad patterns, which are the following:

1. \( c_c : (x_k, \text{descendantOf}, y_j) \), for every existentially quantified variable \( y_j \) in \( \vec{y} \) and universally quantified variable \( x_k \in \{ \vec{x} \} \). This is done because during dChase computation any application of an assignment \( \mu \) on \( r_i \) s.t. \( \vec{x}[\mu] = \vec{a} \), resulting in the generation of a Skolem blank node \( _{-} : b = \mu^{\text{ext}}(\vec{y})(y_j) \), any \( a_i \in \{ \vec{a} \} \) is a descendant of \( _{-} : b \). Hence, due to these additional quad-patterns, quads of the form \( c_c : (a_i, \text{descendantOf}, _{-} : b) \) are also produced, and in this way, keeps track of the descendants of any Skolem blank node produced.

2. \( c_c : (y_j, \text{descendantOf}, y_j) \), in order to maintain the reflexivity of ‘\text{descendantOf}’ relation.

3. \( c_c : (y_j, \text{originContext}, c) \), for every existentially quantified variable \( y_j \) in \( \{ \vec{y} \} \), every \( c \in \text{cScope}(y_j, \text{head}(r_i)) \). This is done because during dChase computation, any application of an assignment \( \mu \) on \( r_i \), s.t. \( \vec{x}[\mu] = \vec{a} \), resulting in the generation of a Skolem blank node \( _{-} : b = \mu^{\text{ext}}(\vec{y})(y_j) \), \( c \) is an origin context of \( _{-} : b \). Hence, due to these additional quad-patterns, quads of the form \( c_c : (\_ : b, \text{originContext}, c) \) is also produced. In this way, keeps track of the origin-contexts of any Skolem blank node produced.

4. \( c_c : (y_j, \text{originVector}, \vec{x}) \). This is done because during dChase computation, any application of an assignment \( \mu \) on \( r_i \), s.t. \( \vec{x}[\mu] = \vec{a} \), resulting in the generation of a Skolem blank node \( _{-} : b = \mu^{\text{ext}}(\vec{y})(y_j) \), \( \vec{a} \) is the origin vector of \( _{-} : b \). Hence, due to these additional quad-patterns, quads of the form \( c_c : (\_ : b, \text{originVector}, \vec{a}) \) is also produced. In this way, keeps track of the origin-vector of any Skolem blank node produced.

5. \( c_c : (y_j, \text{originRuleId}, i) \), for every existentially quantified variable \( y_j \) in \( \{ \vec{y} \} \), in order to keep track of the ruleId of the BR used to create any Skolem blank node.

It can be noticed that for any BR \( r_i \) without existentially quantified variables, the transformations \( \text{augS/ augM/ augC} \) leaves \( r_i \) unchanged. For any set...
of BRs $R$, let

$$\text{augS}(R) \text{ (resp. } \text{augM}(R), \text{ resp. } \text{augC}(R)) =$$

$$\bigcup_{r_i \in R} \text{augS}(r_i) \text{ (resp. } \text{augM}(r_i), \text{ resp. } \text{augC}(r_i)) \cup$$

$$\{c_c : (x_1, \text{ descendantOf}, z_1) \land c_c : (z_1, \text{ descendantOf}, x_2) \rightarrow c_c : (x_1, \text{ descendantOf}, x_2)\}$$

The function $\text{unSafeTest}$ (resp. $\text{unMSafeTest}$, resp. $\text{unCSafeTest}$) defined below, given a BR $r_i = \text{body}(r_i)(\vec{x}, \vec{z}) \rightarrow \text{head}(r_i)(\vec{x}, \vec{y})$, an assignment $\mu$, and a quad-graph $Q$ checks, if application of $\mu$ on $r_i$ violates the safety (resp. $\text{msafe}$, resp. $\text{csafe}$) condition on $Q$.

$\text{unSafeTest}(r_i, \mu, Q) = \text{True iff } \exists \vec{b} : \vec{b}_1 : \vec{b}_2 \in B$, with all the following conditions being satisfied:

- $\vec{b} \in \{\vec{x}[\mu]\}$, and
- $c_c : (\vdash \vec{b}_1, \text{ descendantOf}, \vdash \vec{b}) \in Q$, and
- $c_c : (\vdash \vec{b}_2, \text{ originRuleId}, i) \in Q$, and
- $c_c : (\vdash \vec{b}_1, \text{ originVector}, \vec{d}) \in Q$, and $\vec{d} \equiv \vec{x}[\mu]$. 

Intuitively, $\text{unSafeTest}$ returns True, if $\mu$ applied to $r_i$ will produce a fresh Skolem blank node $\vdash \vec{b}'$, whose child $\vdash \vec{b} \in \{\vec{x}[\mu]\}$, and according to knowledge in $Q$, $\vdash \vec{b}'$ a descendant of $\vdash \vec{b}_1$ s.t. originRuleId of $\vdash \vec{b}'$ is $i$ (which is also the origin ruleId of $\vdash \vec{b}_1$) and origin vector of $\vdash \vec{b}'$ is isomorphic to origin vector of $\vec{x}[\mu]$ (which is also the origin vector of $\vdash \vec{b}_1$). The functions $\text{unMSafeTest}$ and $\text{unCSafeTest}$ are similarly defined as follows:

$\text{unMSafeTest}(r_i, \mu, Q) = \text{True iff } \exists \vec{b} : \vec{b}_1 : \vec{b}_2 \in B$, with all the following conditions being satisfied:

- $\vec{b} \in \{\vec{x}[\mu]\}$, and
- $c_c : (\vdash \vec{b}_1, \text{ descendantOf}, \vdash \vec{b}) \in Q$, and
- $c_c : (\vdash \vec{b}_2, \text{ originRuleId}, i) \in Q$.

$\text{unCSafeTest}(r_i, \mu, Q) = \text{True iff } \exists \vec{b} : \vec{b}_1 : \vec{b}_2 \in B$, $\exists y_j \in \{\vec{y}\}$, with all the following being satisfied:

- $\vec{b} \in \{\vec{x}[\mu]\}$, and
- $c_c : (\vdash \vec{b}_1, \text{ descendantOf}, \vdash \vec{b}) \in Q$, and
- $\{c \mid c_c : (\vdash \vec{b}_1, \text{ originContext}, c) \in Q\} = c\text{Scope}(y_j, \text{head}(r_i)(\vec{x}, \vec{y})) \setminus \{c_c\}$. 

For any BR $r_i$ and an assignment $\mu$, the safe/$\text{msafe}$/csafe application of $\mu$ on $r_i$ w.r.t. a quad-graph $Q$ is defined as follows:

apply$_{\text{safe}}$\(+\)(r$_i$, $\mu$, $Q$) = \begin{cases} \text{unSafe, if } \text{unSafeTest}(r_i, \mu, Q) = \text{True}; \\
\text{apply}(r_i, \mu), \text{ Otherwise}; \end{cases}

apply$_{\text{msafe}}$\(+\)(r$_i$, $\mu$, $Q$) = \begin{cases} \text{unMSafe, if } \text{unMSafeTest}(r_i, \mu, Q) = \text{True}; \\
\text{apply}(r_i, \mu), \text{ Otherwise}; \end{cases}

apply$_{\text{csafe}}$\(+\)(r$_i$, $\mu$, $Q$) = \begin{cases} \text{unCSafe, if } \text{unCSafeTest}(r_i, \mu, Q) = \text{True}; \\
\text{apply}(r_i, \mu), \text{ Otherwise}; \end{cases}

where $\text{unSafe} = c_c : (\vdash \vec{b}_1, \vdash \vec{b}_2, \vdash \vec{b}_3, \vdash \vec{b}_4) \in B$ (resp. $\text{unMSafe} = c_c : (\vdash \vec{b}_1, \vdash \vec{b}_2, \vdash \vec{b}_3, \vdash \vec{b}_4) \in B$, resp. $\text{unCSafe} = c_c : (\vdash \vec{b}_1, \vdash \vec{b}_2, \vdash \vec{b}_3, \vdash \vec{b}_4) \in B)$ is a distinguished quad that is generated, if the prerequisites of safety (resp. $\text{msafe}$, resp. $\text{csafe}$) is violated. For any quad-system $Q_S = (Q_C, R)$, we define its safe $d\text{Chase}$ $d\text{Chase}_{\text{safe}}(Q_S)$ as follows:

$d\text{Chase}_{\text{safe}}(Q_S) = Q$;

$d\text{Chase}_{\text{safe}}^{m+1}(Q_S) = d\text{Chase}_{\text{safe}}^{m}(Q_S) \cup \text{apply}_{\text{safe}}(r_i, \mu, d\text{Chase}_{\text{safe}}^{m}(Q_S))$, if there exists $r_i \in \text{augS}(R)$, assignment $\mu$ s.t. applicable$_{\text{augS}(R)}(r_i, \mu, d\text{Chase}_{\text{safe}}^{m}(Q_S))$;

$d\text{Chase}_{\text{safe}}^{m+1}(Q_S) = d\text{Chase}_{\text{safe}}^{m}(Q_S)$, otherwise, for any $m \in \mathbb{N}$.

$d\text{Chase}_{\text{safe}}(Q_S) = \bigcup_{m \in \mathbb{N}} d\text{Chase}_{\text{safe}}^{m}(Q_S)$

The termination condition for safe $d\text{Chase}$ computation can be implemented using the following conditional: If there exists $m$ s.t.

$d\text{Chase}_{\text{safe}}^{m+1}(Q_S) = d\text{Chase}_{\text{safe}}^{m}(Q_S)$; then $d\text{Chase}_{\text{safe}}(Q_S) = d\text{Chase}_{\text{safe}}^{m}(Q_S)$.

Similarly, $d\text{Chases}$ $d\text{Chase}_{\text{safe}}(Q_S)$ and $d\text{Chase}_{\text{csafe}}(Q_S)$ are defined for $\text{msafe}$ and $\text{csafe}$ quad-systems, respectively.

The following theorem shows that the procedure above described for detecting unsafe quad-systems is sound and complete:

$\textbf{Theorem 4.13.}$ For any quad-system $Q_S = (Q_C, R)$, the quad $\text{unSafe}$ (resp. $\text{unMSafe}$, resp. $\text{unCSafe}$) $\in d\text{Chase}_{\text{safe}}(Q_S)$ (resp. $d\text{Chase}_{\text{msafe}}(Q_S)$, resp. $d\text{Chase}_{\text{csafe}}(Q_S)$), iff $Q_S$ is unsafe (resp. $\text{msafe}$, resp. $\text{csafe}$).

It should be noted that for any quad-system $Q_S = (Q_C, R)$, $d\text{Chase}_{\text{safe}}(Q_S)$ (resp. $d\text{Chase}_{\text{msafe}}(Q_S)$, resp. $d\text{Chase}_{\text{csafe}}(Q_S)$) is a finite set and hence the iterative procedure which we described earlier terminates, regardless of whether $Q_S$ is safe (resp. $\text{msafe}$, resp. $\text{csafe}$) or not. This is because if $Q_S$ is safe (resp. $\text{msafe}$, resp. $\text{csafe}$), then, as we have seen before, there exists a double exponential bound on number of quads in its $d\text{Chase}$. Hence, there is an iteration in which no new quad is generated, which leads
to stopping of computation. Otherwise, if $QS_C$ is unsafe (resp. m-safe, resp. c-safe), then from theorem 4.13, we know that the quad unSafe (resp. unM Safe, resp. unCSafe) gets generated in $d\text{Chase}^\text{safe}(QS_C)$ (resp. $d\text{Chase}^\text{m-safe}(QS_C)$, resp. $d\text{Chase}^\text{c-safe}(QS_C)$). This implies that there exists an iteration $m$ s.t. the quad unSafe (resp. unM Safe, resp. unCSafe) is in $d\text{Chase}^\text{safe}(QS_C)$ (resp. $d\text{Chase}^\text{m-safe}(QS_C)$, resp. $d\text{Chase}^\text{c-safe}(QS_C)$). W.l.o.g. let $m$ be the first such iteration. This means that there exists a BR $r_i \in R$ with head $\text{head}(r_i)(\vec{x}, \vec{y})$, assignment $\mu$ s.t. applicable$_{\text{augS}(R)}(r_i, \mu, d\text{Chase}^\text{safe}(QS_C))$ (resp. applicable$_{\text{augM}(R)}(r_i, \mu, d\text{Chase}^\text{m-safe}(QS_C))$, resp. applicable$_{\text{augC}(R)}(r_i, \mu, d\text{Chase}^\text{c-safe}(QS_C))$) holds. By construction, since $\text{head}(r_i)$ is not generated, and instead the quad unSafe (resp. unM Safe, resp. unCSafe) is generated, applicable$_{\text{augS}(R)}(r_i, \mu, d\text{Chase}^\text{safe}(QS_C))$ (resp. applicable$_{\text{augM}(R)}(r_i, \mu, d\text{Chase}^\text{m-safe}(QS_C))$, resp. applicable$_{\text{augC}(R)}(r_i, \mu, d\text{Chase}^\text{c-safe}(QS_C))$) holds yet again. This means that the termination condition is satisfied at iteration $m+1$, and hence computation stops. Note that regardless of whether a given quad-system is safe (resp. m-safe, resp. c-safe) or not, the number of safe (resp. m-safe, resp. c-safe) dChase iterations is double exponentially bounded in the size of the quad-system.

