Knowledge Graph OLAP

Appendix

Christoph G. Schuetz Loris Bozzato Bernd Neumayr
Michael Schrefl Luciano Serafini

Contents

1 Table of symbols

For ease of reference, in Table 1 we summarize the notation used in the main paper [1].

Table 1: Table of symbols used in the main paper

Concept	Symbol	Section
Cube vocabulary	Ω	3.2.1
Cell names	F	3.2.1
Dimensions	D	3.2.1
Levels	L	3.2.1
Dimension members	I	3.2.1
Cube language	$\mathcal{L}_\Omega$	3.2.1
Dimensional ordering	$\prec_{\mathsf{A}}$	3.2.1
Direct dim. ordering	$\dot{\prec}_{\mathsf{A}}$	3.2.1
Multidimensional space	$\mathfrak{D}_\Omega$	3.2.1
Cell name function	cn	3.2.1
Coverage relation	$\preceq$	3.2.1
Object vocabulary	Σ	3.2.1
Object language	$\mathcal{L}_\Sigma$	3.2.1
Meta-vocabulary	Γ	3.2.2
Module names	M	3.2.2
Context classes	C	3.2.2
Contextual relations	R	3.2.2
Contextual attributes	A	3.2.2
Module role	mod	3.2.2
Class of all contexts	Ctx	3.2.2
Null context class	Null	3.2.2
Meta-language	$\mathcal{L}_\Gamma$	3.2.2

Concept	Symbol	Section
Contextualized Knowledge Repository (CKR)	$\mathfrak{K}$	3.2.2
Global context	$\mathfrak{G}$	3.2.2
Knowledge modules (for M)	K_{M}	3.2.2
CKR interpretation	$\mathfrak{I}$	3.2.2
Global interpretation	$\mathcal{M}$	3.2.2
Local interpretation (for c)	$\mathcal{I}(c)$	3.2.2
Input rules set	I	3.2.3
Deduction rules set	P	3.2.3
Output rules set	O	3.2.3
CKR program	$PK(\mathfrak{K})$	3.2.3
Entailment	$\models$	3.2.3
Slice-and-dice	δ	4.1.1
Level vector	$\mathbf{l}$	4.1.2
Level function	lev	4.1.2
Merge (with method met)	ρ^{met}	4.1.2
Abstraction (with method met)	α^{met}	4.2.1
Pivoting	π	4.2.2
Reification	ϱ	4.2.3

2 $\mathcal{SROIQ}$-RL description logic

We summarize the basic definitions for description logics [2] and the logic $\mathcal{SROIQ}$-RL [3, 4] we use in the formulation of the materialization calculus. $\mathcal{SROIQ}$-RL is a restriction of the description logic $\mathcal{SROIQ}$ [5] to the constructs allowed in the OWL-RL profile [6].

A *DL vocabulary* $\Sigma = \mathrm{NC} \uplus \mathrm{NR} \uplus \mathrm{NI}$ is a set of symbols composed of three mutually disjoint countably infinite subsets: the set NC of *atomic concepts*, the set NR of *atomic roles*, and the set NI of *individual constants*.

Complex *concepts* and *roles* for $\mathcal{SROIQ}$-RL are defined using the concept and role constructors in Table 2, where A is an atomic concept, C and D are (possibly complex) concepts, P and R are atomic roles, S and Q are (possibly complex) roles, a and b are individual constants, and n stands for any positive integer. We define the following grammars for a *left-side concept* C and a *right-side concept* D:

$$C := A \mid \{a\} \mid C \sqcap C \mid C \sqcup C \mid \exists R.C \mid \exists R.\{a\} \mid \exists R.\top$$
$$D := A \mid \neg C \mid D \sqcap D \mid \exists R.\{a\} \mid \forall R.D \mid \leqslant nR.\top$$

where $A \in \mathrm{NC}$, $R \in \mathrm{NR}$, $a \in \mathrm{NI}$ and $n \in \{0, 1\}$. A *both-side concept* is a concept expression that is both a left- and right-side concept.

A $\mathcal{SROIQ}$-RL *knowledge base* $\mathcal{K} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ consists of: a TBox $\mathcal{T}$, which contains axioms of the form $C \sqsubseteq D$, where C is a left-side concept and D is a right-side concept or $E \equiv F$, where E and F are both-side concepts; an RBox $\mathcal{R}$, which contains role inclusion axioms $S \sqsubseteq Q$ and role properties axioms in Table 2; an ABox $\mathcal{A}$, containing concept and role assertions: concept assertions are restricted to the form $D(a)$, where D is a right-side concept. The syntax of all axioms is shown at the lower part of Table 2.

A *DL interpretation* is a pair $\mathcal{I} = \left\langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \right\rangle$ where $\Delta^{\mathcal{I}}$ is a non-empty set called interpretation *domain* and $\cdot^{\mathcal{I}}$ is the *interpretation function* which provides denotations for individuals, concepts and roles. The interpretation function $\cdot^{\mathcal{I}}$ assigns an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ to each $a \in \mathrm{NI}$, a subset $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ to each $C \in \mathrm{NC}$, and a subset $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ to each $R \in \mathrm{NR}$. The definition of semantics for complex concept and role constructors is listed in Table 2.

An axiom α is *satisfied* by an interpretation $\mathcal{I}$ (denoted $\mathcal{I} \models_{\mathrm{DL}} \alpha$) if $\mathcal{I}$ satisfies the respective semantic constraint listed in Table 2. An interpretation $\mathcal{I}$ is a *model* of $\mathcal{K}$ (denoted $\mathcal{I} \models_{\mathrm{DL}} \mathcal{K}$) if it satisfies all axioms of $\mathcal{K}$.

Complying to the definitions of $\mathcal{SROIQ}$ [5], only *simple roles* are allowed in the irreflexivity, asymmetry and the disjointness axioms (this is marked with $*$ in Table 2). Simple roles are defined recursively as follows:

– an atomic role R is simple if it does not occur on the right-hand side of a role inclusion axiom in $\mathcal{R}$;

– an inverse role R^{-} is simple if R is simple;

– if R occurs on the right-hand side of a role inclusion axiom in $\mathcal{R}$ and each such role inclusion axiom is of the from $S \sqsubseteq R$ where S is a simple role, than R is also simple.

Moreover, $\mathcal{SROIQ}$ RBox [5] is required to be *regular* to preserve decidability. Formally, a *regular order* is a strict partial order $\prec$ on roles s.t., for any roles R, S, $R \prec S$ iff $R^{-} \prec S$. A RIA is $\prec$-*regular* if it is in one of the following forms: (i) $R \circ R \sqsubseteq R$; (ii) $R^{-} \sqsubseteq R$; (iii) $S_1 \circ \ldots \circ S_n \sqsubseteq R$ with $S_i \prec R$ for $i \in \{1, \ldots, n\}$; (iv) $R \circ S_1 \circ \ldots \circ S_n \sqsubseteq R$ with $S_i \prec R$ for $i \in \{1, \ldots, n\}$; (v) $S_1 \circ \ldots \circ S_n \circ R \sqsubseteq R$ with $S_i \prec R$ for $i \in \{1, \ldots, n\}$. An RBox $\mathcal{R}$ is *regular*, if there exists a regular order $\prec$ such that all role inclusions in $\mathcal{R}$ are $\prec$-regular.

Table 2: Syntax and Semantics of $\mathcal{SROIQ}$-RL constructors

Concept constructors	Syntax	Semantics
Atomic concept	A	$A^{\mathcal{I}}$
Top concept	$\top$	$\Delta^{\mathcal{I}}$
Bottom concept	$\bot$	$\emptyset$
Complement	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
Intersection	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
Union	$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
Existential restriction	$\exists R.C$	$\left\{ x \in \Delta^{\mathcal{I}} \,\middle\vert\, \begin{array}{l} \exists y.\, \langle x, y \rangle \in R^{\mathcal{I}} \\ \wedge\, y \in C^{\mathcal{I}} \end{array} \right\}$
Universal restriction	$\forall R.D$	$\left\{ x \in \Delta^{\mathcal{I}} \,\middle\vert\, \begin{array}{l} \forall y.\, \langle x, y \rangle \in R^{\mathcal{I}} \\ \rightarrow y \in D^{\mathcal{I}} \end{array} \right\}$
Max. card. restriction	$\leqslant n R.C$	$\left\{ x \in \Delta^{\mathcal{I}} \,\middle\vert\, \begin{array}{l} \sharp\{y \vert\, \langle x, y \rangle \in R^{\mathcal{I}} \\ \wedge\, y \in C^{\mathcal{I}}\} \leq n \end{array} \right\}$
Nominal	$\{a\}$	$\{a^{\mathcal{I}}\}$
Role constructors	Syntax	Semantics
Atomic role	R	$R^{\mathcal{I}}$
Inverse role	R^-	$\{\langle y, x \rangle \mid \langle x, y \rangle \in R^{\mathcal{I}}\}$
Role composition	$S \circ Q$	$\{\langle x, z \rangle \mid \langle x, y \rangle \in S^{\mathcal{I}}, \langle y, z \rangle \in Q^{\mathcal{I}}\}$
Axioms	Syntax	Semantics
Concept inclusion (GCI)	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
Concept definition	$C \equiv D$	$C^{\mathcal{I}} = D^{\mathcal{I}}$
Role inclusion (RIA)	$S \sqsubseteq R$	$S^{\mathcal{I}} \subseteq R^{\mathcal{I}}$
Role disjointness*	$\mathrm{Dis}(P, R)$	$P^{\mathcal{I}} \cap R^{\mathcal{I}} = \emptyset$
Irreflexivity assertion*	$\mathrm{Irr}(R)$	$R^{\mathcal{I}} \cap \{\langle x, x \rangle \mid x \in \Delta^{\mathcal{I}}\} = \emptyset$
Symmetry assertion	$\mathrm{Sym}(R)$	$\langle x, y \rangle \in R^{\mathcal{I}} \Rightarrow \langle y, x \rangle \in R^{\mathcal{I}}$
Asymmetry assertion*	$\mathrm{Asym}(R)$	$\langle x, y \rangle \in R^{\mathcal{I}} \Rightarrow \langle y, x \rangle \notin R^{\mathcal{I}}$
Transitivity assertion	$\mathrm{Trans}(R)$	$\{\langle x, y \rangle, \langle y, z \rangle\} \subseteq R^{\mathcal{I}} \Rightarrow \langle x, z \rangle \in R^{\mathcal{I}}$
Concept assertion	$D(a)$	$a^{\mathcal{I}} \in D^{\mathcal{I}}$
Role assertion	$R(a, b)$	$\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in R^{\mathcal{I}}$
Negated role assertion	$\neg R(a, b)$	$\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \notin R^{\mathcal{I}}$
Equality assertion	$a = b$	$a^{\mathcal{I}} = b^{\mathcal{I}}$
Inequality assertion	$a \neq b$	$a^{\mathcal{I}} \neq b^{\mathcal{I}}$

3 Materialization calculus

In this section, we provide the formal definition for the datalog translation for instance checking in KG-OLAP cubes introduced in Section 3.2.3.

3.1 Calculus definition

We summarize the definitions of the materialization calculus for $\mathcal{SROIQ}$-RL based CKRs originally introduced in [3] (which, in turn, adapts the materialization calculus definitions presented in [7]) and we adapt them to the structure of KG-OLAP cube provided in the main paper.