Hence, after running procedure described above, if the quad unSafe (resp. unM Safe, resp. unCSafe) is not generated, then its safe (resp. m-safe, resp. c-safe) dChase itself can be used for CCQ answering, as in such a case the standard dChase is contained in safe (resp. m-safe, resp. c-safe) dChase, and all the quads generated for accounting information has the context identifier $c_c$. Hence, for any safe (resp. m-safe, resp. c-safe) quad-system, for any boolean CCQ that does not contain quad patterns of the form $c_c: (s, p, o)$, dChase entails CCQ iff safe (resp. m-safe, resp. c-safe) dChase entails CCQ.

5. Range Restricted Quad-Systems: Restricting to Range Restricted BRs

In this section, we investigate the complexity of CCQ entailment over quad-systems, whose BRs do not have existentially quantified variables. Such BRs are of the form:

$$c_1 : t_1(\vec{x}, \vec{z}) \land ... \land c_n : t_n(\vec{x}, \vec{z}) \rightarrow c'_1 : t'_1(\vec{x})$$

$$c'_1 : t'_1(\vec{x}) \land ... \land c'_m : t'_m(\vec{x})$$

Note that any set of BRs $R$ of the form above can be replaced by semantically equivalent set $R'$, s.t. each $r \in R'$ is the form:

$$c_1 : t_1(\vec{x}, \vec{z}), ..., c_n : t_n(\vec{x}, \vec{z}) \rightarrow c'_1 : t'_1(\vec{x})$$

Also $\|R'\|$ is at most quadratic in $\|R\|$, and hence, w.l.o.g. we assume that each $r \in R$ is of the form (4). Borrowing the parlance from the $\forall \exists$ rules setting, where rules whose variables in the head part are contained in the variables in the body part are called range restricted rules [14], we call such BRs range restricted (RR) BRs. We call a quad-system whose BRs are all of RR-type, a RR quad-system. Since there exists no existentially quantified variables in BRs of a RR quad-system, no Skolem blank nodes are produced during dChase computation. Hence, there can be no violation of the safety/m-safe/c-safe condition in section 4, and hence, the class of RR quad-systems are contained in the class of safe/m-safe/c-safe quad-systems, and is also a FCA. Of course, this containment is strict as any quad-system that contains a BR with an existential variable is not RR. We in the following see that restricting to RR BRs, size of the dChase becomes polynomial w.r.t. size of the input quad-system, and the complexity of CCQ entailment further reduces compared to safe/m-safe/c-safe quad-systems.

**Lemma 5.1.** For any RR quad-system $QS_C = \langle Q, R \rangle$, the following holds: (i) $\|d\text{Chase}(QS_C)\| = O(\|QS_C\|)$

(ii) $d\text{Chase}(QS_C)$ can be computed in EXPTIME.

(iii) If $\|R\|$ is fixed to be a constant, $d\text{Chase}(QS_C)$ can be computed in PTIME.

**Proof.** (i) Note that the number of constants in $QS_C$ is roughly equal to $\|QS_C\|$. As no existential variable occur in any RR BR in a RR quad-system $QS_C$, the set of constants $C(d\text{Chase}(QS_C))$ is contained in $C(QS_C)$. Since each $c: (s, p, o) \in d\text{Chase}(QS_C)$ is s.t. $c, s, p, o \in C(QS_C)$, $d\text{Chase}(QS_C) = O(\|C(QS_C)\|)^4$. Hence, $\|d\text{Chase}(QS_C)\| = O(\|C(QS_C)\|^4) = O(\|QS_C\|^4)$.

(ii) Since from (i) $\|d\text{Chase}(QS_C)\| = O(\|QS_C\|^4)$, and in each iteration of the dChase at least one new quad should be added, the number of iterations cannot exceed $O(\|QS_C\|^4)$. Since by lemma 3.2, each iteration i of dChase computation requires $O(|R| \times |d\text{Chase}_{i-1}(QS_C)|^{4^i})$ time, where $rs = \max_{r \in R}|r|$, and $rs \leq \|QS_C\|$, time required for each iteration is of the order $O(2^{|QS_C|})$ time. Although the number of iterations is a polynomial, each iteration requires an exponential amount of time w.r.t $\|QS_C\|$. 


For any formula $\phi$ the quad-system $QS_C Q$ over a quad-system whose set of schema triples, the set of pure 3Horn formulas to CCQ entailment problem we, in the following, reduce the satisfiability problem $P$ can be converted to $\Phi$. Any 3Horn formula $\Phi$ of the form (5), where $P_1 \land \ldots \land P_n \rightarrow P_{n+1}$ (5)

where $P_i$, for $1 \leq i \leq n + 1$, are are propositional variables or constants $t, P$, that represents true and false, respectively. Note that for any propositional variable $P$, the fact that “$P$ holds” is represented by the formula $t \rightarrow P$, and “$P$ does not hold” is represented by the formula $P \rightarrow f$. A 3Horn formula is a formula of the form (5), where $1 \leq n \leq 2$. Note that any (set of) Horn formula(s) $\Phi$ can be transformed in polynomial time to a polynomially sized set $\Phi'$ of 3Horn formulas, by introducing auxiliary propositional variables s.t. $\Phi$ is satisfiable iff $\Phi'$ is satisfiable. A pure 3Horn formula is a 3Horn formula of the form 5, where $n = 2$. Any 3Horn formula $\phi$ that is not pure can be trivially converted to equivalent pure form by appending a $\land t$ on the body part of $\phi$. For instance, $P \rightarrow Q$, can be converted to $P \land t \rightarrow Q$. Hence, w.l.o.g. we assume that any set of 3Horn formulas is pure, and is of the form:

$$P_1 \land P_2 \rightarrow P_3$$ (6)

We, in the following, reduce the satisfiability problem of pure 3Horn formulas to CCQ entailment problem over a quad-system whose set of schema triples, the set of BRs, and the CCQ $CQ$ are all fixed.

For any set of pure Horn formulas $\Phi$, we construct the quad-system $QS_C = (Q_C, R)$, where $C = \{c_t, c_f\}$. For any formula $\phi \in \Phi$ of the form (6), $Q_C$ contains a quad $c_f: (P_1, P_2, P_3)$. In addition $Q_C$ contains a quad $c_t: (t, \text{rdf:type}, T)$. $R$ is the singleton that contains only the following fixed BR:

$$c_t: (x_1, \text{rdf:type}, T), c_t: (x_2, \text{rdf:type}, T),$$

$$c_f: (x_1, x_2, x_3) \rightarrow c_t: (x_3, \text{rdf:type}, T)$$

Let the $CQ$ be the fixed query $c_t: (f, \text{rdf:type}, T)$.

Now, it is easy to see that $QSC \models CQ$, iff $\Phi$ is not satisfiable.

Theorem 5.2. Data complexity of CCQ entailment over RR quad-systems is PTIME-complete.

Proof. (Membership) Follows from the membership in P of data complexity of CCQ entailment for safe quad-systems, whose expressivity subsumes the expressivity of RR quad-systems (Theorem 4.11).

(Hardness) In order to prove P-hardness, we reduce a well known P-complete problem, 3HornSat, i.e. the satisfiability of propositional Horn formulas with at most 3 literals. Note that a (propositional) Horn formula is a propositional formula of the form:

$$P_1 \land \ldots \land P_n \rightarrow P_{n+1}$$ (5)

where $P_i$, for $1 \leq i \leq n + 1$, are propositional variables or constants $t, f$, that represents true and false, respectively. Note that for any propositional variable $P$, the fact that “$P$ holds” is represented by the formula $t \rightarrow P$, and “$P$ does not hold” is represented by the formula $P \rightarrow f$. A 3Horn formula is a formula of the form (5), where $1 \leq n \leq 2$. Note that any (set of) Horn formula(s) $\Phi$ can be transformed in polynomial time to a polynomially sized set $\Phi'$ of 3Horn formulas, by introducing auxiliary propositional variables s.t. $\Phi$ is satisfiable iff $\Phi'$ is satisfiable. A pure 3Horn formula is a 3Horn formula of the form 5, where $n = 2$. Any 3Horn formula $\phi$ that is not pure can be trivially converted to equivalent pure form by appending a $\land t$ on the body part of $\phi$. For instance, $P \rightarrow Q$, can be converted to $P \land t \rightarrow Q$. Hence, w.l.o.g. we assume that any set of 3Horn formulas is pure, and is of the form:

$$P_1 \land P_2 \rightarrow P_3$$ (6)

We, in the following, reduce the satisfiability problem of pure 3Horn formulas to CCQ entailment problem over a quad-system whose set of schema triples, the set of BRs, and the CCQ $CQ$ are all fixed.

For any set of pure Horn formulas $\Phi$, we construct the quad-system $QS_C = (Q_C, R)$, where $C = \{c_t, c_f\}$. For any formula $\phi \in \Phi$ of the form (6), $Q_C$ contains a quad $c_f: (P_1, P_2, P_3)$. In addition $Q_C$ contains a quad $c_t: (t, \text{rdf:type}, T)$. $R$ is the singleton that contains only the following fixed BR:

$$c_t: (x_1, \text{rdf:type}, T), c_t: (x_2, \text{rdf:type}, T),$$

$$c_f: (x_1, x_2, x_3) \rightarrow c_t: (x_3, \text{rdf:type}, T)$$

Let the $CQ$ be the fixed query $c_t: (f, \text{rdf:type}, T)$.

Now, it is easy to see that $QSC \models CQ$, iff $\Phi$ is not satisfiable.

Theorem 5.3. Combined complexity of CCQ entailment over RR quad-systems is EXPTIME-complete.

Proof. (Membership) By lemma 5.1, for any RR quad-system $QSC$, its dChase $dChase(QSC)$ can be computed in EXPTIME. Also by lemma 5.1, its dChase size $|dChase(QSC)|$ is a polynomial w.r.t to $|QSC|$. Since a boolean CCQ $CQ()$ can naively be evaluated by grounding the set of constants in the dChase to the variables in the $CQ()$, and then checking if any of these groundings are contained in $dChase(QSC)$. The number of such groundings can at most be $|dChase(QSC)| \cdot |CQ()|$. Since $|dChase(QSC)|$ is a polynomial in $|QSC|$, there are an exponential number of groundings w.r.t $|CQ()|$. Since containment of each of these groundings can be checked in time polynomial w.r.t. the size of $dChase(QSC)$, and since $|dChase(QSC)|$ is a polynomial w.r.t. $|QSC|$, the time complexity of CCQ entailment is in EXPTIME.

(Hardness) For EXPTIME-hardness, since we already saw in subsection 4.1 that with appropriate BRs and triple patterns one can simulate a DTM. The proof can slightly be modified to simulate an EXPTIME DTM. The steps in the proof is same as the one in Dantsin et al. [25], where EXPTIME-hardness of function-free Horn logic programs (Datalog) are shown.

5.1. Restricted RR Quad-Systems

We call those quad-systems with BRs of form (4) with a fixed bound on $n$ as restricted RR quad-systems. They can be further classified as linear, quadratic, cubic,..., quad-systems, when $n = 1, 2, 3, ...$, respectively.

Theorem 5.4. Data complexity of CCQ entailment over restricted RR quad-systems is P-complete.

Proof. The proof is same as in theorem 5.2, since the size of BRs are fixed to constant.
Theorem 5.5. Combined complexity of CQ entailment over restricted RR quad-systems is NP-complete.

Proof. Let the problem of deciding if $QS_C \models CQ(\lambda)$ be called DP$^\prime$.

(Membership) for any $QS_C$ whose rules are of restricted RR-type, size of any $r \in R$ is a constant. Hence, by lemma 3.2, any dChase iteration can be computed in PTIME. Since, number of iterations are also polynomial in $\|QS_C\|$, $dChase(QS_C)$ can be computed in PTIME in the size of $QS_C$ and $dChase(QS_C)$ has a polynomial number of constants.

Hence, if we guess an assignment $\mu$ for all the existential variables in CCQ $CQ(\lambda)$, to the set of constants in $dChase(QS_C)$. Then, one can evaluate the CCQ, by checking if $c: (s, p, o) \in dChase(QS_C)$, for each $c: (s, p, o) \in CQ(\lambda) \mu$, which can be done in time $O(\|CQ\| \cdot dChase(QS_C))$, and hence is in non-deterministic PTIME, which implies that DP$^\prime$ is in NP.

(Hardness) We show that DP$^\prime$ is NP-hard, by reducing the well known NP-hard problem, 3-colorability to DP$^\prime$. Given a graph $G = (V, E)$, where $V = \{v_1, ..., v_n\}$ is the set of nodes, $E \subseteq V \times V$ is the set of edges, 3-colorability problem is to decide if there exists a labeling function $l: V \rightarrow \{r, b, g\}$ that assigns each $v \in V$ to an element in $\{r, b, g\}$ such that:

$$\forall v, v' \in E \ (l(v) \neq l(v'))$$

for each $(v, v') \in E$, is satisfied.