Normal form. To simplify the presentation of the calculus rules, $\mathcal{SROIQ}$-RL axioms considered in the rules are assumed to appear in normal form. We say that a CKR $\mathfrak{K} = \langle \mathfrak{G}, \mathrm{K}_{\mathsf{M}} \rangle$ is in *normal form* if:

Table 3: Normal form axioms

$A(a)$	$R(a,b)$	$\neg R(a,b)$	$a = b$	$a \neq b$
$A \sqsubseteq B$	$\{a\} \sqsubseteq B$	$A \sqsubseteq \neg B$	$A \sqcap B \sqsubseteq C$	
$\exists R.A \sqsubseteq B$	$A \sqsubseteq \exists R.\{a\}$	$A \sqsubseteq \forall R.B$	$A \sqsubseteq \leqslant 1 R.\top$	
$R \sqsubseteq T$	$R \circ S \sqsubseteq T$	$\mathrm{Dis}(R,S)$	$\mathrm{Inv}(R,S)$	$\mathrm{Irr}(R)$

- $\mathfrak{G}$ contains axioms in $\mathcal{L}_\Gamma$ of the form of Table 3 or in the form $\mathsf{C} \sqsubseteq \exists \mathsf{mod}.\{\mathsf{m}\}$, $\mathsf{C} \sqsubseteq \exists \mathsf{A}.\{\mathsf{d_A}\}$ for $A, B, C \in \mathbf{C}$, $R, S, T \in \mathbf{R}$, $a, b \in \mathbf{N}$, $\mathsf{m} \in \mathbf{M}$, $\mathsf{A} \in \mathbf{A}$ and $\mathsf{d_A} \in \mathsf{D_A}$.
- $\mathfrak{G}$ and every K_m contain axioms in $\mathcal{L}_\Sigma$ of the form of Table 3 for $A, B, C \in \mathrm{NC}$, $a, b \in \mathrm{NI}$, $R, S, T \in \mathrm{NR}$.

In [3, 4] we provide a set of rules that allow to transform any $\mathcal{SROIQ}$-RL CKR in an "equivalent" CKR in normal form[1] and we state the correctness of such translation (which can be proved similarly to the case of [7]).

Translation language and proofs. We follow the same presentation given in [3, 7], and we will thus express our rules in the language of *datalog*. We summarize here the basic definitions of the language. A *signature* is a tuple $\langle \mathbf{C}, \mathbf{P} \rangle$, with $\mathbf{C}$ a finite set of *constants* and $\mathbf{P}$ a finite set of *predicates*. We assume a set $\mathbf{V}$ of *variables* and we call *terms* the elements of $\mathbf{C} \cup \mathbf{V}$. An *atom* over $\langle \mathbf{C}, \mathbf{P} \rangle$ is in the form $p(t_1, \ldots, t_n)$ with $p \in \mathbf{P}$ and every $t_i \in \mathbf{C} \cup \mathbf{V}$ for $i \in \{1, \ldots, n\}$. A *rule* is an expression in the form $B_1, \ldots, B_m \to H$ where H and $B_1, \ldots, B_m$ are datalog atoms (the *head* and *body* of the rule). A *fact* H is a ground rule with empty body. A *program* P is a finite set of datalog rules. A *ground substitution* σ for $\langle \mathbf{C}, \mathbf{P} \rangle$ is a function $\sigma : \mathbf{V} \to \mathbf{C}$. We define as usual substitutions on atoms and *ground instances* of atoms. A *proof tree* for P is a structure $\langle N, E, \lambda \rangle$ where $\langle N, E \rangle$ is a finite directed tree and λ is a labelling function assigning a ground atom to each node, where: for each $v \in N$, there exists a rule $B_1, \ldots, B_m \to H$ in P and a ground substitution σ s.t. (i) $\lambda(v) = \sigma(H)$ and (ii) v has m child nodes w_i in E, with $\lambda(w_i) = \sigma(B_i)$ for $i \in \{1, \ldots, m\}$. A ground atom H is a *consequence* of P (denoted $P \models H$) if there exists a proof tree for P with root node r and with $\lambda(r) = H$.

[1] As in [7], we assume that rule chain axioms in input are already decomposed in binary role chains.

3.2 Calculus rules

As introduced in Section 3.2.3 in the main paper, the calculus has three components:

(a). *Input translations* I_{glob}, I_{rl}: Given an axiom α and $\mathsf{c} \in \mathbf{F}$, each $I(\alpha, \mathsf{c})$ is a set of datalog facts and rules. Intuitively, the datalog facts and rules encode as datalog facts the contents of input global and local DL knowledge bases.

(b). *Deduction rules* P_{glob}, P_{loc}, P_{rl}: Sets of datalog rules represent the inference rules for the instance-level reasoning over the translated axioms.

(c). *Output translation* O: Given an axiom α and $\mathsf{c} \in \mathbf{F}$, $O(\alpha, \mathsf{c})$ is a single datalog fact which encodes the ABox assertion α that we want to prove to be c-entailed by the input CKR.

In the following, we briefly present the form of the different sets of translation and deduction rules. The complete set of rules is presented in Table 4.

(i). *$\mathcal{SROIQ}$-RL translation*: The rules in $I_{rl}(S, c)$ translate into datalog facts $\mathcal{SROIQ}$-RL axioms (in a context c). For example, we translate atomic concept inclusions with the rule $A \sqsubseteq B \mapsto \{\mathtt{subClass}(A, B, c)\}$. The rules in P_{rl} are the deduction rules corresponding to axioms in $\mathcal{SROIQ}$-RL. For example, for atomic concept inclusions we have:

$$\mathtt{subClass}(y, z, c), \mathtt{inst}(x, y, c) \to \mathtt{inst}(x, z, c)$$

(ii). *Global and local translations*: The global input rules of I_{glob} encode the interpretation of Ctx in the global context and (new in this version of the materialization calculus) the translation of the coverage in the level and dimensional hierarchies. The global deduction rules in P_{glob} provide the rule for the propagation of modules, which is defined by the following, with gm the context name of global metaknowledge:

$$\begin{aligned}&\mathtt{triple}(\mathsf{c}_1, \mathsf{covers}, \mathsf{c}_2, \mathsf{gm}),\\ &\quad \mathtt{triple}(\mathsf{c}_1, \mathsf{mod}, m, \mathsf{gm}) \to \mathtt{triple}(\mathsf{c}_2, \mathsf{mod}, m, \mathsf{gm})\end{aligned}$$

Similarly, the local deduction rules P_{loc} provide the rules for the interpretation of the local object language.

(iii). *Output rules*: The rules in $O(\alpha, \mathsf{c})$ provide the translation of ABox assertions that can be verified to hold in context c by applying the rules of the final program. For example, atomic concept assertions in a context c are translated by $A(a) \mapsto \{\mathtt{inst}(a, A, \mathsf{c})\}$.

3.3 Translation process

Given a $\mathcal{SROIQ}$-RL cube $\mathfrak{K} = \langle \mathfrak{G}, \mathrm{K}_{\mathbf{M}} \rangle$ in normal form, the translation to its datalog program $PK(\mathfrak{K})$ now proceeds along the following steps:

(1). The initial *global program* for $\mathfrak{G}$ is translated into:

$$PG(\mathfrak{G}) = I_{glob}(\mathfrak{G}_\Gamma) \cup I_{rl}(\mathfrak{G}_\Gamma, \mathsf{gm}) \cup I_{rl}(\mathfrak{G}_\Sigma, \mathsf{gk}) \cup P_{rl} \cup P_{glob}$$

with gm, gk new context names, $\mathfrak{G}_\Gamma = \{\alpha \in \mathfrak{G} \mid \alpha \in \mathcal{L}_\Gamma\}$ and $\mathfrak{G}_\Sigma = \{\alpha \in \mathfrak{G} \mid \alpha \in \mathcal{L}_\Sigma\}$.

(2). We compute the names and the dimensions associated to each context. For all $\mathbf{d} \in \mathfrak{D}_\Omega$, with $\mathsf{cn}(\mathbf{d}) = \mathsf{c}$, let $S(\mathbf{d}) = \{\mathtt{inst}(\mathsf{c}, \mathsf{Ctx}, \mathsf{gm})\} \cup \bigcup_{A \in \mathbf{A}} \{\mathtt{triple}(\mathsf{c}, A, d_A, \mathsf{gm})\}$. Then:

$$PG'(\mathfrak{G}) = PG(\mathfrak{G}) \cup \bigcup_{\mathbf{d} \in \mathfrak{D}_\Omega} S(\mathbf{d})$$

Table 4: Calculus input, output and deduction rules

Global input rules $I_{glob}(\mathfrak{G})$

(igl-subctx1)	$\mathsf{C} \in \mathbf{C} \mapsto \{\mathtt{subClass}(\mathsf{C}, \mathsf{Ctx}, \mathsf{gm})\}$
(igl-subctx2)	$\mathsf{c} \in \mathbf{N} \mapsto \{\mathtt{inst}(\mathsf{c}, \mathsf{Ctx}, \mathsf{gm})\}$
(igl-coverdim)	$d \prec_A d' \mapsto \{\ \mathtt{triple}(d, \mathtt{rollsUpTo}, d', \mathsf{gm})\ \}$
(igl-coverlvl)	$l \prec_A^L l' \mapsto \{\ \mathtt{triple}(l, \mathtt{rollsUpTo}, l', \mathsf{gm})\ \}$
(igl-ortho)	d orthogonal to $d' \mapsto \{\ \mathtt{triple}(d, \mathtt{ortho}, d', \mathsf{gm})\ \}$

Global deduction rules P_{glob}

(pgl-prop)	$\mathtt{triple}(\mathsf{c}_1, \mathtt{covers}, \mathsf{c}_2, \mathsf{gm}), \mathtt{triple}(\mathsf{c}_1, \mathsf{mod}, m, \mathsf{gm}) \rightarrow \mathtt{triple}(\mathsf{c}_2, \mathsf{mod}, m, \mathsf{gm})$

Local deduction rules P_{loc}

(plc-eq)	$\mathtt{nom}(x, c), \mathtt{eq}(x, y, c') \rightarrow \mathtt{eq}(x, y, c)$

Output translation $O(\alpha, \mathsf{c})$

(o-concept)	$A(a) \mapsto \{\mathtt{inst}(a, A, \mathsf{c})\}$
(o-role)	$R(a, b) \mapsto \{\mathtt{triple}(a, R, b, \mathsf{c})\}$