One can construct a quad-system $QS_C = (\{c\}, \emptyset)$, where $\text{graph}_{Q_C}(c)$ has the following triples:

$$\{(r, \text{edge}, b), (r, \text{edge}, g), (b, \text{edge}, g), (b, \text{edge}, r), (g, \text{edge}, r), (g, \text{edge}, b)\}$$

Let $CQ$ be the boolean CCQ: $\exists x_1, ..., x_n \bigwedge_{(v, v') \in E} \left[ c: (v, \text{edge}, v') \wedge c: (v', \text{edge}, v') \right]$. Then, it can be seen that $G$ is 3-colorable, iff $QS_C \models CQ$. □

6. Quad-Systems and Forall-Existential rules: A formal comparison

In this section, we formally compare the formalism of quad-systems with forall-existential ($\forall\exists$) rules. These are also called Tuple generating dependencies (Tgds)/Datalog+ rules. $\forall\exists$ rules are a fragment of first order logic in which every formula is restricted to a certain syntactic form. A $\forall\exists$ rule is a first order formula of the form:

$$\forall x \forall z \exists y \left[ p_1(x, z) \wedge \ldots \wedge p_n(x, z) \rightarrow \exists y' p'_1(x, y') \wedge \ldots \wedge p'_m(x, y') \right]$$

where $x, y, z$ are vectors of variables s.t. $\{x\}, \{y\}$ and $\{z\}$ are pairwise disjoint, $p_1(x, z)$, for $1 \leq i \leq n$ are predicate atoms whose variables are from $x$ or $z$, $p'_1(x, y)$, for $1 \leq i \leq m$ are predicate atoms whose variables are from $x$ or $y$. We, for short, occasionally note a $\forall\exists$ rule of the form (7) as $\phi(x, z) \rightarrow \psi(x, y)$, where $\phi(x, z) = \{p_1(x, z), ..., p_n(x, y)\}$, $\psi(x, y) = \{p'_1(x, y), ..., p'_m(x, y)\}$. A set of $\forall\exists$ rules is called a $\forall\exists$ rule set. In the realm of $\forall\exists$ rule sets, a conjunctive query (CQ) is an expression of the form:

$$\exists y p_1(x, y) \wedge \ldots \wedge p_r(x, y)$$

where $p_1(x, y)$, for $1 \leq i \leq r$ are predicate atoms over vectors $x$ or $y$. A boolean CQ is defined as usual.

The DP of whether, for a $\forall\exists$ rule set $P$ and a CQ $Q$, if $P \models_{\forall\exists} Q$ is called the CQ EP, where $\models_{\forall\exists}$ is the standard first order logic entailment relation.

Notice that for any quad-system $Q_C = \{c_1: (s_1, p_1, o_1), ..., c_n: (s_r, p_r, o_r)\}$, let $\mathbf{r}_{Q_C}$ be the BR:

$$\exists y_{b_1}, ..., y_{b_q} c_1: (s_1, p_1, o_1)[\mathbf{r}_{Q_C}]$$

$$= \ldots \wedge c_r: (s_r, p_r, o_r)[\mathbf{r}_{Q_C}]$$

where $\{b_1, ..., b_q\}$ is the set of blank nodes in $Q_C$, and $\mathbf{r}_{Q_C}$ is the substitution function $\{b_i \mapsto y_{b_i} \}_{i=1,...,q}$ that assigns each blank-node to a fresh existentially quantified variable. It can be noted the quad-systems $\{Q_C, R\}$ and $\{\emptyset, R \cup \{r_{Q_C}\}\}$ are semantically equivalent.

The following property gives the relation between CCQ entailment of unrestricted quad-systems and standard first order CQ entailment of $\forall\exists$ rule sets.

Property 6.1. Suppose $\tau_\forall$ be the function from the set of quad patterns to the set of ternary atoms s.t. for any quad-pattern $c: (s, p, o), \tau_\forall(c): (s, p, o) = c(s, p, o)$.

Let $\tau_{br}$ be a function from the set of BRs to the set of $\forall\exists$ rules, s.t. for any $BR$ of the form (2):

$$\tau_{br}(r) = \forall x \forall y \exists z \left[ \tau_c(c_1: t_1(x, z)) \wedge \ldots \wedge \tau_c(c_n: t_n(x, z)) \rightarrow \exists y_1 \tau(c'_1: t'_1(x, y_1)) \wedge \ldots \wedge \tau(c'_m: t'_m(x, y_m)) \right]$$

And, let $\tau$ be the function s.t. for any quad-system $QS_C = (Q_C, R), \tau(QS_C) = \tau_{br}(R) \cup \{\tau_{br} (r_{Q_C})\}$, where $\tau_{br}(R) = \bigcup_{r \in R} \tau_{br}(r)$.

Also, let $\tau_{ccq}$ be a function defined from the set of boolean CCQs to the set of boolean CQs, s.t. for any boolean CCQ $QC = \exists y c_1: t_1(a, y) \wedge \ldots \wedge c_r: t_r(a, y), \tau_{ccq}(QC)$ is:

$$\exists y \tau_c(c_1: t_1(a, y)) \wedge \ldots \wedge \tau(c_r: t_r(a, y))$$

then, for any quad-system $QS_C$, $CCQ, QS_C \models QC$ iff $(QC) \models_{\forall\exists} \tau_{ccq}(QC)$.
Proof. Notice that every context \( c \in C \) becomes a ternary predicate symbol in the resulting translation. Also, \( \tau(QSc) \) is a \( \forall \exists \) rule set, and for any CCQ \( CQ \), \( \tau_{ccq}(CQ) \) is a CQ.

In order to construct the restricted chase for \( \tau(QSc) \), suppose that \( \neg \chi \) is also extended to set of instances s.t. for any two quad-graphs \( Qc, Qc' \), \( Qc \prec Qc' \) iff \( \tau_{q}(Qc) \prec \tau_{q}(Qc') \). Suppose \( \neg \) is extended similarly to set of instances. Also assume that during the construction of standard chase \( d\text{Chase}(QSc) \) of \( \tau(QSc) \), for any application of a \( \tau_{br}(r) \) with existentially quantified variables, with \( r \in R \), suppose the skolem blank nodes generated in \( d\text{Chase}(QSc) \) follow the same order as they are generated in \( d\text{Chase}(QSc) \).

Also let us extend the rule applicability function to the \( \forall \exists \) rules setting s.t. for any set of BRs \( R \), for any \( r \in R \), quad-graph \( Qc' \), assignment \( \mu \), applicable\( \tau_{r}(r, \mu, Qc') \) iff applicable\( \tau_{br}(r, \mu, Qc') \).

Now \( d\text{Chase}_{e}(QSc) \) is a CQ and a CCQ \( \tau_{ccq}(CQ) \) is a \( \forall \exists \) rule set, and for any CCQ \( \tau_{ccq}(CQ) \) is a CQ.

\[ \tau(QSc) \Rightarrow_{\text{fol}} \tau_{ccq}(CQ). \]

\[ \neg \chi \] is a CQ.

\[ \tau_{ccq}(CQ) \] follows the same order as they are generated in \( d\text{Chase}(QSc) \).

Theorem 6.2. There exists a polynomial time translation function \( \tau \) (resp. \( \tau_{ccq} \)) from the set of unrestricted quad-systems (resp. CCQs) to the set of \( \forall \exists \) rule sets (resp. \( \forall \exists \) rule sets). Also let us extend the rule applicability function to the \( \forall \exists \) rules setting s.t. for any set of BRs \( R \), for any \( r \in R \), quad-graph \( Qc' \), assignment \( \mu \), applicable\( \tau_{r}(r, \mu, Qc') \) iff applicable\( \tau_{br}(r, \mu, Qc') \).

Theorem 6.3. For quad-systems, the EPs: (i) quad EP, (ii) quad-graph EP, (iii) BR EP, (iv) BRs EP, (v) Quad-System EP, and (vi) CCQ EP are polynomially reducible to entailment of \( \forall \exists \) rule sets.

A \( \forall \exists \) rule set \( P \) is said to be a ternary \( \forall \exists \) rule set, iff all the predicate symbols in the vocabulary of \( P \) is of arity less than or equal to three. \( P \) is a purely ternary rule set, iff all the predicate symbols in the vocabulary of \( P \) is of arity three. Similarly, a (purely) ternary CQ is defined. The following property gives the relation between the CQ entailment problem of \( \forall \exists \) rule sets and CCQ EP of unrestricted quad-systems.

Theorem 6.4. There exists a polynomial time translation function \( \nu \) (resp. \( \nu_{ccq} \)) from ternary \( \forall \exists \) rule sets (resp. ternary CCQs) to unrestricted quad-systems (resp. CCQs) s.t. for any \( \forall \exists \) rule set \( P \) and a CCQ \( Q \), \( P \Rightarrow_{\text{fol}} CQ \) iff \( \nu(P) \Rightarrow_{\text{fol}} \nu_{ccq}(Q) \).

Proof. Notice that the CQ EP of any ternary \( \forall \exists \) rule set \( P \), whose set of predicate symbols is \( P \), and \( CQ \) over \( P \), can polynomially reduced to the CQ EP of a purely ternary rule set \( P' \) and purely ternary CCQ \( Q' \), by the following transformation function \( \chi \). Let \( \Box \) be an ad-hoc fresh URI: \( \chi \) is s.t. for any ternary atom \( c(s, p, o) \), \( \chi(c(s, p, o)) = c(s, p, o) \). For any binary atom \( c(s, p) \), \( \chi(c(s, p)) = c(s, p, \Box) \), and for any unary atom \( c(s) \), \( \chi(c(s)) = c(s, \Box, \Box) \). For any \( \forall \exists \) rule \( r \) of the form \( (7) \),

\[ \chi(r) = \forall \exists \forall \exists [\chi(p_{1}(\vec{x}, \vec{z})) \land \ldots \land \chi(p_{n}(\vec{x}, \vec{z})) \land \exists \forall \exists [\chi(p'_{1}(\vec{x}, \vec{y})) \land \ldots \land \chi(p'_{m}(\vec{x}, \vec{y}))]] \]

And, for any \( \forall \exists \) rule set \( P \), \( \nu(P) = \bigcup_{r \in P} \chi(r) \).

For any CQ \( Q \), \( \nu(Q) \) is similarly defined. Note that for any ternary \( \forall \exists \) rule set \( P \), ternary CCQ \( CQ \), \( \nu(P) \) is purely ternary, and \( P \Rightarrow_{\text{fol}} CQ \) iff \( \nu(P) \Rightarrow_{\text{fol}} \nu(Q) \).

Also, it can straightforwardly see that \( \tau_{ccq}^{-1}(\chi(P)) \) is a set of BRs (resp. CCQ). Suppose, \( \nu(P) \) is s.t. \( \nu(P) = QSc = (\emptyset, \tau_{br}^{-1}(\chi(P))) \). Intuitively, \( C \) contains a context identifier \( c \) for each predicate symbol \( c \in P \). Also suppose, \( \nu_{ccq}(Q) = \tau_{ccq}^{-1}(\chi(Q)) \). Notice that \( \nu_{ccq}(Q) \) is CCQ. It can straightforwardly see that \( \nu \) and \( \nu_{ccq} \) can be computed in polynomial time, and \( P \Rightarrow_{\text{fol}} CQ \).

Thanks to the theorem 6.2 and theorem 6.4, the following theorem immediately holds:

Theorem 6.5. The CCQ EP over quad-systems is polynomially equivalent to CQ EP over ternary \( \forall \exists \) rule sets.
By virtue of the theorem above, we derive the following property:

**Property 6.6.** For quad-systems, the Quad EP, Quad-graph EP, BR(s) EP, and Quad-system EP are polynomially reducible to CCQ EP.

**Proof.** The following claim is a folklore in the realm of ∀∃ rules.

**Claim (1)** The ∀∃ rule set EP is polynomially reducible to CQ EP.

Reducibility of ∀∃ rule EP to CQ EP is a folklore in the realm of ∀∃ rules. For a formal proof, we refer the reader to Baget et al. [14], where it is shown that the ∀∃ rule EP is polynomially reducible to fact (a set of instances) EP, and fact EP are equivalent to CQ EP. Also, Cali et al [33] shows that CQ containment problem, which is equivalent to ∀∃ rule EP, is reducible to CQ EP. Since a ∀∃ rule set is a set of ∀∃ rules, by using a series of oracle calls to a function that solves the ∀∃ rule EP, we can define a function for deciding ∀∃ rule set entailment. Hence, the claim holds.