RL input translation $I_{rl}(S, c)$

(irl-nom)	$a \in \mathrm{NI} \mapsto \{\mathtt{nom}(a, c)\}$
(irl-cls)	$A \in \mathrm{NC} \mapsto \{\mathtt{cls}(A, c)\}$
(irl-rol)	$R \in \mathrm{NR} \mapsto \{\mathtt{rol}(R, c)\}$
(irl-inst1)	$A(a) \mapsto \{\mathtt{inst}(a, A, c)\}$
(irl-triple)	$R(a, b) \mapsto \{\mathtt{triple}(a, R, b, c)\}$
(irl-ntriple)	$\neg R(a, b) \mapsto \{\mathtt{negtriple}(a, R, b, c)\}$
(irl-eq)	$a = b \mapsto \{\mathtt{eq}(a, b, c)\}$
(irl-neq)	$a \neq b \mapsto \{\mathtt{neq}(a, b, c)\}$
(irl-inst2)	$\{a\} \sqsubseteq B \mapsto \{\mathtt{inst}(a, B, c)\}$
(irl-subc)	$A \sqsubseteq B \mapsto \{\mathtt{subClass}(A, B, c)\}$
(irl-top)	$\top(a) \mapsto \{\mathtt{inst}(a, \mathtt{top}, c)\}$
(irl-bot)	$\bot(a) \mapsto \{\mathtt{inst}(a, \mathtt{bot}, c)\}$
(irl-not)	$A \sqsubseteq \neg B \mapsto \{\mathtt{supNot}(A, B, c)\}$
(irl-subcnj)	$A_1 \sqcap A_2 \sqsubseteq B \mapsto \{\mathtt{subConj}(A_1, A_2, B, c)\}$
(irl-subex)	$\exists R.A \sqsubseteq B \mapsto \{\mathtt{subEx}(R, A, B, c)\}$
(irl-supex)	$A \sqsubseteq \exists R.\{a\} \mapsto \{\mathtt{supEx}(A, R, a, c)\}$
(irl-forall)	$A \sqsubseteq \forall R.B \mapsto \{\mathtt{supForall}(A, R, B, c)\}$
(irl-leqone)	$A \sqsubseteq \leqslant 1 R.\top \mapsto \{\mathtt{supLeqOne}(A, R, c)\}$
(irl-subr)	$R \sqsubseteq S \mapsto \{\mathtt{subRole}(R, S, c)\}$
(irl-subrc)	$R \circ S \sqsubseteq T \mapsto \{\mathtt{subRChain}(R, S, T, c)\}$
(irl-dis)	$\mathrm{Dis}(R, S) \mapsto \{\mathtt{dis}(R, S, c)\}$
(irl-inv)	$\mathrm{Inv}(R, S) \mapsto \{\mathtt{inv}(R, S, c)\}$
(irl-irr)	$\mathrm{Irr}(R) \mapsto \{\mathtt{irr}(R, c)\}$

RL deduction rules P_{rl}

(prl-ntriple)	$\mathtt{negtriple}(x, v, y, c), \mathtt{triple}(x, v, y, c) \rightarrow \mathtt{inst}(x, \mathtt{bot}, c)$
(prl-eq1)	$\mathtt{nom}(x, c) \rightarrow \mathtt{eq}(x, x, c)$
(prl-eq2)	$\mathtt{eq}(x, y, c) \rightarrow \mathtt{eq}(y, x, c)$
(prl-eq3)	$\mathtt{eq}(x, y, c), \mathtt{inst}(x, z, c) \rightarrow \mathtt{inst}(y, z, c)$
(prl-eq4)	$\mathtt{eq}(x, y, c), \mathtt{triple}(x, u, z, c) \rightarrow \mathtt{triple}(y, u, z, c)$
(prl-eq5)	$\mathtt{eq}(x, y, c), \mathtt{triple}(z, u, x, c) \rightarrow \mathtt{triple}(z, u, y, c)$
(prl-eq6)	$\mathtt{eq}(x, y, c), \mathtt{eq}(y, z, c) \rightarrow \mathtt{eq}(x, z, c)$
(prl-neq)	$\mathtt{eq}(x, y, c), \mathtt{neq}(x, y, c) \rightarrow \mathtt{inst}(x, \mathtt{bot}, c)$
(prl-top)	$\mathtt{inst}(x, z, c) \rightarrow \mathtt{inst}(x, \mathtt{top}, c)$
(prl-subc)	$\mathtt{subClass}(y, z, c), \mathtt{inst}(x, y, c) \rightarrow \mathtt{inst}(x, z, c)$
(prl-not)	$\mathtt{supNot}(y, z, c), \mathtt{inst}(x, y, c), \mathtt{inst}(x, z, c) \rightarrow \mathtt{inst}(x, \mathtt{bot}, c)$
(prl-subcnj)	$\mathtt{subConj}(y_1, y_2, z, c), \mathtt{inst}(x, y_1, c), \mathtt{inst}(x, y_2, c) \rightarrow \mathtt{inst}(x, z, c)$
(prl-subex)	$\mathtt{subEx}(v, y, z, c), \mathtt{triple}(x, v, x', c), \mathtt{inst}(x', y, c) \rightarrow \mathtt{inst}(x, z, c)$
(prl-supex)	$\mathtt{supEx}(y, r, x', c), \mathtt{inst}(x, y, c) \rightarrow \mathtt{triple}(x, r, x', c)$
(prl-supforall)	$\mathtt{supForall}(z, r, z', c), \mathtt{inst}(x, z, c), \mathtt{triple}(x, r, y, c) \rightarrow \mathtt{inst}(y, z', c)$
(prl-leqone)	$\mathtt{supLeqOne}(z, r, c), \mathtt{inst}(x, z, c),$ $\mathtt{triple}(x, r, x_1, c), \mathtt{triple}(x, r, x_2, c) \rightarrow \mathtt{eq}(x_1, x_2, c)$
(prl-subr)	$\mathtt{subRole}(v, w, c), \mathtt{triple}(x, v, x', c) \rightarrow \mathtt{triple}(x, w, x', c)$
(prl-subrc)	$\mathtt{subRChain}(u, v, w, c), \mathtt{triple}(x, u, y, c), \mathtt{triple}(y, v, z, c) \rightarrow \mathtt{triple}(x, w, z, c)$
(prl-dis)	$\mathtt{dis}(u, v, c), \mathtt{triple}(x, u, y, c), \mathtt{triple}(x, v, y, c) \rightarrow \mathtt{inst}(x, \mathtt{bot}, c)$
(prl-inv1)	$\mathtt{inv}(u, v, c), \mathtt{triple}(x, u, y, c) \rightarrow \mathtt{triple}(y, v, x, c)$
(prl-inv2)	$\mathtt{inv}(u, v, c), \mathtt{triple}(x, v, y, c) \rightarrow \mathtt{triple}(y, u, x, c)$
(prl-irr)	$\mathtt{irr}(u, c), \mathtt{triple}(x, u, x, c) \rightarrow \mathtt{inst}(x, \mathtt{bot}, c)$

(3). We compute the coverage across contexts. Let $PG''(\mathfrak{G})$ be the program obtained from $PG'(\mathfrak{G})$ by adding

$$\mathtt{triple}(\mathsf{c}_1, \mathtt{covers}, \mathsf{c}_2, \mathsf{gm})$$

if for each $A \in \mathbf{A}$, $PG'(\mathfrak{G}) \models \mathtt{triple}(\mathsf{c}_1, A, d_A, \mathsf{gm})$, $PG'(\mathfrak{G}) \models \mathtt{triple}(\mathsf{c}_2, A, e_A, \mathsf{gm})$ and it holds $PG'(\mathfrak{G}) \models \mathtt{triple}(d_A, \mathtt{rollsUpTo}, e_A, \mathsf{gm})$ or $\mathtt{eq}(d_A, e_A, \mathsf{gm})$.

(4). We define the set of contexts

$$\mathbf{F}_{\mathfrak{G}} = \{\mathsf{c} \in \mathbf{F} \mid PG''(\mathfrak{G}) \models \mathtt{inst}(\mathsf{c}, \mathsf{Ctx}, \mathsf{gm}) \text{ and } PG''(\mathfrak{G}) \not\models \mathtt{inst}(\mathsf{c}, \mathsf{Null}, \mathsf{gm})\}$$

For every $\mathsf{c} \in \mathbf{F}_{\mathfrak{G}}$, we define its associated knowledge base:

$$\mathrm{K}_{\mathsf{c}} = \bigcup \{\mathrm{K}_{\mathsf{m}} \in \mathfrak{K} \mid PG(\mathfrak{G}) \models \mathtt{triple}(\mathsf{c}, \mathsf{mod}, \mathsf{m}, \mathsf{gm})\}$$

(5). We define each *local program* for $\mathsf{c} \in \mathbf{F}_{\mathfrak{G}}$ as:

$$PC(\mathsf{c}) := P_{loc} \cup I_{rl}(\mathrm{K}_{\mathsf{c}}, \mathsf{c}) \cup I_{rl}(\mathfrak{G}_{\Sigma}, \mathsf{c})$$

(6). The final *CKR program* is then defined as:

$$PK(\mathfrak{K}) = PG''(\mathfrak{G}) \cup \textstyle\bigcup_{\mathsf{c} \in \mathbf{F}_{\mathfrak{G}}} PC(\mathsf{c})$$

3.4 Calculus correctness proofs

The correctness of the proposed calculus with respect to c-entailment can be proved by easily adapting the proofs provided in [3] to the newly added model conditions, calculus rules and translation procedure. In the following we report the proofs highlighting, when needed, the new modifications.

3.4.1 Soundness

Lemma 1. *Given $\mathfrak{K} = \langle \mathfrak{G}, \mathrm{K}_{\mathbf{M}} \rangle$ a $\mathcal{SROIQ}$-RL cube in normal form, and $\alpha \in \mathcal{L}_{\Gamma}$ (resp. $\alpha \in \mathcal{L}_{\Sigma}$) with $O(\alpha, \mathsf{g})$ defined and $\mathsf{g} = \mathsf{gm}$ (resp. $\mathsf{g} = \mathsf{gk}$). Then, $PG''(\mathfrak{G}) \models O(\alpha, \mathsf{g})$ implies $\mathfrak{G} \models \alpha$.*

Proof. We follow the proof schema used for proving soundness in [3, 7], by adapting it to the rules of our calculus. We can assign an interpretation to the ground atoms derived from $PG''(\mathfrak{G})$ (i.e. the final global program) as follows, where $\mathsf{g} = \mathsf{gm}$ or $\mathsf{g} = \mathsf{gk}$:

- $\mathtt{inst}(a, A, \mathsf{g})$ with $a \in \mathrm{NI}_{\Gamma} \cup \mathrm{NI}_{\Sigma}$, $A \in \mathrm{NC}_{\Gamma} \cup \mathrm{NC}_{\Sigma}$, then $\mathfrak{G} \models A(a)$;
- $\mathtt{inst}(a, \mathtt{top}, \mathsf{g})$ with $a \in \mathrm{NI}_{\Gamma} \cup \mathrm{NI}_{\Sigma}$, then $\mathfrak{G} \models \top(a)$;
- $\mathtt{inst}(a, \mathtt{bot}, \mathsf{g})$ with $a \in \mathrm{NI}_{\Gamma} \cup \mathrm{NI}_{\Sigma}$, then $\mathfrak{G} \models \bot(a)$;
- $\mathtt{triple}(a, R, b, \mathsf{g})$ with $a, b \in \mathrm{NI}_{\Gamma} \cup \mathrm{NI}_{\Sigma}$, $R \in \mathrm{NR}_{\Gamma} \cup \mathrm{NR}_{\Sigma}$, then $\mathfrak{G} \models R(a, b)$;
- $\mathtt{eq}(a, b, \mathsf{g})$ with $a, b \in \mathrm{NI}_{\Gamma} \cup \mathrm{NI}_{\Sigma}$, then $\mathfrak{G} \models a = b$;
- $\mathtt{neq}(a, b, \mathsf{g})$ with $a, b \in \mathrm{NI}_{\Gamma} \cup \mathrm{NI}_{\Sigma}$, then $\mathfrak{G} \models a \neq b$;

We claim that, for any ground atom H of the above form with the corresponding semantic condition $C(H)$, $PG''(\mathfrak{G}) \models H$ implies $\mathfrak{G} \models C(H)$.

We can prove the claim by induction on the possible proof tree of the above atoms H: we show some representative cases (including the newly added rule in P_{glob}).