(a) Thanks to translation functions τ, τbr defined earlier, s.t. for any quad-system QS_C, quad-graph Q'_C, QS_C \models Q'_C, iff \( \tau(QS_C) \models_{\tau} \tau_{br}(rQ'_C) \), we can infer that quad-graph EP is polynomially reducible to ∀∃ rule set EP. Applying claim 1, it follows the quad-graph EP over quad-systems is polynomially reducible to CQ EP over ∀∃ rule sets. By theorem 6.4, we can deduce that quad-graph EP is polynomially reducible to CCQ EP.

(b) By the translation functions τ and τbr, defined earlier, s.t. for any quad-system QS_C, a set of BRs R, QS_C \models R iff \( \tau(QS_C) \models_{\tau} \tau_{br}(R) \), we can infer that BRs EP is polynomially reducible to ∀∃ rule set EP. Similar to (a) above, we deduce that BRs EP is polynomially reducible to CCQ EP.

From (a) and (b), it follows that Quad-system EP is reducible to CCQ EP. □

Having seen that the CCQ EP over quad-systems is polynomially equivalent to CQ EP over ternary ∀∃ rule sets, we now compare some of the well known techniques used to ensure decidability of CQ entailment in the ∀∃ rules settings to the decidability techniques for quad-systems that we saw earlier in the previous sections. Note that since all the quad-system classes we proposed in this paper are FECs, for a judicious comparison, the ∀∃ rule classes to which we compare are classes which have a finite chase property. We compare to the following three well known classes: (i) Weakly Acyclic rule sets (WA), (ii) Jointly Acyclic rule sets (JA), and (iii) Model Faithful Acyclic ∀∃ rule sets (MFA). The following property is well known in the realm of ∀∃ rules:

**Property 6.7.** For the any ∀∃ rule set \( \mathbb{P} \), the following holds:

1. If \( \mathbb{P} \in \text{WA} \), then \( \mathbb{P} \in \text{JA} \) (from [36]).
2. If \( \mathbb{P} \in \text{JA} \), then \( \mathbb{P} \in \text{MFA} \) (from [31]).
3. WA \( \subseteq \) JA \( \subset \) MFA (from [36] and [31]).

Note that a description of few other ∀∃ rule classes that do not have the finite chase property, but still enjoy decidability of CQ entailment are given in the related work.

6.1. **Weak Acyclicity**

Weak acyclicity [23,24] is a popular technique used to detect whether a ∀∃ rule set has a finite chase, thus ensuring decidability of query answering. The set WA represents class of ternary ∀∃ rule sets that have the weak acyclicity property.

For any predicate atom \( p(t_1, \ldots, t_n) \), an expression \( \langle p, i \rangle \), for \( i = 1, \ldots, n \) is called a position of \( p \). In the above case, \( t_1 \) is said to occur at position \( \langle p, 1 \rangle \), \( t_2 \) at \( \langle p, 2 \rangle \), and so on. For a set of ∀∃ rules \( \mathbb{P} \), its dependency graph is a graph whose nodes are positions of predicate atoms in \( \mathbb{P} \); for each \( r \in \mathbb{P} \) of the form (7), and for any variable \( x \) occurring in position \( \langle p, i \rangle \) in head of \( r \):

1. if \( x \) is universally quantified and \( x \) occurs in body of \( r \) at position \( \langle p', j \rangle \), then there exists an edge from \( \langle p', j \rangle \) to \( \langle p, i \rangle \)
2. if \( x \) is existentially quantified, then for any universally quantified variable \( x' \) occurring in head of \( r \), with \( x' \) also occurring in the body of \( r \) at position \( \langle p', j \rangle \), there exists a special edge from \( \langle p', j \rangle \) to \( \langle p, i \rangle \).

\( \mathbb{P} \) is called weakly acyclic, iff its dependency graph does not contain cycles going through a special edge. For any ∀∃ rule set \( \mathbb{P} \), if \( \mathbb{P} \) is WA, then its chase is finite, and hence CQ EP is decidable. Note that the nodes in the dependency graph that has incoming special edges corresponds to the positions of predicates where new values are created due to existential variables, and the normal edges capture the propagation of constants from one predicate position to another predicate position. In this way, absence of cycles involving special edges ensures that newly created Skolem
blank nodes are not recursively used to create other new Skolem blank nodes in the same position, leading to termination of chase computation.

Example 6.8. Let us revisit the quad-system $QS_C = (Q_C, R)$ mentioned in example 4.3, whose dependency graph is shown in Fig. 3. Note that the $QS_C$ is unsafe, since its dChase contains a Skolem blank-node $\_ : b_1$, which has as descendant another Skolem blank node $\_ : b_1$, with the same origin context $c_2$ (see Fig. 1). However, it can be seen from Fig. 3 that the dependency graph of $\tau(QS_C)$ does not contain any directed cycle involving special edges. Hence, $\tau(QS_C)$ is weakly acyclic.

It turns out that there exists no inclusion relationship between the classes WA and CSAFE in either directions, i.e. WA \( \nsubseteq \) CSAFE (from example 6.8), and CSAFE \( \nsubseteq \) WA (from the fact that WA \( \subset \) JA, and example 6.9 below). Whereas WA \( \subset \) MSAFE, since WA \( \subset \) MFA and MFA = MSAFE (theorem 6.10).

6.2. Joint Acyclicity

Joint acyclicity [36] extends weak acyclicity, by also taking into consideration the join between variables in body of \( \forall \exists \) rules while analyzing the rules for acyclicity. The set JA represents the class of all ternary \( \forall \exists \) rule sets that have the joint acyclicity property. A \( \forall \exists \) rule set $P$ is said to be renamed apart, if for any $r \neq r' \in R$, $V(r) \cap V(r') = \emptyset$. Since any set of rules can be converted to an equivalent renamed apart one by simple variable renaming, we assume that any rule set $P$ is renamed apart. Also for any $r \in P$ and for a variable $y$, let $Pos^s_P(r, y)$ (Pos$^t_P(r, y)$) be the set of positions in which $y$ occurs in the head (resp. body) of $r$. For any \( \forall \exists \) rule set $P$ and an existentially quantified variable $y$ occurring in a rule in $P$, we define $Mov^s_P(y)$ to be as follows:

- $Pos^s_P(y) \subseteq Mov^s_P(y)$, if $y$ occurs in $r$;
- $Pos^t_P(x) \subseteq Mov^t_P(y)$, if $x$ is a universally quantified variable and $Pos^t_P(x) \subseteq Mov^t_P(y)$;

for any $r \in P$. The existential dependency graph of a (renamed apart) set of rules $P$ is a graph whose nodes are the existentially quantified variables in $P$. There exists an edge from a variable $y$ to $y'$, if $r$ is a rule in which $y'$ occurs and there exists a universally quantified variable $x$ in the head (and body) of $r$ s.t. $Pos^t_P(x) \subseteq Mov^t_P(y)$. A \( \forall \exists \) rule set $P$ is jointly acyclic, if its existential dependency graph is acyclic. Analyzing the containment relationships, it happens to be the case that JA \( \nsubseteq \) CSAFE (since WA \( \subset \) JA, and eg. 6.8). Also example 6.9 shows us that CSAFE \( \nsubseteq \) JA. However JA \( \subset \) MSAFE, since JA \( \subset \) MFA and MFA = MSAFE (theorem 6.10).

Example 6.9. Consider the quad-system $QS_C = (Q_C, R)$, where $Q_C = \{ c_1 : (a, b, c) \}$. Suppose $R$ is the following set:

$$R = \left\{ \begin{array}{l}
c_1 : (x_{11}, x_{12}, z_1) \rightarrow c_2 : (x_{11}, x_{12}, y_1) \ (r_1) \\
c_2 : (x_{21}, x_{22}, z_2), c_2 : (x_{22}, x_{21}, x_{23}) \\
c_3 : (x_{31}, x_{32}, x_{33}) \rightarrow c_1 : (x_{33}, x_{31}, x_{32}) \ (r_3)
\end{array} \right\}$$

Iterations during dChase construction are:

$$dChase_0(QS_C) = \{ c_1 : (a, b, c) \}$$
$$dChase_1(QS_C) = \{ c_1 : (a, b, c), c_2 : (a, b, \_ : b_1) \}$$
$$dChase(QS_C) = dChase_1(QS_C)$$

Note that the lone Skolem blank node generated is $\_ : b_1$, which do not have any descendants. Hence, by definition $QS_C$ is c-safe (m-safe). Now analyzing the BRs for joint acyclicity, we note that for the only existentially quantified variable $y_1$,

$$Mov_R(y_1) = \{ (c_2, 3), (c_3, 3), (c_1, 1) \}$$

Since the BR $r_1$ is which $y_1$ occurs contains the universally quantified variable $x_{11}$ in head of $r_1$ s.t. $Pos^t_B(x_{11}) \subseteq Mov_R(y_1)$, there exists a cycle from $y_1$ to $y_1$ itself in the existential dependency graph of $\tau(QS_C)$. Hence, by definition $\tau(QS_C)$ is not joint acyclic. Also since the class of weakly acyclic rules are contained in the class of jointly acyclic rule, it follows that $\tau(QS_C)$ is also not weakly acyclic.
6.3. Model Faithful Acyclicity (MFA)

MFA, proposed in Bernardo et al. [31], is an acyclicity technique that guarantees finiteness of chase and decidability of query answering, in the realm of \( \exists \forall \) rules. The set MFA denotes class of all ternary \( \exists \forall \) rule sets that is model faithfully acyclic. As far as we know, the MFA technique subsumes almost all other known techniques that guarantee a finite chase, in the \( \exists \forall \) rules settings. Obviously, \( \text{WA} \subset \text{JA} \subset \text{MFA} \).

For any \( \exists \forall \) rule \( r = \phi(r)(\vec{x}, \vec{z}) \rightarrow \psi(r)(\vec{x}, \vec{y}) \), for each \( y_j \in \{\vec{y}\} \), let \( Y_{\vec{y}}^j(\vec{y}_j) \) be a fresh unary predicate unique for \( y_j \) and \( r \); furthermore, let \( S \) be a fresh binary predicate. The transformation \( \text{mfa} \) of \( r \) is defined as:

\[
\text{mfa}(r) = \phi(r)(\vec{x}, \vec{z}) \rightarrow \psi(r)(\vec{x}, \vec{y}) \wedge \\
\bigwedge_{y_j \in \{\vec{y}\}} [Y_{\vec{y}}^j(\vec{y}_j) \wedge \bigwedge_{x_k \in \{\vec{x}\}} S(x_k, y_j)]
\]

Also let \( r_1 \) and \( r_2 \) be two additional rules defined as:

\[
S(x_1, z) \land S(z, x_2) \rightarrow S(x_1, x_2) \quad (r_1)
\]

\[
Y_{\vec{y}}^j(x_1) \land S(x_1, x_2) \land Y_{\vec{y}}^j(x_2) \rightarrow C \quad (r_2)
\]

where \( C \) is a fresh nullary predicate. For any set of \( \exists \forall \) rules \( \mathbb{P} \), let \( \text{ad}(\mathbb{P}) \) be the union of \( r_1 \) with the set of rules obtained by instantiating \( r_2 \), for each \( r \in \mathbb{P} \), for each existential variable \( y_j \) in \( r \). For a set of \( \exists \forall \) rules \( \mathbb{P} \), \( \text{mfa}(\mathbb{P}) = \bigcup_{r \in \mathbb{P}} \text{mfa}(r) \cup \text{ad}(\mathbb{P}) \). A \( \exists \forall \) rule set \( \mathbb{P} \) is said to be MFA, iff \( \text{mfa}(\mathbb{P}) \models \text{fof } \mathbb{C} \). It was shown in Cuenca Grau et al. [31] that if \( \mathbb{P} \) is MFA, then \( \mathbb{P} \) has a finite chase, thus ensuring decidability of query answering. The following theorem establishes the fact that notion of msafety is equivalent to MFA, thanks to the polynomial time translations between quad-systems and ternary \( \exists \forall \) rule sets.

**Theorem 6.10.** Let \( \tau \) be the translation function from the set of unrestricted quad-systems to the set of ternary \( \exists \forall \) rule sets, as defined in property 6.1, then, for any quad-system \( QS_C = (Q_C, R_C) \), \( QS_C \) is msafe iff \( \tau(QS_C) \) is MFA.

**Proof.** (outline) Recall that for \( \tau = \langle \tau_q, \tau_r \rangle \), where \( \tau_q \) is the quad translation function and \( \tau_r \) is the translation function from BRs to \( \exists \forall \) rules. Also, \( \tau(QS_C) = \tau_q(\text{dChase}_{\mathbb{C}}(\tau(QS_C))) = \text{Chase}_{\mathbb{C}}(\tau(QS_C)) \), for any \( \mathbb{C} \). We already saw that for such a transformation, the following property holds: for any \( m \in \mathbb{N} \), \( \tau_q(\text{dChase}_{\mathbb{C}}(\tau(QS_C))) = \text{Chase}_{\mathbb{C}}(\tau(QS_C)) \), and for any \( \mathbb{C} \).
easily be noticed that the technique of safety can be applied to ∀∃ rule sets of arbitrary arity, and can be used to extend currently established tools and systems that work on existing notions of acyclicity such as WA, JA, or MFA.

7. Related Work

Contexts and Distributed Logics Work on contexts gained its attention as early as in the 80s, as McCarthy [1] proposed context as a solution to the generality problem in AI. After this, various studies about logics of contexts mainly in the field of KR was done by Guha [17], Distributed First Order Logics by Ghidini et al. [16] and Local Model Semantics by Giunchiglia et al. [8]. Primarily in these works contexts are formalized as a first order/propositional theory and bridge rules were provided to inter-operate the various theories of contexts. Some of the initial works on contexts relevant to semantic web were the ones like Distributed Description Logics [5] by Borgida et al., and Context-OWL [7] by Bouquet et al., and the work of CKR [12,9] by Serafini et al. These were mainly logics based on DLs, which formalized contexts as OWL KBs, whose semantics is given using a distributed interpretation structure with additional semantic conditions that suits varying requirements. Compared to these works, the bridge rules we consider are much more expressive with conjunctions and existential variables that supports value/blank-node creation.

Temporal RDF/Annotated RDF Studies in extending standard RDF with dimensions such as time and annotations has already been accomplished. Gutierrez et al. in [38] tried to add a temporal extension to RDF and defines the notion of a ‘temporal rdf graph’, in which a triple is augmented to a quadruple of form $t(\langle s, p, o \rangle)$, where $t$ is a time point. Whereas annotated extensions to RDF and querying annotated graphs has been studied in Udrea et al. [39] and Straccia et al. [40]. Unlike the case of time, here the quadruple has the form: $a(\langle s, p, o \rangle)$, where $a$ is an annotation. The authors provide semantics, inference rules and query language that allows to express temporal/annotated queries. Although these approaches, in a way address contexts by means of time and annotations, the main difference in our work is that we provide the means to specify expressive bridge rules for inter-operating the reasoning between the various contexts.

∀∃ rules Works on extending DL KBs with Data- log like rules was studied by Horrocks et al. [28] giving rise to the SWRL[28] language. The related initiatives proposes a formalism using which one can mix a DL ontology with the Unary/Binary Datalog RuleML sub-languages of the Rule Markup Language, and hence enables horn-like rules to be combined with an OWL KB. Since SWRL is undecidable in general, studies on computable sub-fragments gave rise to works like Description Logic Rules [37], where the authors deal with rules that can be totally internalized by a DL knowledge base, and hence if the DL considered is decidable, then also is a DL+rules KB. The authors give various fragments of the rule bases like SROIQ rules, EL++ rules etc. and show that certain new constructs that are not expressible by plain DL can be expressed using rules although they are finally internalized into DL KBs. Unlike in our scenario, these works consider only horn rules with out existential variables.

∀∃ rules, TGDs, Datalog+- rules Query answering over rules with universal-existential quantifiers in the context of databases, where these rules are called Datalog+- rules/tuple generating dependencies (TGDs), was done by Beeri and Vardi [13] even in the early 80s, where the authors show that the query entailment problem, in general, is undecidable. However, recently many classes of such rules have been identified for which query answering is decidable. These classes (according to [14]) can broadly be divided into the following three categories: (i) bounded treewidth sets (BTS), (ii) finite unification sets (FUS), and (iii) finite extension sets (FES). BTS contains the classes of ∀∃ rule sets, whose models have bounded treewidth. Some of the important classes of these set are the linear ∀∃ rules [19], (weakly) guarded rules [33], (weakly) frontier guarded rules [14], and jointly frontier guarded rules [36]. BTS classes in general need not have a finite chase, and query answering is done by exploiting the fact that the chase is tree shaped, whose nodes (which are sets of instances) start replicating (upto isomorphism) after a while. Hence, one could stop the computation of the chase, once it can be made sure that any future iterations of chase can only produce nodes that are isomorphic to existing nodes. A deterministic algorithm for deciding query entailment for this class is provided in Thomazo et al. [15].

FUS classes includes the class of ‘sticky’ rules [34], atomic hypothesis rules in which body of each rule contains only a single atom, and also the class of linear ∀∃ rules. The approach used for query answering...
in FUS classes is to rewrite the input query w.r.t. to the $\forall \exists$ rule sets to another query that can be evaluated directly on the set of instances, s.t. the answers for the former query and latter query coincides. The approach is called the query rewriting approach. Compared to approaches proposed in this paper, these approaches do not enjoy the finite chase property, and is hence not conducive to materialization/forward chaining based query answering.

Unlike BTS and FUS, the FES classes are characterized by the finite chase property, and hence is most related to the techniques proposed in our work. Some of the classes in this set employ termination guaranteeing checks called ‘acyclicity tests’ that analyze the information flow between rules to check whether cyclic dependencies exists that can lead to infinite chase. Weak acyclicity [23,24], was one of the first such notions, and was extended to joint acyclicity [36] and super weak acyclicity [35]. The main approach used in these techniques is to exploit the structure of the rules and use a dependency graph that models the propagation path of constants across various predicates in the rules, and restricting the dependency graph to be acyclic. The main drawback of these approaches is that they only analyze the schema/Tbox part of the rule sets, and ignore the instance part, and hence, produces a large number of false alarms, i.e. it is often the case that although dependency graph is cyclic, the chase is finite. Recently, a more dynamic approach, called the MFA technique, that also takes into account the instance part of the rule sets was proposed in Cuenca Grau et al. [31], where existence of cyclic Skolem blank-node/constant generations in the chase is detected by augmenting the rules with extra information that keeps track of the Skolem function used to generate each Skolem blank-node. As shown in section 6, our technique of safety subsumes the MFA technique, and supports for much more expressive rule sets, by also keeping track of the vectors used by rule bodies while Skolem blank-nodes are generated.

The approach based on dependency graph, for instance, is used by Halevi et al. in the context of peer-peer data management systems [18], and decidability is attained by not allowing any kind cycles in the peer topology. Whereas in the context of Data exchange, WA is used in [23,24] to assure decidability, and the recent work by Marnette [35] employs the super weak acyclicity (SWA) to ensure decidability. It was shown in Cuenca Grau et al [31] that their MFA technique strictly subsumes both WA and SWA techniques in expressivity. Since, we saw in section 6 that our technique of safety subsumes the MFA technique and allows to represent much more expressive rule sets, safety technique can straightforwardly be employed in the above mentioned systems with decidability guarantees for query answering.

8. Summary and Conclusion

In this paper, we study the problem of query answering over contextualized RDF knowledge in the presence of forall-existential bridge rules. We show that the problem, in general, is undecidable, and present a few decidable classes of quad-systems. Table 1 displays the complexity results of chase computation and query entailment for the various classes of quad-systems, we have derived. Classes csafe, msafe, and safe, ensure decidability by restricting the structure of Skolem blank-nodes generated in the dChase. Briefly, the above classes do not allow an infinite descendant chain for Skolem blank-nodes generated, by constraining each Skolem blank-node in a descendant chain to have a different value for certain attributes, whose value sets are finite. RR and restricted RR quad-systems, do not allow the generation of Skolem blank nodes, thus constraining the dChase to have only constants from the initial quad-system. The above classes which suit varying situations, can be used to extend the currently established tools for contextual reasoning to give support for expressive bridge rules with conjunctions and existential quantifiers with decidability guarantees. From an expressivity point of view, the class of safe quad-systems subsumes all the above classes, and other well known classes in the realm of $\forall \exists$ rules with finite chases. We view the results obtained in this paper as a general foundation for contextual reasoning and query answering over contextualized RDF knowledge formats such as quads, and can straightforwardly be used to extend existing quad stores.
<table>
<thead>
<tr>
<th>Quad-System Fragment</th>
<th>Chase size w.r.t. input quad-system</th>
<th>Data Complexity of CCQ entailment</th>
<th>Combined Complexity of CCQ entailment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unrestricted Quad-Systems</td>
<td>Infinite</td>
<td>Undecidable</td>
<td>Undecidable</td>
</tr>
<tr>
<td>Safe Quad-Systems</td>
<td>Double exponential</td>
<td>PTIME-complete</td>
<td>2EXPTIME-complete</td>
</tr>
<tr>
<td>MSafe Quad-Systems</td>
<td>Double exponential</td>
<td>PTIME-complete</td>
<td>2EXPTIME-complete</td>
</tr>
<tr>
<td>CSafe Quad-Systems</td>
<td>Double exponential</td>
<td>PTIME-complete</td>
<td>2EXPTIME-complete</td>
</tr>
<tr>
<td>RR Quad-Systems</td>
<td>Polynomial</td>
<td>PTIME-complete</td>
<td>EXPTIME-complete</td>
</tr>
<tr>
<td>Restricted RR Quad-Systems</td>
<td>Polynomial</td>
<td>PTIME-complete</td>
<td>NP-complete</td>
</tr>
</tbody>
</table>

Table 1
Complexity info for various quad-system fragments

References

Suppose (a) is the case, then
\[ Q \not\preceq Q \]
This means that neither of the conditions (i) and (ii) of
\[ Q \preceq Q \]
is irreflexive, transitive, and linear.

Note that a strict linear order is a relation
Property 3.1.

Appendix

A. Proofs of Section 3

Property 3.1. Note that a strict linear order is a relation that is irreflexive, transitive, and linear.

Irreflexivity: By contradiction, suppose \( \preceq_q \) is not irreflexive, then there exists \( Q \in Q \) s.t. \( Q \not\preceq_q Q \) holds. This means that neither of the conditions (i) and (ii) of \( \preceq_q \) definition holds for \( Q \). Hence, due to condition (iii)
\[ Q \not\preceq_q Q \]
which is a contradiction.

Linearity: Note that for any two distinct \( Q, Q' \in Q \), one of the following holds: (a) \( Q \subset Q' \), (b) \( Q' \subset Q \), or (c) \( Q \cap Q' \) and \( Q' \cap Q \) are non-empty and disjoint. Suppose (a) is the case, then \( Q \not\preceq_q Q' \) holds. Similarly, if (b) is the case then \( Q' \not\preceq_q Q \) holds. Otherwise if (c) is the case, then by condition (ii), either \( Q \not\preceq_q Q' \) or \( Q' \not\preceq_q Q \) should hold. Hence, \( \preceq_q \) is a linear order over \( Q \).

Transitivity: Suppose there exists \( Q, Q', Q'' \in Q \)
s.t. \( Q \not\preceq_q Q' \) and \( Q' \not\preceq_q Q'' \). Then, one of the following four cases hold: (a) \( Q \not\preceq_q Q' \) due to (i) and \( Q' \not\preceq_q Q'' \) due to (i), (b) \( Q \not\preceq_q Q' \) due to (i) and \( Q' \not\preceq_q Q'' \) due to (ii), (c) \( Q \not\preceq_q Q' \) due to (ii) and \( Q' \not\preceq_q Q'' \) due to (i), (d) \( Q \not\preceq_q Q' \) due to (ii) and \( Q' \not\preceq_q Q'' \) due to (ii).

Suppose if (a) is the case, then trivially \( Q \subset Q'' \), and hence by applying condition (i) \( Q \not\preceq_q Q'' \). Otherwise if (b) is the case, then either (1) \( Q \subset Q'' \) or (2) \( Q \subset Q'' \). Suppose, (1) is the case then, by (i) \( Q \not\preceq_q Q'' \). Otherwise, if (2) is the case, then since, \( Q \subset Q'' \), it cannot be the case that greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\) \( \not\preceq_q \) greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\), and it cannot be the case that greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\) \( \not\preceq_q \) greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\). Hence, it should be the case that greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\) \( \preceq_q \) greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\) and greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\) \( \preceq_q \) greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\). But since, greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\) \( \preceq_q \) greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\), it follows that greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\) \( \preceq_q \) greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\) and hence by condition (ii), \( Q \not\preceq_q Q'' \). Hence, if (b) is the case, then in both possible case (1) or (2), it should be the case that \( Q \not\preceq_q Q'' \). Otherwise if (c) is the case, then similar to the arguments in (b), by condition (i) or (ii), it can easily be seen that \( Q \not\preceq_q Q'' \).