– **(prl-eq1):** then $H = \mathtt{eq}(a, a, \mathrm{g})$ and, by I_{rl} rules, $a \in \mathrm{NI}_\Gamma \cup \mathrm{NI}_\Sigma$. For any model $\mathcal{M}$ of $\mathfrak{G}$, for any $a \in \mathrm{NI}_\Gamma \cup \mathrm{NI}_\Sigma$ it holds that $a^\mathcal{M} = a^\mathcal{M}$, thus this verifies $\mathfrak{G} \models (a = a)$.

– **(prl-eq2):** then $H = \mathtt{eq}(b, a, \mathrm{g})$ and $PG''(\mathfrak{G}) \models \mathtt{eq}(a, b, \mathrm{g})$. By the above interpretation of atoms, $\mathfrak{G} \models (a = b)$: by symmetricity of equality relation this directly implies that $\mathfrak{G} \models (b = a)$.

– **(prl-eq3):** then $H = \mathtt{inst}(b, B, \mathrm{g})$ and $PG''(\mathfrak{G}) \models \mathtt{eq}(a, b, \mathrm{g})$, $PG''(\mathfrak{G}) \models \mathtt{inst}(a, B, \mathrm{g})$. By the above interpretation of atoms, $\mathfrak{G} \models (a = b)$ and $\mathfrak{G} \models B(a)$. This directly implies that $\mathfrak{G} \models B(b)$, thus proving the assertion.

– **(prl-neq):** then $H = \mathtt{inst}(a, \mathtt{bot}, \mathrm{g})$ and $PG''(\mathfrak{G}) \models \mathtt{neq}(a, b, \mathrm{g})$, $PG''(\mathfrak{G}) \models \mathtt{eq}(a, b, \mathrm{g})$. By induction hypothesis and the above semantic conditions, we obtain $\mathfrak{G} \models (a = b)$ and $\mathfrak{G} \models (a \neq b)$. This is an absurd, thus there cannot be an interpretation satisfying $\mathfrak{G}$: this justifies the consequence $\mathfrak{G} \models \bot(a)$.

– **(prl-subc):** then $H = \mathtt{inst}(a, B, \mathrm{g})$, $A \sqsubseteq B \in \mathfrak{G}$ and $PG''(\mathfrak{G}) \models \mathtt{inst}(a, A, \mathrm{g})$. By the above semantic conditions, $\mathfrak{G} \models A(a)$: this directly implies that $\mathfrak{G} \models B(a)$.

– **(prl-subex):** then $H = \mathtt{inst}(a, B, \mathrm{g})$, $\exists R.A \sqsubseteq B \in \mathfrak{G}$ and $PG''(\mathfrak{G}) \models \mathtt{triple}(a, R, b, \mathrm{g})$, $PG''(\mathfrak{G}) \models \mathtt{inst}(b, A, \mathrm{g})$. By induction hypothesis, this implies that $\mathfrak{G} \models R(a, b)$ and $\mathfrak{G} \models A(b)$: by definition of the semantics, this proves that $\mathfrak{G} \models (\exists R.A)(a)$ which implies $\mathfrak{G} \models B(a)$.

– **(prl-leqone):** then $H = \mathtt{eq}(b, c, \mathrm{g})$, $A \sqsubseteq \leqslant 1R.\top \in \mathfrak{G}$. Moreover, $PG''(\mathfrak{G}) \models \mathtt{inst}(a, A, \mathrm{g})$, $PG''(\mathfrak{G}) \models \mathtt{triple}(a, R, b, \mathrm{g})$ and $PG''(\mathfrak{G}) \models \mathtt{triple}(a, R, c, \mathrm{g})$. By induction hypothesis, $\mathfrak{G} \models A(a)$ and thus $\mathfrak{G} \models (\leqslant 1R.\top)(a)$. Moreover, $\mathfrak{G} \models R(a, b)$ and $\mathfrak{G} \models R(a, c)$. By definition of the semantics, for every model $\mathcal{M}$ of $\mathfrak{G}$, it holds that $b^\mathcal{M} = c^\mathcal{M}$, which implies $\mathfrak{G} \models (b = c)$.

– **(prl-subr):** then $H = \mathtt{triple}(a, S, b, \mathrm{g})$, $R \sqsubseteq S \in \mathfrak{G}$ and $PG''(\mathfrak{G}) \models \mathtt{triple}(a, R, b, \mathrm{g})$. By the above semantic constraints, $\mathfrak{G} \models R(a, b)$ which directly implies $\mathfrak{G} \models S(a, b)$.

– **(prl-subrc):** then $H = \mathtt{triple}(a, T, b, \mathrm{g})$, $R \circ S \sqsubseteq T \in \mathfrak{G}$ and $PG''(\mathfrak{G}) \models \mathtt{triple}(a, R, c, \mathrm{g})$, $PG''(\mathfrak{G}) \models \mathtt{triple}(c, S, b, \mathrm{g})$. By the above semantic constraints, $\mathfrak{G} \models R(a, c)$ and $\mathfrak{G} \models S(c, b)$: by definition of the semantics, this implies that $\mathfrak{G} \models T(a, b)$.

– **(pgl-prop):** then $H = \mathtt{triple}(\mathsf{c}_2, \mathsf{mod}, \mathsf{m}, \mathsf{gm})$ and it holds that $PG''(\mathfrak{G}) \models \mathtt{triple}(\mathsf{c}_1, \mathsf{covers}, \mathsf{c}_2, \mathsf{gm})$, $\mathtt{triple}(\mathsf{c}_1, \mathsf{mod}, \mathsf{m}, \mathsf{gm})$. Then, by the above semantic conditions, $\mathfrak{G} \models \mathsf{c}_2 \prec \mathsf{c}_1$ and $\mathfrak{G} \models \mathsf{mod}(\mathsf{c}_1, \mathsf{m})$. By the semantic conditions of KG-OLAP cube models (Definition 3.(vi)), this implies that $\mathfrak{G} \models \mathsf{mod}(\mathsf{c}_2, \mathsf{m})$.

With respect to the construction of $PG''(\mathfrak{G})$, we also note that by this result we obtain that the syntactic definition of cn and coverage are correctly interpreted: if $\mathsf{cn}(\mathbf{d}) = \mathsf{c}$, then it holds that $\mathfrak{G} \models \mathsf{Ctx}(\mathsf{c})$ and $\mathfrak{G} \models \mathsf{A}(\mathsf{c}, \mathsf{d}_\mathsf{A})$ for every dimension $\mathsf{A} \in \mathbf{D}$; $\mathfrak{G} \models \mathsf{c}_1 \preceq \mathsf{c}_2$ iff, for every $\mathsf{A} \in \mathbf{D}$, if $\mathfrak{G} \models \mathsf{A}(\mathsf{c}_1, \mathsf{d}_\mathsf{A})$ and $\mathfrak{G} \models \mathsf{A}(\mathsf{c}_2, \mathsf{e}_\mathsf{A})$ then either $\mathfrak{G} \models \mathsf{d}_\mathsf{A} \prec \mathsf{e}_\mathsf{A}$ or $\mathfrak{G} \models \mathsf{d}_\mathsf{A} = \mathsf{e}_\mathsf{A}$. Moreover, note that the ordering relations across dimensions, levels and cells are managed by the common rules for roles, role chains and their properties. $\square$

Theorem 1 (Soundness)**.** Given $\mathfrak{K} = \langle \mathfrak{G}, \mathrm{K}_{\mathsf{M}} \rangle$ a $\mathcal{SROIQ}$-RL cube in normal form, $\alpha \in \mathcal{L}_\Sigma$ and $\mathsf{c} \in \mathbf{F}$ s.t. $O(\alpha, \mathsf{c})$ is defined. Then, $PK(\mathfrak{K}) \models O(\alpha, \mathsf{c})$ implies $\mathfrak{K} \models \mathsf{c} : \alpha$.

Proof. To prove the assertion, we extend the construction of the previous Lemma 1 to the local interpretations for contexts and the definition of the program representing the whole input KG-OLAP cube.

For Lemma 1, we can also easily derive that, for every interpretation $\mathcal{M}$ such that $\mathcal{M} \models \mathfrak{G}$: if $\mathsf{c} \in \mathbf{F}_\mathfrak{G}$ (that is, if $PG''(\mathfrak{G}) \models \mathtt{inst}(\mathsf{c}, \mathsf{Ctx}, \mathsf{gm})$) then $\mathsf{c}^\mathcal{M} \in \mathsf{Ctx}^\mathcal{M}$; if $\mathrm{K}_\mathsf{m} \in \mathrm{K}_\mathsf{c}$ (that is, if $PG''(\mathfrak{G}) \models \mathtt{triple}(\mathsf{c}, \mathsf{mod}, \mathsf{m}, \mathsf{gm})$ and $PG''(\mathfrak{G}) \not\models \mathtt{inst}(\mathsf{c}, \mathsf{Null}, \mathsf{gm})$) then $\langle \mathsf{c}^\mathcal{M}, \mathsf{m}^\mathcal{M} \rangle \in \mathsf{mod}^\mathcal{M}$.

As in previous lemma, we can assign a semantic constraint to the ground atoms derived from $PK(\mathfrak{K})$ as follows, where $\mathsf{c} \in \mathbf{F}_\mathfrak{G}$:

- $\mathtt{inst}(a, A, \mathsf{c})$ with $a \in \mathrm{NI}_\Sigma$, $A \in \mathrm{NC}_\Sigma$, then $\mathfrak{K} \models \mathsf{c} : A(a)$;
- $\mathtt{inst}(a, \mathtt{top}, \mathsf{c})$ with $a \in \mathrm{NI}_\Sigma$, then $\mathfrak{K} \models \mathsf{c} : \top(a)$;
- $\mathtt{inst}(a, \mathtt{bot}, \mathsf{c})$ with $a \in \mathrm{NI}_\Sigma$, then $\mathfrak{K} \models \mathsf{c} : \bot(a)$;
- $\mathtt{triple}(a, R, b, \mathsf{c})$ with $a, b \in \mathrm{NI}_\Sigma$, $R \in \mathrm{NR}_\Sigma$, then $\mathfrak{K} \models \mathsf{c} : R(a, b)$;
- $\mathtt{eq}(a, b, \mathsf{c})$ with $a, b \in \mathrm{NI}_\Sigma$, then $\mathfrak{K} \models \mathsf{c} : a = b$;
- $\mathtt{neq}(a, b, \mathsf{c})$ with $a, b \in \mathrm{NI}_\Sigma$, then $\mathfrak{K} \models \mathsf{c} : a \neq b$;

We claim that, for any ground atom H of the above form with the corresponding semantic condition $C(H)$, $PK(\mathfrak{K}) \models H$ implies $\mathfrak{K} \models \mathsf{c} : C(H)$. We show the claim by induction on the possible proof tree of the above atoms H: the cases for the rules in P_{rl} are analogous to what has been shown in the previous lemma, thus we only have to prove the assertion for the rule in P_{loc}.