Otherwise, if (d) is the case, then it should be the case that greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\) \( \preceq_q \) greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\) \( \preceq_q \), then one of the following holds: (1) \( Q'' \not\preceq_q Q \) by condition (i) or (2) \( Q'' \not\preceq_q Q \) by condition (ii). Suppose, (1) is the case, then it should be the case that \( Q'' \not\preceq_q Q \). Hence, it should not be the case that greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\) \( \preceq_q \) greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\), and it should not be the case that greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\) \( \preceq_q \) greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\). Hence, it should be the case that greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\) \( \preceq_q \) greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\) and it should not be the case that greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\) \( \preceq_q \) greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\). Applying, (1) in \( \preceq_q \), we get greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\) \( \preceq_q \) greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\), and it should be the case that greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\) \( \preceq_q \) greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\) (\( \bowtie \)). Applying, (i) in \( \preceq_q \), we get greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\) \( \preceq_q \) greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\), and it should be the case that greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\) \( \preceq_q \) greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\), which is a contradiction. Suppose if (2) is the case, then it should be the case that greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\) \( \preceq_q \) greatestQuad\(_{\preceq_q}(Q'' \setminus Q)\). The above can be written as: greatestQuad\(_{\preceq_q}(Q'' \setminus (Q \cap Q'))\) \( \preceq_q \) greatestQuad\(_{\preceq_q}(Q'' \setminus (Q \cap Q'))\). Using \( Q \cap Q' \cap Q'' \subseteq Q \cap Q' \), it follows that
greatestQuad_{<i}(Q'' \setminus (Q \cap Q' \cap Q'')) \preceq_i \text{greatestQuad}_{<i}(Q \setminus (Q \cap Q' \cap Q'')) \cup \text{greatestQuad}_{<i}(Q' \setminus (Q \cap Q' \cap Q'')) \cup \text{greatestQuad}_{<i}(Q'' \setminus (Q \cap Q' \cap Q''))

\text{Proof}

We apply similar transformations in (1) and (2). Also applying similar transformations in (3) and (4), we get

\text{greatestQuad}_{<i}(Q \setminus (Q \cap Q' \cap Q'')) \preceq_i \text{greatestQuad}_{<i}(Q' \setminus (Q \cap Q' \cap Q'')) \preceq_i \text{greatestQuad}_{<i}(Q'' \setminus (Q \cap Q' \cap Q''))

By the hypothesis, we proceed by induction on the form of the greatestQuad_{<i}. In the former case, we say that $Q \prec_i Q''$. Hence, it should be the case that $Q \prec_i Q''$.

\begin{flushright}
\text{\ding{51}}
\end{flushright}

\textbf{Theorem 3.5.} We show that CCQ entailment is undecidable for unrestricted quad-systems, by showing that the well-known undecidable problem of “non-emptiness of intersection of context-free grammars” is reducible to the CCQ answering problem.

Given an alphabet $\Sigma$, string $\vec{w}$ is a sequence of symbols from $\Sigma$. A language $L$ is a subset of $\Sigma^*$, where $\Sigma^*$ is the set of all strings that can be constructed from the alphabet $\Sigma$, and also includes the empty string $\epsilon$. Grammars are machineries that generate a particular language. A grammar $G$ is a quadruple $\langle V, T, S, P \rangle$, where $V$ is the set of variables, $T$, the set of terminals, $S \in V$ is the start symbol, and $P$ is a set of production rules (PR), in which each PR $r \in P$, is of the form:

$$\vec{w} \rightarrow \vec{w}'$$

where $\vec{w}, \vec{w}' \in \{T \cup V\}^*$. Intuitively application of a PR $r$ of the form above on a string $\vec{w}_1$, replaces every occurrence of the sequence $\vec{w}$ in $\vec{w}_1$ with $\vec{w}'$. PRs are applied starting from the start symbol $S$ until it results in a string $\vec{w}$, with $\vec{w} \in \Sigma^*$ or no more production rules can be applied on $\vec{w}$. In the former case, we say that $\vec{w} \in L(G)$, the language generated by grammar $G$. For a detailed review of grammars, we refer the reader to Harrison et al. [32]. A context-free grammar (CFG) is a grammar, whose set of PRs $P$, have the following property:

\textbf{Property A.1.} For a CFG, every PR is of the form $v \rightarrow \vec{w}$, where $v \in V$, $\vec{w} \in \{T \cup V\}^*$.

Given two CFGs, $G_1 = \langle V_1, T, S_1, P_1 \rangle$ and $G_2 = \langle V_2, T, S_2, P_2 \rangle$, where $V_1, V_2$ are the set of variables, $T$ such that $T \cap (V_1 \cup V_2) = \emptyset$ is the set of terminals, $S_1 \in V_1$ is the start symbol of $G_1$, and $P_1$ are the set of PRs of the form $v \rightarrow \vec{w}$, where $v \in V$, $\vec{w}$ is a sequence of the form $w_1...w_n$, where $w_5 \in V_1 \cup T$. Similarly $s_2, P_2$ is defined. Deciding whether the language generated by the grammars $L(G_1)$ and $L(G_2)$ have non-empty intersection is known to be undecidable [32].

Given two CFGs, $G_1 = \langle V_1, T, S_1, P_1 \rangle$ and $G_2 = \langle V_2, T, S_2, P_2 \rangle$, we encode grammars $G_1, G_2$ into a quad-system of the form $QS_c = (Q_c, R_c)$, with a single context identifier $c$. Each PR $r = v \rightarrow \vec{w} \in P_1 \cup P_2$, with $\vec{w} = w_1w_2w_3..w_n$, is encoded as a BR of the form:

$$c: (x_1, w_1, x_2), c: (x_2, w_2, x_3), ..., c: (x_n, w_n, x_{n+1}) \rightarrow c: (x_1, v, x_{n+1})$$

(9)

where $x_1, ..., x_{n+1}$ are variables. W.l.o.g. we assume that the set of terminal symbols $T$ is equal to the set of terminal symbols occurring in $P_1 \cup P_2$. For each terminal symbol $t_i \in T, R$ contains a BR of the form:

$$c: (x, rdf:type, C) \rightarrow \exists y c: (x, t_i, y),$$

$$c: (y, rdf:type, C)$$

(10)

and $Q_c$ contains only the triple:

$$c: (a, rdf:type, C)$$

We in the following show that:

$$QS_c \models \exists y c: (a, S_1, y) \land c: (a, S_2, y) \leftrightarrow L(G_1) \cap L(G_2) \neq \emptyset$$

(11)

\textbf{Claim (1)} For any $\vec{w} = t_1, .. , t_p \in T^*$, there exists $b_1, .. , b_p$, such that:

$$c: (a, t_1, b_1), c: (b_1, t_2, b_2), ..., c: (b_{p-1}, t_p, b_p), c: (b_p, rdf:type, C) \in dChase(QS_c).$$

We proceed by induction on the $|\vec{w}|$.

\textbf{base case} suppose if $|\vec{w}| = 1$, then $\vec{w} = t_i$, for some $t_i \in T$. But Since by construction $c: (a, rdf:type, C) \in dChase_a(QS_c)$, on which rules of the form (10) is applicable, hence, there exists an $i$ such that $dChase_a(QS_c)$ contains $c: (a, t_i, b_i), c: (b_i, rdf:type, C)$, for each $t_i \in T$. Hence, the base case.

\textbf{hypothesis} for any $\vec{w} = t_1, .. , t_p$, if $|\vec{w}| \leq p'$, then there exists $b_1, .. , b_p$, such that $c: (a, t_1, b_1), c: (b_1, t_2, b_2), ..., c: (b_{p-1}, t_p, b_p), c: (b_p, rdf:type, C) \in dChase(QS_c)$.

\textbf{inductive step} suppose $\vec{w} = t_1, .. , t_{p+1}$, with $|\vec{w}| \leq p' + 1$. Since $\vec{w}$ can be written as $\vec{w}'t_{p+1}$, where $\vec{w}' = t_1, .. , t_p$, and by hypothesis, there exists $b_1, .. , b_p$ such that $c: (a, t_1, b_1), c: (b_1, t_2, b_2), ..., c: (b_{p-1}, t_p, b_p), c: (b_p, rdf:type, C) \in dChase(QS_c),$
dChase(QS_c). Also since rules of the form (10) are applicable on c: (b_p, rdf:type, C), and hence produces triples of the form c: (b_{p+1}, t_i, b_{p+1}), c: (b_{p+1}, rdf:type, C), for each t_i \in T. Since t_{p+1} \in T, the claim follows.

For a grammar G = (V, T, S, P), whose start symbol is S, and for any \bar{w} \in \{V \cup T\}^*, for some V_j \in V, we denote by V_j \rightarrow \bar{w}, the fact that \bar{w} was derived from V_j by i production steps, i.e. there exists steps V_j \rightarrow r_1, ..., r_i \rightarrow \bar{w}, which lead to the production of \bar{w}. For any \bar{w}, \bar{w} \in L(G), iff there exists an i such that S \rightarrow^i \bar{w}. For any V_j \in V, we use V_j \rightarrow^* \bar{w} to denote the fact that there exists an arbitrary i, such that V_j \rightarrow^i \bar{w}.

Claim (2) For any \bar{w} = t_1...t_p \in \{V \cup T\}^*, and for any V_j \in V, if V_j \rightarrow^* \bar{w} and there exists b_1, ..., b_{p+1}, with c: (b_1, t_1, b_2), ..., c: (b_p, t_p, b_{p+1}) \in dChase(QS_c), then c: (b_1, V_j, b_{p+1}) \in dChase(QS_c).

We prove this by induction on the size of \bar{w}.

Base case Suppose |\bar{w}| = 1, then \bar{w} = t_k, for some t_k \in T. If there exists b_1, b_k such that c: (b_1, t_k, b_2). But since there exists a PR V_j \rightarrow t_k, by transformation given in (9), there exists a BR c: (x_1, t_k, x_2) \rightarrow c: (x_1, V_j, x_2) \in R, which is applicable on c: (b_1, t_k, b_2) and hence the quad c: (b_1, V_j, b_2) \in dChase(QS_c).

Hypothesis For any \bar{w} = t_1...t_p, with |\bar{w}| \leq p', and for any V_j \in V, if V_j \rightarrow^* \bar{w} and there exists b_1, ..., b_p, b_{p+1}, such that c: (b_1, t_1, b_2), ..., c: (b_p, t_p, b_{p+1}) \in dChase(QS_c), then c: (b_1, V_j, b_{p+1}) \in dChase(QS_c).

Inductive step Suppose if \bar{w} = t_1...t_{p+1}, with |\bar{w}| \leq p'+1, and V_j \rightarrow^* \bar{w}, and there exists b_1, ..., b_{p+1}, b_{p+2}, such that c: (b_1, t_1, b_2), ..., c: (b_{p+1}, t_{p+1}, b_{p+2}) \in dChase(QS_c). Also, one of the following holds (i) i = 1, or (ii) i > 1. Suppose (i) is the case, then it is trivially the case that c: (b_1, V_j, b_{p+2}) \in dChase(QS_c). Suppose if (ii) is the case, one of the two sub cases holds (a) V_j \rightarrow^{i-1} V_k, for some V_k \in V and V_k \rightarrow^* \bar{w} or (b) there exist a V_k \in V, such that V_k \rightarrow^* t_{q+1}...t_{q+l}, with 2 \leq l \leq p, where V_j \rightarrow^* t_1...t_q V_k t_{l-1}...t_{p+1}. If (a) is the case, trivially then c: (b_1, V_k, b_{q+2}) \in dChase(QS_c), and since by construction there exists c: (x_0, V_k, x_1) \rightarrow c: (x_0, V_{k+1}, x_2), ..., c: (x_0, V_{k+l}, x_1) \rightarrow c: (x_0, V_j, x_1) \in R, c: (b_1, V_j, b_{q+2}) \in dChase(QS_c). If (b) is the case, then since |t_{q+1}...t_{q+l}| \geq 2, |t_1...t_q V_k t_{l-1}...t_{p+1}| \leq p'. This implies that c: (b_1, V_j, b_{p+2}) \in dChase(QS_c).

Similarly, by construction of dChase(QS_c), the following claim can straightforwardly be shown to hold:

**Claim (3)** For any \bar{w} = t_1...t_p \in \{V \cup T\}^*, and for any V_j \in V, if there exists b_1, ..., b_p, b_{p+1}, with c: (b_1, t_1, b_2), ..., c: (b_p, t_p, b_{p+1}) \in dChase(QS_c) and c: (b_1, V_j, b_{p+1}) \in dChase(QS_c), then V_j \rightarrow^* \bar{w}.

(a) For any \bar{w} = t_1...t_p \in T^*, if \bar{w} \in L(G_1) \cap L(G_2), then by claim 1, since there exists b_1, ..., b_p, such that c: (a_1, t_1, b_1), ..., c: (b_{p-1}, t_p, b_p) \in dChase(QS_c). But since \bar{w} \in L(G_1) and \bar{w} \in L(G_2), S_1 \rightarrow \bar{w} and S_2 \rightarrow \bar{w}. Hence by claim 2, c: (a_1, S_1, b_p), c: (a_2, S_2, b_p) \in dChase(QS_c), which implies that dChase(QS_c) |\exists y c: (a_1, s_1, y) \land c: (a_2, s_2, y). Hence, QS_c |= \exists y c: (a_1, s_1, y) \land c: (a_2, s_2, y).

(b) Suppose if QS_c |= \exists y c: (a_1, s_1, y) \land c: (a_2, s_2, y), then this implies that there exists b_p such that c: (a_1, S_1, b_p), c: (a_2, S_2, b_p) \in dChase(QS_c). Then it is the case that there exists \bar{w} = t_1...t_p \in T^*, and b_1, ..., b_p, such that c: (a_1, t_1, b_1), ..., c: (b_{p-1}, t_p, b_p), c: (a_1, S_1, b_p), c: (a_2, S_2, b_p) \in dChase(QS_c). Then by claim 3, S_1 \rightarrow^* \bar{w}, S_2 \rightarrow^* \bar{w}. Hence, \bar{w} \in L(G_1) \cap L(G_2).