- **(plc-eq):** then $H = \mathtt{eq}(a, b, \mathsf{c})$, $PK(\mathfrak{K}) \models \mathtt{nom}(a, \mathsf{c})$ and $PK(\mathfrak{K}) \models \mathtt{eq}(a, b, \mathsf{c}')$. By induction hypothesis, by rules in I_{rl} we have $a \in \mathrm{NI}_\Sigma$ and $\mathfrak{K} \models \mathsf{c}' : (a = b)$. Then, for every model $\mathfrak{I} = \langle \mathcal{M}, \mathcal{I} \rangle$ of $\mathfrak{K}$, we have that $a^{\mathcal{I}(\mathsf{c}'^\mathcal{M})} = b^{\mathcal{I}(\mathsf{c}'^\mathcal{M})}$. By the condition on local interpretation of individuals in the definition of KG-OLAP cube model, we have that $a^{\mathcal{I}(\mathsf{c}^\mathcal{M})} = a^{\mathcal{I}(\mathsf{c}'^\mathcal{M})} = b^{\mathcal{I}(\mathsf{c}'^\mathcal{M})} = b^{\mathcal{I}(\mathsf{c}^\mathcal{M})}$. Thus it holds that $\mathcal{I}(\mathsf{c}^\mathcal{M}) \models (a = b)$ which means $\mathfrak{K} \models \mathsf{c} : (a = b)$.

□

3.4.2 Completeness

Lemma 2. *Let $\mathfrak{K} = \langle \mathfrak{G}, \mathrm{K}_{\mathsf{M}} \rangle$ be a $\mathcal{SROIQ}$-RL cube in normal form and $PK(\mathfrak{K})$ its associated program. We define the equivalence relation $\approx$ on the Herbrand universe of $PK(\mathfrak{K})$ as the reflexive, symmetric and transitive closure of*

$$\{\langle a, b \rangle \mid PK(\mathfrak{K}) \models \mathtt{eq}(a, b, c),\ \textit{for}\ a, b, c \in \mathrm{NI}_\Gamma \cup \mathrm{NI}_\Sigma\}$$

Given $a, b, c, d \in \mathrm{NI}_\Gamma \cup \mathrm{NI}_\Sigma$ with $a \approx b$, it holds that:

(i) if $PK(\mathfrak{K}) \models \mathtt{inst}(a, A, c)$, then $PK(\mathfrak{K}) \models \mathtt{inst}(b, A, c)$;

(ii) if $PK(\mathfrak{K}) \models \mathtt{triple}(a, R, d, c)$, then $PK(\mathfrak{K}) \models \mathtt{triple}(b, R, d, c)$;

(iii) if $PK(\mathfrak{K}) \models \mathtt{triple}(d, R, a, c)$, then $PK(\mathfrak{K}) \models \mathtt{triple}(d, R, b, c)$;

(iv) if $PK(\mathfrak{K}) \models \texttt{inst}(d, A, a)$, then $PK(\mathfrak{K}) \models \texttt{inst}(d, A, b)$;

(v) if $PK(\mathfrak{K}) \models \texttt{triple}(c, R, d, a)$, then $PK(\mathfrak{K}) \models \texttt{triple}(c, R, d, b)$;

Proof. By rules (prl-eq2) and (prl-eq3), it follows immediately that if $PK(\mathfrak{K}) \models \mathtt{eq}(a, b, c)$ then $PK(\mathfrak{K}) \models \texttt{inst}(a, A, c)$ iff $PK(\mathfrak{K}) \models \texttt{inst}(b, A, c)$. This also proves point (i) of the assertion. By rule (prl-eq2) and (prl-eq4), we can derive that if $PK(\mathfrak{K}) \models \mathtt{eq}(a, b, c)$, then $PK(\mathfrak{K}) \models \texttt{triple}(a, R, d, c)$ iff $PK(\mathfrak{K}) \models \texttt{triple}(b, R, d, c)$, proving point (ii). Point (iii) can be proved similarly by rules (prl-eq2) and (prl-eq5).

For point (iv), let us assume that $PK(\mathfrak{K}) \models \texttt{inst}(d, A, a)$ with $a \neq c$ (otherwise the assertion is immediate). Then, by the definition of the program, it must be that $PG''(\mathfrak{G}) \models \mathtt{eq}(a, b, \mathsf{gm})$. By rules (prl-eq2)-(prl-eq5), in particular this implies that $PG''(\mathfrak{G}) \models \texttt{triple}(a, \mathsf{mod}, \mathsf{m}, \mathsf{gm})$ iff $PG''(\mathfrak{G}) \models \texttt{triple}(b, \mathsf{mod}, \mathsf{m}, \mathsf{gm})$, meaning that, by definition of the translation, they have analogous local programs $PC(a)$ and $PC(b)$ (in which only the "context argument" in the atoms translated by the input translations changes). Thus, we obtain that $PK(\mathfrak{K}) \models \texttt{inst}(d, A, b)$. The proof for point (v) follows from similar reasoning. □

Lemma 3. *Given $\mathfrak{K} = \langle \mathfrak{G}, \mathrm{K}_{\mathsf{M}} \rangle$ a consistent $\mathcal{SROIQ}$-RL cube in normal form, and $\alpha \in \mathcal{L}_\Gamma$ (resp. $\alpha \in \mathcal{L}_\Sigma$) with $O(\alpha, \mathsf{g})$ defined and $\mathsf{g} = \mathsf{gm}$ (resp. $\mathsf{g} = \mathsf{gk}$). Then, $\mathfrak{G} \models \alpha$ implies $PG''(\mathfrak{G}) \models O(\alpha, \mathsf{g})$.*

Proof. Let us assume that $\alpha \in \mathcal{L}_\Gamma$ (the other case can be proved similarly). We show by contrapositive that: $PG''(\mathfrak{G}) \not\models O(\alpha, \mathsf{gm})$ implies $\mathfrak{G} \not\models \alpha$. Then there exists an Herbrand model $\mathcal{H}$ of $PG''(\mathfrak{G})$ such that $\mathcal{H} \not\models O(\alpha, \mathsf{gm})$. We show that from this model for $PG''(\mathfrak{G})$ we can build a model $\mathcal{M}$ for $\mathfrak{G}$ (meeting the conditions of KG-OLAP models on the global interpretation) such that $\mathcal{M} \not\models \alpha$, which allow us to derive that $\mathfrak{G} \not\models \alpha$.

Let us consider the equivalence relation $\approx$ as defined in Lemma 2. We define the equivalence classes $[c] = \{d \mid d \approx c\}$, that will be used to define the domain of the built interpretation.

Then, we define the interpretation $\mathcal{M} = \langle \Delta^{\mathcal{M}}, \cdot^{\mathcal{M}} \rangle$ as follows:

- $\Delta^{\mathcal{M}} = \{[c] \mid c \in \mathrm{NI}_\Gamma \cup \mathrm{NI}_\Sigma\}$;
- For each $e \in \Delta^{\mathcal{M}}$, we define the projection function $\iota(e)$ such that, if $e = [c]$, then $\iota(e) = b$ with a fixed $b \in [c]$;
- $c^{\mathcal{M}} = [c]$, for every $c \in \mathrm{NI}_\Gamma \cup \mathrm{NI}_\Sigma$;
- $A^{\mathcal{M}} = \{d \in \Delta^{\mathcal{M}} \mid \mathcal{H} \models \texttt{inst}(\iota(d), A, \mathsf{g})\}$, for every $A \in \mathrm{NC}_\Gamma \cup \mathrm{NC}_\Sigma$, with $\mathsf{g} = \mathsf{gm}$ or $\mathsf{g} = \mathsf{gk}$;
- $R^{\mathcal{M}}$ is the smallest set such that $\langle d, d' \rangle \in R^{\mathcal{M}}$ if one of the following conditions hold:
  - $\mathcal{H} \models \texttt{triple}(\iota(d), R, \iota(d'), \mathsf{g})$ with $\mathsf{g} = \mathsf{gm}$ or $\mathsf{g} = \mathsf{gk}$;
  - $S \sqsubseteq R \in \mathfrak{G}$ and $\langle d, d' \rangle \in S^{\mathcal{M}}$;
  - $S \circ T \sqsubseteq R \in \mathfrak{G}$ and $\langle d, e \rangle \in S^{\mathcal{M}}$, $\langle e, d' \rangle \in T^{\mathcal{M}}$ for $e \in \Delta^{\mathcal{M}}$;
  - $\mathrm{Inv}(R, S) \in \mathfrak{G}$ or $\mathrm{Inv}(S, R) \in \mathfrak{G}$ and $\langle d', d \rangle \in S^{\mathcal{M}}$;

Note that by Lemma 2, the definition of $\mathcal{M}$ does not depend on the choice of the $\iota([c]) \in [c]$. It is easy to see that, given $\alpha \in \mathcal{L}_\Gamma$ with $\mathcal{H} \not\models O(\alpha, \mathsf{gm})$, then $\mathcal{M} \not\models \alpha$. For example, if $\alpha = \mathsf{C}(\mathsf{a})$, then $\mathcal{H} \not\models \mathtt{inst}(\mathsf{a}, \mathsf{C}, \mathsf{gm})$ which implies by definition that $\mathcal{M} \not\models \mathsf{C}(\mathsf{a})$.

In order to show that $\mathcal{M}$ is a model for $\mathfrak{G}$, we have to prove that $\mathcal{M}$ satisfies the definition of global model from the definition of KG-OLAP model, and in particular that $\mathcal{M} \models \mathfrak{G}$. We easily prove that $\mathbf{N}^{\mathcal{M}} \subseteq \mathsf{Ctx}^{\mathcal{M}}$: by the definition of rule (igl-subctx2), for every $\mathsf{c} \in \mathbf{N}$ we have $\mathcal{H} \models \mathtt{inst}(\mathsf{c}, \mathsf{Ctx}, \mathsf{gm})$, which implies $\mathsf{c}^{\mathcal{M}} \in \mathsf{Ctx}^{\mathcal{M}}$. The condition $\mathsf{C}^{\mathcal{M}} \subseteq \mathsf{Ctx}^{\mathcal{M}}$ for every $\mathsf{C} \in \mathbf{C}$ can be shown similarly by the rule (igl-subctx1). Given $\mathsf{c}_1, \mathsf{c}_2 \in \mathbf{N}$, by the construction of $PG'(\mathfrak{G})$ we have that if $\mathcal{H} \models \mathtt{triple}(\mathsf{c}_1, \mathsf{A}, \mathsf{d}_\mathsf{A}, \mathsf{gm})$ and $\mathcal{H} \models \mathtt{triple}(\mathsf{c}_2, \mathsf{A}, \mathsf{d}_\mathsf{A}, \mathsf{gm})$ for all $\mathsf{A} \in \mathbf{D}$, then $\mathcal{M} \models \mathsf{c}_1^{\mathcal{M}} = \mathsf{c}_2^{\mathcal{M}}$. For the same step in the construction of $PG'(\mathfrak{G})$, it also holds that if $\mathsf{cn}(\mathbf{d}) = \mathsf{c} \in \mathbf{N}$, then $\mathcal{M} \models \mathsf{A}(\mathsf{c}, \mathsf{d}_\mathsf{A})$ for each dimension $\mathsf{A} \in \mathbf{D}$. Finally, by construction of $PG''(\mathfrak{G})$, we have that if $\mathsf{c}_2 \preceq \mathsf{c}_1$ then $\mathcal{H} \models \mathtt{triple}(\mathsf{c}_1, \mathtt{covers}, \mathsf{c}_2, \mathsf{gm})$; if $\mathcal{H} \models \mathtt{triple}(\mathsf{c}_1, \mathsf{mod}, \mathsf{m}, \mathsf{gm})$, by rule (pgl-prop) we can derive $\mathcal{H} \models \mathtt{triple}(\mathsf{c}_2, \mathsf{mod}, \mathsf{m}, \mathsf{gm})$ and thus $\mathcal{M} \models \mathsf{mod}(\mathsf{c}_2, \mathsf{m})$.