By (a),(b) it follows that there exists \bar{w} \in L(G_1) \cap L(G_2) iff QS_c |= \exists y c: (a_1, s_1, y) \land c: (a_2, s_2, y). As we have shown that the intersection of CFGs, which is an undecidable problem, is reducible to the problem of query entailment on unrestricted quad-system, the latter is undecidable.

\(\square\)

**B. Proofs of Section 4**

**Theorem 4.13.** We in the following show the case of dChase\(\text{safe}(QS_c)\), i.e. unCSafe \(\in dChase\(\text{safe}(QS_c)\) iff QS_c is unsafe. The proof follows from lemma B.1 and lemma B.2 below.

The proofs for the case of dChase\(\text{safe}(QS_c)\) and dChase\(\text{safe}(QS_c)\) is similar, and is omitted.

\(\square\)

**Lemma B.1** (Soundness). For any quad-system QS_c = \(\{Q_c, R\}\), if the quad unCSafe \(\in dChase\(\text{safe}(QS_c)\) then QS_c is unsafe.

**Proof.** Note that augC(R) = \(\bigcup_{r \in R} \text{augC}(r) \cup \{br\text{TR}\}\), where brTR is the range restricted BR c: (x_1, descendantOf, z), c: (z, descendantOf, x_2) \rightarrow c: (x_1, descendantOf, x_2). Also for each r \in R,
body(r) = body(augC(r)), and for any c ∈ C, c: (s, p, o) ∈ head(r) iff c: (s, p, o) ∈ head(augC(r)). That is, head(r) = head(augC(r))(C), where head(r)(C) denotes the quad-patterns in head(r), whose context identifiers is in C. Also, head(augC(r)) = head(augC(r))(C) ∪ head(augC(r))(c), and also the set of existentially quantified variables in head(augC(r))(c) is contained in the set of existentially quantified variables in head(augC(r))(C). We first prove the following claim:

Claim (0) For any quad-system QSC = (Qc, R), let i be a c-safe dChase iteration, and j be the number of c-safe dChase iterations before i in which brTR was applied, then dChasei−j(QSC) = dChasei(QSC)(C).

We approach the proof of the above claim by induction on i.

base case If i = 1, then dChase1safe(QSC)(c) = ∅ and dChase1safe(QSC)(C) = dChase0safe(QSC) = dChasei(QSC). Hence, it should be the case that applicableaugC(R)(brTR, µ, dChase0safe(QSC)) does not hold, for any µ. Hence, applicableR(r, µ, dChase0(QSC)) if applicableaugC(R)(augC(r), µ, dChase0safe(QSC)), for any r ∈ R, assignment µ. Also using (†), it follows that dChase1(QSC) = dChasei(QSC)(C).

hypothesis for any i ≤ k, if i is a c-safe dChase iteration, and j be the number of c-safe dChase iterations before i in which brTR was applied, then dChasei−j(QSC) = dChasei(QSC)(C).

inductive suppose i = k+1, then one of the following three cases should hold: (a) applicableaugC(R)(r, µ, dChasenk(QSC)) does not hold for any r ∈ augC(R), assignment µ, and dChasenk+1(QSC) = dChasenk(QSC), or (b) applicableaugC(R)(brTR, µ, dChasenk(QSC)) holds, for some assignment µ, or (c) applicableaugC(R)(r, µ, dChasenk(QSC)) holds, for some r ∈ augC(R) \ {brTR}, for some assignment µ. If (a) is the case, then it should be the case that applicableR(r′, µ, dChasenk−j(QSC)) does not hold, for any r′ ∈ R, assignment µ. As a result dChasenk+1−j(QSC) = dChasenk−j(QSC), and hence, dChasenk+1−j(QSC) = dChasenk−j+1(QSC)(C). If (b) is the case, then since dChasenk+1−j(QSC)(C) = dChasenk−j(QSC)(C) ∈ dChasek−j+1−1(QSC) = dChasek−j(QSC). If (c) is the case, then it should be the case that applicableR(r′, µ, dChasenk−j(QSC), where r = augC(r′) and head(r)(C) = head(r). Hence, it should be the case that dChasek+1safe(QSC)(C) = dChasek+1−j(QSC).

The following claim, which straightforwardly follows from claim 0, shows that any quad c: (s, p, o), with c ∈ C derived in c-safe dChase, is also derived in its standard dChase. In this way, c-safe dChase do not generate any unsound triples in any context c ∈ C.

Claim (1) For any quad c: (s, p, o), where c ∈ C, if c: (s, p, o) ∈ dChasei(QSC), then c: (s, p, o) ∈ dChasei(QSC).

The following claim shows that the set of origin context quads are also sound.

Claim (2) If there exists quad c: (b, originContext, c) ∈ dChasei(QSC), then c ∈ originContexts(b).

If c: (b, originContext, c) ∈ dChasei(QSC), there exists i ∈ N, s.t. c: (b, originContext, c) ∈ dChasei(QSC) and there exists no j < i with c: (b, originContext, c) ∈ dChasej(QSC). But if c: (b, originContext, c) ∈ dChasei(QSC) implies that there exists an augC(r) = body(x, z) → head(x, y), with c: (yj, originContext, c) ∈ head(x, y), yj ∈ {y}, s.t. c: (b, originContext, c) was generated due to application of an assignment µ on augC(r), with b = yj[µext(y)]. This implies that there exists c: (s, p, o) ∈ head(x, y), with s = yj or p = yj or o = yj, c ∈ C. Since according to our assumption, i is the first iteration in which c: (b, originContext, c) is generated, it follows that i is the first iteration in which c: (s, p, o)[µext(y)] is also generated. Let k be the number of iterations before i in which brTR was applied. By applying claim 0, it should be the case that c: (s, p, o) ∈ dChasei−k(QSC), and i − k should be the first such dChase iteration. Hence, c ∈ originContexts(b).

In the following claim, we prove the soundness of the descendant quads generated in a safe dChase.

Claim (3) For any two distinct blank nodes b, b′ in dChasei(QSC), if c: (b′, descendantOf, b) ∈ dChasei(QSC) then b′ is a descendant of b.

Since any quad of the form c: (b′, descendantOf, b) ∈ dChasei(QSC) is not an element of Qc, and can only be introduced by an application of a BR r ∈ augC(R), any quad of the form c: (b′, descendantOf, b) can only be introduced, earliest in the first iteration of dChasei(QSC). Suppose c: (b′, descendantOf, b) ∈ dChasei(QSC), then there exists an iteration i ≥ 1 s.t. c: (b′, descendantOf, b)
in $dChase_i^\text{cSafe}(Q_SC)$, for any $j \geq i$, and $c_j : (b', \text{descendantOf}, b) \in dChase_i^\text{cSafe}(Q_SC)$, for any $j' < i$. We apply induction on $i$ for the proof.

**base case** suppose $c_i : (b', \text{descendantOf}, b) \in dChase_i^\text{cSafe}(Q_SC)$ and since $b \neq b'$, then there exists a BR $r \in \text{augC}(R)$, $\exists \mu$ s.t. $\text{applicable}_{\text{augC}(R)}(r, \mu, dChase_{i-1}^\text{cSafe}(Q_SC))$, i.e. $\text{body}(r)(\vec{x}, \vec{z})[\mu] \subseteq dChase_0^\text{cSafe}(Q_SC)$ and $c_i : (b', \text{descendantOf}, b) \in \text{head}(r)(\vec{x}, \vec{y})[\mu^\text{ext}(\vec{y})]$. Then by construction of $\text{augC}(r)$, it follows that $b = y_j[\mu^\text{ext}(\vec{y})]$, for some $y_j \in \{\vec{y}\}$ and $b' = \mu(x_i)$, for some $x_i \in \{\vec{x}\}$. Since $dChase_0(Q_SC) = dChase_0^\text{cSafe}(Q_SC)$, it follows using (i) that $\text{applicable}_{\text{augC}(r)}(r', \mu, dChase_{0}^\text{cSafe}(Q_SC))$ holds, for $r' = \text{body}(r')(\vec{x}, \vec{z}) \rightarrow \text{head}(r')(\vec{x}, \vec{y})$, with $\text{augC}(r') = r$. Hence, by construction, it follows that $b = y_j[\mu^\text{ext}(\vec{y})] \in C(dChase_{1}^\text{cSafe}(Q_SC))$, for $y_j \in \{\vec{y}\}$ and $b' = \mu(x_i)$, for $x_i \in \{\vec{x}\}$. Hence $b'$ is a descendant of $b$ (by definition).

**hypothesis** if $c_i : (b', \text{descendantOf}, b) \in dChase_i^\text{cSafe}(Q_SC)$, for $1 \leq i \leq k$, then $b'$ is a descendant of $b$.

**inductive step** suppose $c_i : (b', \text{descendantOf}, b) \in dChase_{k+1}^\text{cSafe}(Q_SC)$, then either (i) $c_i : (b', \text{descendantOf}, b) \in dChase_{k+1}^\text{cSafe}(Q_SC)$ or (ii) $c_i : (b', \text{descendantOf}, b) \notin dChase_{k+1}^\text{cSafe}(Q_SC)$. Suppose (i) is the case, then by hypothesis, $b'$ is a descendant of $b$. If (ii) is the case, then either (a) $c_i : (b', \text{descendantOf}, b)$ is the result of the application of a $brTR \in \text{augC}(R)$ on $dChase_k^\text{cSafe}(Q_SC)$ or (b) $c_i : (b', \text{descendantOf}, b)$ is the result of the application of a $r \in \text{augC}(R) \setminus \{brTR\}$ on $dChase_k^\text{cSafe}(Q_SC)$. If (a) is the case, then there exists a $b'' \in C(dChase_k^\text{cSafe}(Q_SC))$ s.t. $c_i : (b', \text{descendantOf}, b'') \in dChase_k^\text{cSafe}(Q_SC)$ and $c_i : (b'', \text{descendantOf}, b) \in dChase_k^\text{cSafe}(Q_SC)$. Hence, by hypothesis $b'$ is a descendant of $b''$ and $b''$ is a descendant of $b$. Since ‘descendantOf’ relation is transitive, $b'$ is a descendant of $b$. Otherwise if (b) is the case then similar to the arguments used in the base case, it can easily be seen that $b'$ is a descendant of $b$.

Suppose if the quad $\text{unCSafe} \in dChase_i^\text{cSafe}(Q_SC)$, then this implies that there exists an iteration $i$ s.t. the function $\text{unCSafeTest}$ on $\text{augC}(r)$, with $r = \text{body}(r)(\vec{x}, \vec{z}) \rightarrow \text{head}(r)(\vec{x}, \vec{y}) \in R$, assignment $\mu$, and $dChase_{i-1}^\text{cSafe}(Q_SC)$ returns True. This implies that, there exists $b, b' \in B$, $y_j \in \{\vec{y}\}$ s.t. $\text{body}(r)(\vec{x}, \vec{z})[\mu] \subseteq dChase_i^\text{cSafe}(Q_SC)$, $b \in \{\mu(\vec{x})\}$, $c_i : (b', \text{descendantOf}, b) \in dChase_i^\text{cSafe}(Q_SC)$ and $\{c_i, c : (b', \text{originContext}, c) \in dChase_i^\text{cSafe}(Q_SC)\} = \text{cScope}(y_j, head(r)(\vec{x}, \vec{y}))$. Suppose $k$ be the number of $dChase$ iterations before $i$, in which $brTR$ was applied. Hence, by claim 0, $dChase_{i-k-1}(Q_SC) = dChase_{i-k-1}^\text{cSafe}(Q_SC)\{C\}$, and consequently $\text{applicable}_{\text{R}(r, \mu, dChase_{i-k-1}(Q_SC))}$ holds. Hence, as a result of $\mu$ being applied on $r$, there exists $b' = y_j[\mu^\text{ext}(\vec{y})] \in B(dChase_{i-k}(Q_SC))$, with $b' \in \{\mu(\vec{x})\}$. Hence, by definition $\text{originContext}(b'') = \text{cScope}(y_j, head(r))$, and $b$ is a descendantOf $b''$. If $b' \neq b'$, then by Claim 2, $b'$ is a descendantOf $b$, otherwise $b' = b$ and hence $b'$ is a descendantOf $b''$. Consequently, $b'$ is a descendantOf $b''$. Also, applying claim 3, we get that $\text{originContext}(b'') = \text{originContext}(b')$, which means that prerequisites of unsafeness is satisfied, and hence, $Q_SC$ is unsafe.

**Lemma B.2 (Completeness).** For any quad-system, $Q_SC = (Q_C, R)$, if $Q_SC$ is unsafe then $\text{unCSafe} \in dChase_i^\text{cSafe}(Q_SC)$.