To prove that $\mathcal{M} \models \mathfrak{G}$, we proceed by cases and consider the form of all of the axioms $\beta \in \mathcal{L}_\Gamma$ or $\beta \in \mathcal{L}_\Sigma$ that can appear in $\mathfrak{G}$. We show only some representative cases:

– Let $\beta = A(a) \in \mathfrak{G}$, then $\mathcal{H} \models \mathtt{inst}(a, A, \mathrm{g})$[2]. This directly implies that $a^{\mathcal{M}} = [a] \in A^{\mathcal{M}}$.

– Let $\beta = R(a, b) \in \mathfrak{G}$, then $\mathcal{H} \models \mathtt{triple}(a, R, b, \mathrm{g})$. By definition, we directly have that $\langle [a], [b] \rangle \in R^{\mathcal{M}}$.

– Let $\beta = (a = b) \in \mathfrak{G}$, then $\mathcal{H} \models \mathtt{eq}(a, b, \mathrm{g})$. By the definition of $\approx$, it holds that $a \approx b$, thus $\{a, b\} \subseteq [a]$ and $a^{\mathcal{M}} = b^{\mathcal{M}} = [a]$.

– Let $\beta = A \sqsubseteq B \in \mathfrak{G}$, then $\mathcal{H} \models \mathtt{subClass}(A, B, \mathrm{g})$. If $d \in A^{\mathcal{M}}$, then by definition $\mathcal{H} \models \mathtt{inst}(\iota(d), A, \mathrm{g})$: by rule (prl-subc) we obtain that $\mathcal{H} \models \mathtt{inst}(\iota(d), B, \mathrm{g})$ and thus $d \in B^{\mathcal{M}}$.

– Let $\beta = A \sqsubseteq \neg B \in \mathfrak{G}$, then $\mathcal{H} \models \mathtt{supNot}(A, B, \mathrm{g})$. Suppose that $d \in A^{\mathcal{M}}$, then $\mathcal{H} \models \mathtt{inst}(\iota(d), A, \mathrm{g})$. Moreover, suppose that $d \in B^{\mathcal{M}}$: this implies that $\mathcal{H} \models \mathtt{inst}(\iota(d), B, \mathrm{g})$. By rule (prl-not) and Lemma 2, we would obtain that $\mathcal{H} \models \mathtt{inst}(\iota(d), \mathtt{bot}, \mathrm{g})$. This contradicts our assumptions on the consistency of $\mathfrak{K}$, thus $d \notin B^{\mathcal{M}}$ as required.

– Let $\beta = \exists R.A \sqsubseteq B \in \mathfrak{G}$, then $\mathcal{H} \models \mathtt{subEx}(R, A, B, \mathrm{g})$. Let $d \in (\exists R.A)^{\mathcal{M}}$: by definition of the semantics this means that there exists $d' \in A^{\mathcal{M}}$ such that $\langle d, d' \rangle \in R^{\mathcal{M}}$. Thus, $\mathcal{H} \models \mathtt{inst}(\iota(d'), A, \mathrm{g})$ and $\mathcal{H} \models \mathtt{triple}(\iota(d), R, \iota(d'), \mathrm{g})$. By rule (prl-subex), we obtain that $\mathcal{H} \models \mathtt{inst}(\iota(d), B, \mathrm{g})$: thus $d \in B^{\mathcal{M}}$ as required.

– Let $\beta = A \sqsubseteq \leqslant 1R.\top \in \mathfrak{G}$, then $\mathcal{H} \models \mathtt{supLeqOne}(A, R, \mathrm{g})$. Let $d \in A^{\mathcal{M}}$, then $\mathcal{H} \models \mathtt{inst}(\iota(d), A, \mathrm{g})$. Suppose that there exist $d_1, d_2 \in \Delta^{\mathcal{M}}$ such that $\langle d, d_1 \rangle \in R^{\mathcal{M}}$ and $\langle d, d_2 \rangle \in R^{\mathcal{M}}$. Thus $\mathcal{H} \models \{\mathtt{triple}(\iota(d), R, \iota(d_1), \mathrm{g}), \mathtt{triple}(\iota(d), R, \iota(d_2), \mathrm{g})\}$. By (prl-leqone) rule we obtain $\mathcal{H} \models \mathtt{eq}(\iota(d_1), \iota(d_2), \mathrm{g})$. This implies that $\iota(d_1) \approx \iota(d_2)$ and thus they are interpreted as the same domain element $d_1 = d_2$ in $\mathcal{M}$.

[2]In the proof of these cases, for simplicity of notation, we assume $\mathrm{g} = \mathsf{gm}$ or $\mathrm{g} = \mathsf{gk}$.

– The cases for $\beta = R \sqsubseteq S, R \circ S \sqsubseteq T$ and $\mathrm{Inv}(R, S)$ follow directly from the interpretation of roles in $\mathcal{M}$.

$\square$

Theorem 2 (Completeness)**.** Given $\mathfrak{K} = \langle \mathfrak{G}, \mathrm{K}_{\mathsf{M}} \rangle$ a consistent $\mathcal{SROIQ}$-RL cube in normal form, $\alpha \in \mathcal{L}_\Sigma$ and $\mathsf{c} \in \mathbf{F}$ s.t. $O(\alpha, \mathsf{c})$ is defined. Then, $\mathfrak{K} \models \mathsf{c} : \alpha$ implies $PK(\mathfrak{K}) \models O(\alpha, \mathsf{c})$.

Proof. As in the case of soundness, we prove the assertion by extending the previous construction on the global context to the whole structure of the input KG-OLAP cube.

We prove by contrapositive that $PK(\mathfrak{K}) \not\models O(\alpha, \mathsf{c})$ implies $\mathfrak{K} \not\models \mathsf{c} : \alpha$. Assuming that $PK(\mathfrak{K}) \not\models O(\alpha, \mathsf{c})$, then there exists an Herbrand model $\mathcal{H}$ of $PK(\mathfrak{K})$ such that $\mathcal{H} \not\models O(\alpha, \mathsf{c})$. As in the previous lemma, from this model for $PK(\mathfrak{K})$ we build a KG-OLAP model $\mathfrak{I} = \langle \mathcal{M}, \mathcal{I} \rangle$ for $\mathfrak{K}$ such that $\mathcal{I}(\mathsf{c}^{\mathcal{M}}) \not\models \alpha$, implying that $\mathfrak{K} \not\models \mathsf{c} : \alpha$.

We consider again the equivalence relation $\approx$ defined in Lemma 2 and the equivalence classes $[c] = \{d \mid d \approx c\}$ as from the above lemma. Then we build $\mathfrak{I} = \langle \mathcal{M}, \mathcal{I} \rangle$ as follows: the global interpretation $\mathcal{M} = \langle \Delta^{\mathcal{M}}, \cdot^{\mathcal{M}} \rangle$ is a structure defined as in Lemma 3; for each $e \in \Delta^{\mathcal{M}}$, we define again the projection function $\iota(e)$ such that, if $e = [c]$, then $\iota(e) = b$ with a fixed $b \in [c]$.

As in the case of Theorem 1, since we can show $\mathcal{M} \models \mathfrak{G}$ then: if $\mathsf{c} \in \mathbf{F}_{\mathfrak{G}}$ (that is, if $PG''(\mathfrak{G}) \models \mathtt{inst}(\mathsf{c}, \mathsf{Ctx}, \mathsf{gm})$ and $PG''(\mathfrak{G}) \not\models \mathtt{inst}(\mathsf{c}, \mathsf{Null}, \mathsf{gm})$) then $\mathsf{c}^{\mathcal{M}} \in \mathsf{Ctx}^{\mathcal{M}}$; if $\mathrm{K}_{\mathsf{m}} \in \mathrm{K}_{\mathsf{c}}$ (that is, if $PG''(\mathfrak{G}) \models \mathtt{triple}(\mathsf{c}, \mathsf{mod}, \mathsf{m}, \mathsf{gm})$) then $\langle \mathsf{c}^{\mathcal{M}}, \mathsf{m}^{\mathcal{M}} \rangle \in \mathsf{mod}^{\mathcal{M}}$. For every $c \in \mathbf{F}_{\mathfrak{G}}$, we build the local interpretation $\mathcal{I}(c) = \langle \Delta_c, \cdot^{\mathcal{I}(c)} \rangle$ as follows:

- $\Delta_c = \{[d] \mid d \in \mathrm{NI}_\Sigma\}$;
- $a^{\mathcal{I}(c)} = [a]$, for every $a \in \mathrm{NI}_\Sigma$;
- $A^{\mathcal{I}(c)} = \{d \in \Delta_c \mid \mathcal{H} \models \mathtt{inst}(\iota(d), A, c)\}$, for every $A \in \mathrm{NC}_\Sigma$;
- $R^{\mathcal{I}(c)}$ is the smallest set such that $\langle d, d' \rangle \in R^{\mathcal{I}(c)}$ if one of the following conditions hold:
  - $\mathcal{H} \models \mathtt{triple}(\iota(d), R, \iota(d'), c)$;
  - $S \sqsubseteq R \in \mathrm{K}_c \cup \mathfrak{G}_\Sigma$ and $\langle d, d' \rangle \in S^{\mathcal{I}(c)}$;
  - $S \circ T \sqsubseteq R \in \mathrm{K}_c \cup \mathfrak{G}_\Sigma$ and $\langle d, e \rangle \in S^{\mathcal{I}(c)}$, $\langle e, d' \rangle \in T^{\mathcal{I}(c)}$ for $e \in \Delta_c$;
  - $\mathrm{Inv}(R, S) \in \mathrm{K}_c \cup \mathfrak{G}_\Sigma$ or $\mathrm{Inv}(S, R) \in \mathrm{K}_c \cup \mathfrak{G}_\Sigma$ and $\langle d', d \rangle \in S^{\mathcal{I}(c)}$;

As in the above lemma, we can see that, given $\mathcal{H} \not\models O(\alpha, \mathsf{c})$, then $\mathcal{I}(\mathsf{c}^{\mathcal{M}}) \not\models \alpha$ as required.

To show the assertion, we have to prove that $\mathfrak{I}$ meets the definition of KG-OLAP cube model and that $\mathfrak{I} \models \mathfrak{K}$. By Lemma 3 we directly obtain that the conditions on the global interpretation $\mathcal{M}$ are verified. Given $x, y \in \mathsf{Ctx}^{\mathcal{M}}$, we note also that, by the definition of $\iota(e)$, for every $a \in \mathrm{NI}_\Sigma$ it holds that $a^{\mathcal{I}(x)} = a^{\mathcal{I}(y)} = a^{\mathcal{M}} = [a]$.

To complete the proof, we have to show that for every K_{m} s.t. $\langle c, \mathsf{m}^{\mathcal{M}} \rangle \in \mathsf{mod}^{\mathcal{M}}$ (that is, every $\mathrm{K}_{\mathsf{m}} \in \mathrm{K}_c$) we have $\mathcal{I}(c) \models \mathrm{K}_{\mathsf{m}}$ and $\mathcal{I}(c) \models \mathfrak{G}_\Sigma$. Since axioms in each K_{m} and $\mathfrak{G}_\Sigma$ are in the general normal form of Table 3, this can be shown analogously as in the case of Lemma 3, by proceeding by cases and considering the form of all of the axioms $\beta \in \mathcal{L}_\Sigma$ that can appear in $\mathrm{K}_c \cup \mathfrak{G}_\Sigma$. $\square$

4 KG-OLAP system

We provide a SPARQL-based proof-of-concept implementation of a KG-OLAP system in Java. We employ an off-the-shelf quad store for storing the contents of the KG-OLAP cube. We employ the RDFpro framework to materialize inferences within KG-OLAP cubes. In this section, we first describe the architecture of the KG-OLAP system. We then introduce a DL vocabulary for defining multidimensional KG-OLAP models and explain the RDFpro ruleset. Finally, we present interface and SPARQL implementation of the query operators. The source code is available online[3], along with the scripts and full logs of the performance experiments.

4.1 Architecture

The KG-OLAP system consists of two repositories, namely base and temporary repository (Fig. 1). The base repository contains the base data which are periodically updated by extract, transform, and load (ETL) routines, which are outside the scope of this paper; we refer to data warehousing literature [8] for more information on ETL. The base repository contains both schema and instance data of a KG-OLAP cube. The temporary repository contains a working copy of (selected partitions of) the KG-OLAP cube from the base repository. Using the slice-and-dice operator, an analyst selects a subset of the data from the base repository to be loaded into the temporary repository for further analysis. Merge, abstract, reification, and pivoting operations are then performed on the temporary repository.

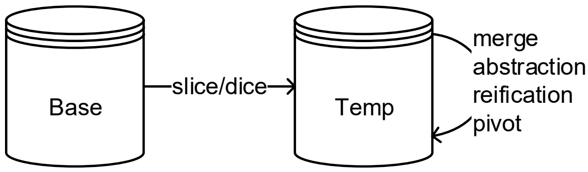

Figure 1: The architecture of the proof-of-concept KG-OLAP system

The RDFpro[4] rule inference engine computes context coverage as well as the materialization of inferences and propagation of knowledge across contexts. The current implementation, for evaluation purposes, loads the repository into an RDF model in main memory, performs inferences, and writes the model back into the repository. RDFpro, however, also supports stream-based computation using the Sesame/RDF4J `RDFHandler` interface.

Due to the SPARQL-based implementation of query operations, off-the-shelf RDF quad stores may manage base and temporary repositories of a KG-OLAP system. In theory, any RDF quad store can be used; the current implementation has been tested using Ontotext GraphDB[5]. Performance optimization was not a concern of this study and is left to future work. In particular, due to the modularized nature of KG-OLAP cubes, we expect a parallelized and distributed implementation on multiple server nodes to benefit performance.

[3]`http://kg-olap.dke.uni-linz.ac.at/`
[4]`http://rdfpro.fbk.eu/`
[5]`http://graphdb.ontotext.com/`

$\exists$atLevel.$\top \sqsubseteq$ DimensionMember	(1)
$\top \sqsubseteq \forall$atLevel.Level	(2)
$Func$(atLevel)	(3)
DimensionMember $\sqcap$ Level $\sqsubseteq \bot$	(4)
$Trans$(rollsUpTo), $Asym$(rollsUpTo)	(5)
directlyRollsUpTo $\sqsubseteq$ rollsUpTo	(6)
DimensionMember $\sqsubseteq \exists$rollsUpTo.Self	(7)
Level $\sqsubseteq \exists$rollsUpTo.Self	(8)
DimensionMember $\sqsubseteq \forall$rollsUpTo.DimensionMember	(9)
Level $\sqsubseteq \forall$rollsUpTo.Level	(10)
$\exists$hasDimensionAttributeValue.$\top \sqsubseteq$	(11)
$\top \sqsubseteq \forall$hasDimensionAttributeValue.DimensionMember	(12)
$L_A \sqcap L_B \sqsubseteq \bot, A \neq B$	(13)
$D_A \sqcap D_B \sqsubseteq \bot, A \neq B$	(14)
$d_A, d_B \sqsubseteq$ hasDimensionAttributeValue	(15)
$Func(d_A), Func(d_B)$	(16)
$\top \sqsubseteq \forall d_A.D_A, \forall d_B.D_B$	(17)
Cell $\sqsubseteq$ Context	(18)
$\exists$hasDimensionAttributeValue.$\top \sqsubseteq$ Cell	(19)
$\top \sqsubseteq \forall$hasDimensionAttributeValue.DimensionAttributeValue	(20)
$\exists$covers.$\top \sqsubseteq$ Cell	(21)
$\top \sqsubseteq \forall$covers.Cell	(22)
$\exists$hasAssertedModule.$\top \sqsubseteq$ Context	(23)
$\top \sqsubseteq \forall$hasAssertedModule.Module	(24)
hasAssertedModule $\sqsubseteq$ hasModule	(25)
Context $\sqcap$ Module $\sqsubseteq \bot$	(26)

Figure 2: A DL representation of the KG-OLAP cube modeling language

4.2 Multidimensional Model

Figure 2 shows an intuitive representation in DL notation of the axioms that define the elements of $\mathcal{L}_\Omega$ – the cube knowledge (see Sect. 3.2 in the main paper). The Cell class corresponds to the set **F** in the formal definition of the cube vocabulary Ω, the Level class corresponds to the set **L**, and the DimensionAttributeValue class corresponds to the set **I** of dimension members; the subroles of hasDimensionAttributeValue (Line 15) constitute the set **D**.

The upper part of Fig. 2 (Lines 1-12) shows roles for the definition of the ordering of dimension members and levels as well as roles for the association of dimension members with levels and cells with dimension members. The atLevel role associates a dimension member with a level (Lines 1-2); the atLevel role is functional (Line 3). The DimensionMember and Level concepts are mutually disjoint (Line 4). The transitive and asymmetric rollsUpTo role (Line 5) establishes the hierarchical order of dimension members and levels, respectively. The directlyRollsUpTo role is a sub-property of rollsUpTo (Line 6) used to assert direct roll-up relationships between levels and between dimension members, respectively. The rollsUpTo relationships derive from the explicitly defined directlyRollsUpTo relationships. Each dimension member rolls up to itself, so does each level (Lines 7-8). Dimension members only roll up to other dimension members, levels only to other levels (Lines 9-10). The hasDimensionAttributeValue role associates a cell with a dimension member (Lines 11-12). Intuitively, the set of an individual cell's hasDimensionAttributeValue property values defines the position of the cell within the OLAP cube.

The middle part of Fig. 2 (Lines 13-17) generalizes the definition of dimension-specific concepts and roles. Let A, B denote two different dimensions. The disjoint concepts L_A and L_B (Line 13) represent the sets of levels of the dimensions A and B. The disjoint concepts D_A and D_B (Line 14) represent the sets of members of the dimensions A and B. The roles d_A and d_B represent the actual dimensions and are subsets of the hasDimensionAttributeValue role (Line 15); the roles d_A and d_B are functional (Line 16). The roles d_A and d_B associate a cell with a member of the dimensions A and B (Line 17), respectively.

The lower part of Fig. 2 (Lines 18-26) relates to cells, coverage relationships between cells, and knowledge modules. The Cell class is a subclass of the CKR core concept Context (Line 18). The hasDimensionAttributeValue role links a cell with a dimension attribute value (Lines 19-20). The covers role relates cells based on the granularity level; the cell at the more general granularity covers the cell at the more specific level (Lines 21-22). A context/cell has an asserted module (Line 23-24) which contains the explicitly defined knowledge facts. Each context/cell also has a module with inferred knowledge generated by the RDFpro rules, which is linked to the cell via the hasModule role. The hasAssertedModule rule is a sub property of hasModule (Line 25). Context and Module are disjoint classes (Line 26).

4.3 RDFpro ruleset

We investigate complexity of reasoning using two separate RDFpro rulesets. The first ruleset considers subclass relationships between contexts in order to determine class membership. The second ruleset considers subclass relationships as well as domain/range constraints. We stress, though, that the RDFpro ruleset could be extended in order to support $\mathcal{SROIQ}$-RL reasoning. We show an example for the evaluation of more complex rules that evaluate conditions over literal values, thereby allowing for the selective inclusion of more complex rules when needed, without the overhead introduced by full $\mathcal{SROIQ}$-RL support.

The ruleset application strategy follows these phases:

Phase 1: Global inference graph. The global inference graph holds the inference closure of the global context. Most notably, the transitive and reflexive closure of the coverage relationships will be associated with this context.

Phase 2: Global closure. In the global context, (restricted) RDFS and OWL reasoning is performed; the closure is stored in the global inference graph. In particular, global closure includes derivations from class-membership reasoning and transitive properties.

Phase 3: Context coverage. For defined contexts, a rule derives coverage relationships based on the context coordinates. Listing 1 shows the rule for deriving coverage in a three-dimensional cube in the ATM scenario with location, time, and aircraft dimensions. In this respect, the ruleset is cube-specific, but this customized part of the ruleset could be automatically generated from the cube definition.

Listing 1: Rule for deriving coverage relationships in a three-dimensional ATM cube

```
:cts-compute-covers a rr:NonFixpointRule ;
rr:phase "3" ;
rr:head """ GRAPH ?global_inf { ?c2 olap:covers ?c1 } """ ;
rr:body """ GRAPH ckr:global {
              ?c1 cube:hasAircraft ?air1 .
              ?c1 cube:hasLocation ?loc1 .
              ?c1 cube:hasDate ?date1.
              ?c2 cube:hasAircraft ?air .
              ?c2 cube:hasLocation ?loc .
              ?c2 cube:hasDate ?date .
            }
            GRAPH ?global_inf {
              ?air1 olap:rollsUpTo ?air .
              ?loc1 olap:rollsUpTo ?loc .
              ?date1 olap:rollsUpTo ?date .
            } """ .
```

Phase 4: Global closure (recomputation). The derivation of coverage relationships necessitates a recomputation of the global closure in order to correctly update the transitive and reflexive closure of the coverage relationships.

Phase 5: Local dependencies. For each context, an inference model is created that holds the closure of the context's module. Downward propagation of modules is also computed here.

Phase 6: Local closures. An inference model's content consists of the knowledge inferred by application of reasoning rules over the knowledge module of the respective context.

Phase 7: Value rules. Custom rules allow for the definition of more complex business terms, e.g., OWL derived concepts such as HeavyWeight aircraft characteristic (Listing 2), defined as an aircraft characteristic referring to aircraft with a weight above 136. OWL reasoning may also be enabled, but the explicit definition of rules avoids overhead introduced by OWL reasoning.

Listing 2: Rule for membership reasoning for a derived concept

```
:prl-min-hfl a rr:NonFixpointRule ;
rr:phase "7" ;
rr:head """ GRAPH ?g_inf { ?x a ?y } """ ;
rr:body """ GRAPH ?g1 {
              ?x obj:weight ?v
            }
            GRAPH ?g2 {
              ?x obj:weightInterpretation "above"
            }
            GRAPH ?g3 {
              ?y owl:DataHasValue ?minV
            }
            GRAPH ?g4 {
```

```
          ?y owl:onProperty obj:minWeight
        }
        GRAPH ?g5 {
          ?y rdfs:subClassOf obj:HeavyWeight
        }
        FILTER (xsd:double(?minV) <= xsd:double(?v))
        GRAPH ?g_inf {
          ?g_inf ckr:derivedFrom ?g1, ?g2, ?g3, ?g4, ?g5
        } """ .
```

4.4 Statement classes

The `at.jku.dke.kgolap.operators` package contains implementations of the KG-OLAP query operators. The implementation introduces additional parameters beyond the more general formalization. In particular, selection conditions allow for a targeted application of the query operations to groups of cells and triples within cells.