**Proof.** We first prove a few supporting claims in order to prove the theorem.

**Claim (0)** For any quad-system $Q_SC = (Q_C, R)$, suppose $\text{unCSafe} \notin dChase_i^\text{cSafe}(Q_SC)$, then for any $dChase$ iteration $i$, there exists a $j \geq 0$ s.t. $dChase_i(Q_SC) = dChase_{i+j}^\text{cSafe}(Q_SC)\{C\}$.

We approach the proof by induction on $i$.

**base case** for $i = 0$, we know that $dChase_0(Q_SC) = dChase_0^\text{cSafe}(Q_SC) = Q_C$. Hence, the base case trivially holds.

**hypothesis** for $i \leq k \in \mathbb{N}$, there exists $j \geq 0$ s.t. $dChase_i(Q_SC) = dChase_{i+j}^\text{cSafe}(Q_SC)\{C\}$.

**step case** for $i = k + 1$, one of the following holds:

(a) $dChase_{k+1}(Q_SC) = dChase_k(Q_SC)$ or (b) $dChase_{k+1}(Q_SC) = dChase_k(Q_SC) \cup \text{head}(r)(\vec{x}, \vec{y})[\mu^\text{ext}(\vec{y})]$ and $\text{applicable}_{\text{R}(r, \mu, dChase_k(Q_SC))}$ holds, for some $r = \text{body}(r)(\vec{x}, \vec{z}) \rightarrow \text{head}(r)(\vec{x}, \vec{y})$, assignment $\mu$. If (a) is the case, then trivially the claim holds. Otherwise, if (b) is the case, then let $j \in \mathbb{N}$ be s.t. $dChase_k(Q.SC) = dChase_{k+j}^\text{cSafe}(Q_SC)\{C\}$. Let $j' \geq j, l \in \mathbb{N}$ be s.t. $\text{applicable}_{\text{augC}(R)\{brTR\}}(brTR, \mu, dChase_k^\text{cSafe}(Q_SC))$, for any $j' \geq j$, and $\text{applicable}_{\text{augC}(R)\{brTR\}}(brTR, \mu, dChase_k^\text{cSafe}(Q_SC))$ does not hold. By construction, it should be the case that $\text{applicable}_{\text{R}'(r', \mu, dChase_k^\text{cSafe}(Q_SC))}$ holds, where $r' = \text{augC}(r)$. Also since no new Skolem blank node was introduced in any safe $dChase$ iteration $k + l$, for any $j \leq l \leq j'$, it should be the case that $\text{head}(r')[\mu^\text{ext}(\vec{y})] = \text{head}(r')[\mu^\text{ext}(\vec{y})]$.

Since, $dChase_{k+j}^\text{cSafe}(Q_SC)\{C\} = dChase_k(Q_SC)$, for
any \( j \leq l \leq j' \), and \( d\text{Chase}^{csafe}_{k+j+1}(Q_S) = d\text{Chase}^{csafe}_{k+j'}(Q_S) \cup \text{head}(r')[\mu^{ext}(\vec{b})], d\text{Chase}^{csafe}_{k+j+1}(Q_S)(\mathcal{C}) = d\text{Chase}_{k+1}(Q_S)^{csafe}(Q_S) \). Hence, the claim follows.

The following claim, which straightforwardly follows from claim 0, shows that, for csafe quad-systems its standard dChase is contained in its safe dChase.

\textbf{Claim (1)} Suppose \( \text{unCSafe} \not\in d\text{Chase}^{csafe}(Q_S) \), then \( d\text{Chase}(Q_S) \subseteq d\text{Chase}^{csafe}(Q_S) \).

Claim below shows that the generation of originContext-quad in csafe dChase is complete.

\textbf{Claim (2)} For any quad-system \( Q_S \), if \( \text{unCSafe} \not\in d\text{Chase}^{csafe}(Q_S) \), then for any skolem blank-node \( b \) generated in \( d\text{Chase}(Q_S) \), and for any \( c \in C \), if \( c \in \text{originContext}(b) \), then there exists a quad \( c_c : (b, \text{originContext}, c) \in d\text{Chase}^{csafe}(Q_S) \).

Since the only way a skolem blank node \( b \) gets generated in any iteration \( i \) of \( d\text{Chase}(Q_S) \) is by the application of a BR \( r \) in \( R \), i.e. when there \( \exists r = \text{body}(r)(\vec{x}, \vec{z}) \to \text{head}(r)(\vec{x}, \vec{y}) \in R \), assignment \( \mu \), s.t. \( \text{applicable}_R(r, \mu, d\text{Chase}_{i-1}(Q_S)) \), and \( b = y_j[\mu^{ext}(\vec{b})] \), for some \( y_j \in \{\vec{y}\} \), and \( d\text{Chase}_i(Q_S) = d\text{Chase}_{i-1}(Q_S) \cup \text{head}(r)(\vec{x}, \vec{y})[\mu^{ext}(\vec{b})] \). Also since \( c \in \text{originContext}(b) \), it should be the case that \( c \in c\text{Scope}(y_j, \text{head}(r)) \). From claim 0, we know that there exists \( j \geq 0 \), s.t. \( d\text{Chase}_i(Q_S) = d\text{Chase}^{csafe}_{i+j}(Q_S)(\mathcal{C}) \). W.l.o.g. assume that \( i + j \) is the first such csafe dChase iteration. Hence, it follows that \( \text{applicable}_{\text{augC}(R)}(r', \mu, d\text{Chase}^{csafe}_{i+j-1}(Q_S)) \), where \( r' = \text{augC}(r) \). Since, \( \text{head}(r') \subseteq \text{head}(r') \), it should be the case that \( c \in c\text{Scope}(y_j, \text{head}(r')) \). Hence, by construction of \( \text{augC} \), \( c_c : (y_j, \text{originContext}, c) \in \text{head}(r') \), and as a result of application of \( \mu \) on \( r' \) in iteration \( i + j \), \( c_c : (b, \text{originContext}, c) \) gets generated in \( d\text{Chase}^{csafe}_{i+j}(Q_S) \). Hence, the claim holds.

For the claim below, we introduce the concept of the sub-distance. For any two blank nodes, their sub-distance is inductively defined as:

\textbf{Definition B.3.} For any two blank nodes \( b, b' \), sub-distance \( (b, b') \) is defined inductively as:

- \( \text{sub-distance}(b, b') = 0 \), if \( b' = b \);
- \( \text{sub-distance}(b, b') = \infty \), if \( b \neq b' \) and \( b \) is not a descendant of \( b' \);
- \( \text{sub-distance}(b, b') = \min_{t \in \{ \vec{x}[\mu] \}} \{ \text{sub-distance}(b, t) \} + 1 \), if \( b' \) was generated by application of \( \mu \) on \( r = \text{body}(r)(\vec{x}, \vec{z}) \to \text{head}(r)(\vec{x}, \vec{y}) \), i.e. \( b' = y_j[\mu^{ext}(\vec{b})] \), for some \( y_j \in \{\vec{y}\} \), and \( b \) is a descendant of \( b' \).

\textbf{Claim (3)} For any quad-system \( Q_S = \langle Q_C, R \rangle \), if \( \text{unCSafe} \not\in d\text{Chase}^{csafe}(Q_S) \), then for any two skolem blank nodes \( b, b' \) in \( d\text{Chase}(Q_S) \), if \( b \) is a descendant of \( b' \) then there exists a quad of the form \( c_c : (b, \text{descendantOf}, b') \in d\text{Chase}^{csafe}(Q_S) \).

Note by the definition of sub-distance that if \( b \) is a descendant of \( b' \), then sub-distance \( (b, b') \in \mathbb{N} \). Assuming \( \text{unCSafe} \not\in d\text{Chase}^{csafe}(Q_S) \), and \( b \) is a descendant of \( b' \), we approach the proof by induction on sub-distance \( (b, b') \).

\textbf{Base Case} Suppose \( \text{sub-distance}(b, b') = 1 \), then this implies that there exists \( r = \text{body}(\vec{x}, \vec{z}) \to \text{head}(r)(\vec{x}, \vec{y}) \), assignment \( \mu \) s.t. \( b' \) was generated due to application of \( \mu \) on \( r \), i.e. \( b' = y_j[\mu^{ext}(\vec{b})] \), for some \( y_j \in \{\vec{y}\} \), and \( b \in \{\vec{x}[\mu]\} \). This implies that there exists a dChase iteration \( i \) s.t. \( \text{applicable}_{\text{augC}(R)}(r, \mu, d\text{Chase}(Q_S)) \) and \( d\text{Chase}_{i+1}(Q_S) = d\text{Chase}_i(Q_S) \cup \text{apply}(r, \mu) \). Since \( \text{unCSafe} \not\in d\text{Chase}^{csafe}(Q_S) \), using claim 0, there exists \( k \geq i \) s.t. \( d\text{Chase}_k(Q_S) = d\text{Chase}^{csafe}_k(Q_S)(\mathcal{C}) \). W.l.o.g., let \( k \) be the first such csafe dChase iteration. This means that \( \text{applicable}_{\text{augC}(R)}(r', \mu, d\text{Chase}^{csafe}_{i+j-1}(Q_S)) \), where \( r' = \text{augC}(r) \), and \( d\text{Chase}^{csafe}_{i+j}(Q_S) = d\text{Chase}^{csafe}_{i+j}(Q_S) \). Since there exists a quad-pattern \( c_c : (x_i, \text{descendantOf}, y_i) \in \text{head}(r') \), for any \( x_i \in \{\vec{x}\} \), \( y_i \in \{\vec{y}\} \), it follows that \( c_c : (b, \text{descendantOf}, b') \in d\text{Chase}^{csafe}(Q_S) \).

\textbf{Inductive Step} Suppose sub-distance \( (b, b') = k + 1 \), then there exists a \( b'' \neq b \), assignment \( \mu \), and BR \( r = \text{body}(r)(\vec{x}, \vec{z}) \to \text{head}(r)(\vec{x}, \vec{y}) \in R \) s.t. \( b'' \) was generated due to the application of \( \mu \) or \( r \) with \( b'' \in \{\vec{x}[\mu]\} \), i.e. \( b'' = y_j[\mu^{ext}(\vec{b})] \), for \( y_j \in \{\vec{y}\} \), and \( b \) is a descendant of \( b'' \). This implies that \( \text{sub-distance}(b'', b') = 1 \), and sub-distance \( (b, b'') = k \), and hence by hypothesis \( c_c : (b, \text{descendantOf}, b'') \in d\text{Chase}^{csafe}(Q_S) \), and \( c_c : (b'', \text{descendantOf}, b') \in d\text{Chase}^{csafe}(Q_S) \). Hence, by construction of csafe dChase, \( c_c : (b, \text{descendantOf}, b') \in d\text{Chase}^{csafe}(Q_S) \).
Suppose \( QS_C \) is unsafe, then by definition, there exists a blank nodes \( b, b' \) in \( B_{sk}(dChase(QS_C)) \), s.t. \( b \) is descendant of \( b' \), and \( originContexts(b) = originContexts(b') \).

By contradiction, if \( unCSafe \not\in dChase^{csafe}(QS_C) \), then by claim 1, \( dChase(QS_C) \subseteq dChase^{csafe}(QS_C) \). Since by claim 2, for any \( c \in originContexts(b) \), there exists quads of the form \( c_c: (b, originContext, c) \in dChase^{csafe}(QS_C) \) and for every \( c' \in originContexts(b') \), there exists \( c_c: (b', originContext, c') \in dChase^{csafe}(QS_C) \). Since \( originContexts(b) = originContexts(b') \), it follows that \( \{ c | c_c: (b, originContext, c) \in dChase^{csafe}(QS_C) \} = \{ c' | c_c: (b', originContext, c') \in dChase^{csafe}(QS_C) \} \)

Also by claim 3, since \( b \) is a descendant of \( b' \), there exists a quad of the form \( c_c: (b, descendantOf, b') \in dChase^{csafe}(QS_C) \). But, by construction of \( dChase^{csafe}(QS_C) \), it should be the case that there exist a \( b'' \in B_{sk}(dChase^{csafe}(QS_C)) \), \( r = body(r)(\vec{x}, \vec{y}) \rightarrow head(r)(\vec{x}, \vec{y}) \in augC(R) \), assignment \( \mu \) s.t. \( b' \) was generated due to the application of \( \mu \) on \( r \), i.e. \( b' = y_j[\mu^{ext}(\vec{y})] \) with \( b'' \in \{ \vec{x}[\mu] \} \), and \( c_c: (b, descendantOf, b'') \in dChase^{csafe}(QS_C) \). But, since \( \{ c | c_c: (b, originContext, c) \in dChase^{csafe}(QS_C) \} = cScope(y_j, head(r_i)) \), the method \( unCSafeTest(r, \mu, dChase^{csafe}(QS_C)) \) should return True, for some \( l \in \mathbb{N} \). Hence, it should be the case that \( unCSafe \in dChase^{csafe}(QS_C) \), which is a contradiction to our assumption. Hence \( unCSafe \in dChase^{csafe}(QS_C) \), if \( dChase(QS_C) \) is unsafe.  

\( \Box \)