The abstract `Statement` class is the root class of all statements; the KG-OLAP query operators are implemented as extensions of `Statement`, which are instantiated in order to perform an operation. Abstract method `prepareStatement` is implemented by the subclasses of `Statement` to return the SPARQL `SELECT` statement corresponding to the specific operation. The SPARQL `SELECT` statements returns a "delta" table, i.e., a set of quads along with an indication whether the quad is to be inserted or deleted. The `prepareUpdate` method transforms the `SELECT` statement returned by `prepareStatement` into an `INSERT/DELETE` statement. The `Statement` class implements the public methods `execute`, `executeInMemory` and `executeUpdate` with no parameters, which represent different ways of performing the operation. The `execute` method executes the SPARQL `SELECT` statement returned by `prepareStatement`, with the delta table being written to disk. The `executeInMemory` method executes the `SELECT` statement and keeps the delta table in memory. Both `execute` and `executeInMemory` perform bulk deletions and insertions of the statements in the delta table. The `executeUpdate` method, on the other hand, performs a SPARQL `INSERT/DELETE` statement derived from the SPARQL `SELECT` delta query.

The `Statement` class works over `Repo` instances, which specify the KG-OLAP repositories that the query operations apply to. The `Statement` class has members for source and target repository but only for the slice/dice implementation there is a distinction between the two. The abstract `Repo` class defines methods for accessing and updating a KG-OLAP repository, providing an additional layer of abstraction from the actual framework that is used for connecting to the repository. The abstract `SesameRepo` class is an implementation of `Repo` using the Sesame/RDF4J framework for connecting to the repository in the quad store, with `SesameHTTPRepo` and `SesameSailRepo` being concrete implementations.

4.4.1 SliceDice

The `SliceDice` class extends `Statement` and implements the *slice-and-dice* operator. The constructor takes values for the private member variables `sourceRepository` and `targetRepository` (`Repo` objects) as well as `prefixes`, a string containing prefix definitions for the SPARQL query. The source repository of the operation – typically the base repository of the KG-OLAP cube – is where the data are selected from; the source repository remains unaffected by the operation. The target repository –

typically the temporary repository – is where the selected data are inserted into. The `addDiceCoordinates` method, which takes a dimensional vector as parameter, i.e., a map from dimensions to dimension members, sets the selection criteria. To be precise, the `addDiceCoordinates` method takes the string representations of the IRIs of coordinate dimensions and dimension members, and the same applies for analogous cases in other operators. Internally, the private `prepareStatement` method prepares a SPARQL query taking the argument values into account and returns the query string to the `execute` methods which invoke the corresponding `Repo` methods. Listing 3 shows an instantiation and execution of `SliceDice` that selects cells concerning the LOVV region relevant for all aircraft types and dates.

Listing 3: Initialization and execution of a slice-and-dice operation

```
SliceDice sliceDice = new SliceDice(
  baseRepository, tempRepository, prefixes
);

Map<String, String> coordinates = new HashMap<String,String>();

coordinates.put(
  "cube:hasAircraft", "cube:Level_Aircraft_All-All"
);

coordinates.put(
  "cube:hasLocation", "cube:Level_Location_Region-LOVV"
);

coordinates.put("cube:hasDate", "cube:Level_Date_All-All");

coordinates.put(
  "cube:hasImportance", "cube:Level_Importance_All-All"
);

sliceDice.addDiceCoordinates(diceCoordinates);

sliceDice.execute();
```

4.4.2 Merge

The `Merge` class extends `Statement` and implements the *merge* operator. The constructor takes an object representing the repository that the operation works on (a `Repo` object) – typically the temporary repository – as well as a value for `prefixes` (a string). The inherited members `sourceRepository` and `targetRepository` reference the same object. The `setMethod` method allows to choose a value – either `UNION` or `INTERSECT` – from the public `Method` enumeration in order to determine the merge method; the proof-of-concept prototype implements only the union variant. The `setGranularity` method, which takes a dimension and a level as parameters, sets the aggregation level in a particular dimension. Listing 4 shows an instantiation and execution of the `Merge` class, performing a merge union in order to roll up cells to the aircraft type, region, all-date, and all-importance levels.

Listing 4: Initialization and execution of a merge-union operation

```
Merge merge = new Merge(tempRepository, prefixes);

merge.setMethod(Merge.Method.UNION);

merge.setGranularity(
  "cube:hasAircraft", "cube:Level_Aircraft_Type"
);

merge.setGranularity(
  "cube:hasLocation", "cube:Level_Location_Region"
);

merge.setGranularity("cube:hasDate", "cube:Level_Date_All");

merge.setGranularity(
  "cube:hasImportance", "cube:Level_Importance_All"
);

merge.execute();
```

4.4.3 ReplaceByGrouping (triple-generating abstraction)

The `ReplaceByGrouping` class extends `Statement` and implements triple-generating abstraction. Two variants exist, each characterized by a separate constructor: The first takes a context as argument, the other a granularity level. Both constructors take an object representing the repository that the operation works on (a `Repo` object) – typically the temporary repository – as well as a value for `prefixes` (a string). The inherited members `sourceRepository` and `targetRepository` reference the same object. The `setGroupingProperty` method, which takes a property as parameter, sets the property that determines the grouping for the abstraction. The `setGroupingResource` method, which takes a resource as parameter, allows for a restriction of the triples affected by abstraction: Only subjects of the specified type should be replaced by the grouping. For example, in Listing 5, the abstraction replaces resources of type ManoeuvringAreaUsage with their usageType property in all cells at aircraft model, location segment, day, and importance granularity.

Listing 5: Initialization and execution of a triple-generating abstraction operation

```
Map<String, String> granularity = new HashMap<String,String>();

granularity.put(
  "cube:hasAircraft", "cube:Level_Aircraft_Model"
);

granularity.put(
  "cube:hasLocation", "cube:Level_Location_Segment"
);

granularity.put("cube:hasDate", "cube:Level_Date_Day");

granularity.put(
```

```
  "cube:hasImportance", "cube:Level_Importance_Importance"
);

ReplaceByGrouping abstraction = new ReplaceByGrouping(
  tempRepository, prefixes, granularity
);

abstraction.setGroupingProperty("obj:usageType");
abstraction.setGroupingResource("obj:ManoeuvringAreaUsage");

abstraction.execute();
```

4.4.4 GroupByProperties (individual-generating abstraction)

The `GroupByProperties` class extends `Statement` and implements individual-generating abstraction. Two variants exist, each characterized by a separate constructor: The first takes a context as argument, the other a granularity level. Both constructors take an object representing the repository that the operation works on (a `Repo` object) – typically the temporary repository – as well as a value for `prefixes` (a string). The `addGroupingProperty` method adds a property which is used for grouping the resources. As an extension of the formalization of the basic idea, multiple grouping properties are allowed. All resources with the same grouping property values are grouped together in the context's RDF triples. As opposed to the formalization, the implementation supports specification of several group-by properties. The `setGeneratedGrouping` method specifies the property that is generated in order to keep track which resources are grouped together into which grouping. The `setGroupedResourceClass` method specifies the class to restrict grouped individuals, i.e., only instances of that particular class are grouped. In case the property is `null`, no restriction applies. For example, in Listing 6, with `obj:operationalStatus` as grouping property as well as `obj:grouping` as property for the generated grouping, all individuals with the same operational status are replaced by a new individual in statements, and the original resources receive a `obj:grouping` property to the generated individual.

Listing 6: Initialization and execution of an individual-generating operation

```
Map<String, String> granularity = new HashMap<String,String>();

granularity.put(
  "cube:hasAircraft", "cube:Level_Aircraft_Model"
);

granularity.put(
  "cube:hasLocation", "cube:Level_Location_Segment"
);

granularity.put("cube:hasDate", "cube:Level_Date_Day");

granularity.put(
  "cube:hasImportance", "cube:Level_Importance_Importance"
);

```

```
GroupByProperties abstraction = new GroupByProperties(
  tempRepository, prefixes, granularity
);

abstraction.addGroupingProperty("obj:operationalStatus");
abstraction.setGeneratedGrouping("obj:grouping");
abstraction.setGroupedResourceClass(null);

abstraction.execute();
```

4.4.5 AggregatePropertyValues (value-generating abstraction)

The `AggregatePropertyValues` class extends `Statement` and implements value-generating abstraction. Two variants exist, each characterized by a separate constructor: The first takes a context as argument, the other a granularity level. Both constructors take an object representing the repository that the operation works on (a `Repo` object) – typically the temporary repository – as well as a value for `prefixes` (a string). The `setAggregateFunction` method allows to choose a value – either `SUM`, `MIN`, `MAX`, `AVG`, and `COUNT` – from the public `AggregateFunction` enumeration in order to determine the aggregation function. The `setAggregatedProperty` method specifies the property the values of which are aggregated for the same individual. The `setGroupedResourceClass` method specifies the class used for selection of the individuals the aggregated property of which is aggregated. For example, in Listing 7, the abstraction operation computes the average wingspan of each AircraftCharacteristic individual, assuming a dataset where AircraftCharacteristic individuals with multiple wingspan properties exist.

Listing 7: Initialization and execution of a merge operation

```
Map<String, String> granularity = new HashMap<String,String>();

granularity.put(
  "cube:hasAircraft", "cube:Level_Aircraft_Model"
);

granularity.put(
  "cube:hasLocation", "cube:Level_Location_Segment"
);

granularity.put("cube:hasDate", "cube:Level_Date_Day");

granularity.put(
  "cube:hasImportance", "cube:Level_Importance_Importance"
);

AggregatePropertyValues abstract = new AggregatePropertyValues(
  tempRepository, prefixes, granularity
);

abstract.setAggregatedProperty("obj:wingspan");

abstract.setAggregateFunction(
```

```
  AggregateLiterals.AggregateFunction.AVG
);

abstract.setGroupedResourceClass("obj:AircraftCharacteristic");

abstract.execute();
```

4.4.6 Reification

The `Reification` class extends `Statement` and implements the reification operation. Two variants exist, each characterized by a separate constructor: The first takes a context as argument, the other a granularity level. Both constructors take an object representing the repository that the operation works on (a `Repo` object) – typically the temporary repository – as well as a value for `prefixes` (a string). The `setReificationPredicate` method, which takes the IRI of a property as parameter, defines the property that the reified statements must have as predicate. For example, in Listing 8, the reification operation reifies triples where `obj:usage` is the predicate.

Listing 8: Initialization and execution of a reification operation

```
Map<String, String> granularity = new HashMap<String,String>();

granularity.put(
  "cube:hasAircraft", "cube:Level_Aircraft_Model"
);

granularity.put(
  "cube:hasLocation", "cube:Level_Location_Segment"
);

granularity.put("cube:hasDate", "cube:Level_Date_Day");

granularity.put(
  "cube:hasImportance", "cube:Level_Importance_Importance"
);


Reification reification =
  new Reification(granularity, tempRepository, prefixes);

reification.setReificationPredicate("obj:usage");

reification.execute();
```

4.4.7 Pivot

The `Pivot` class extends `Statement` and implements the *pivoting* operator. Two variants exist, each characterized by a separate constructor: The first takes a context as argument, the other a granularity level. The `setDimensionProperty` method, which takes a property as parameter, determines the property from the dimensional meta-knowledge that should be included in the instance data. The `setPivotProperty` method, which takes a property as parameter, determines which property should be used in the instance data for the previously defined dimension property. The `setSelectionCondition` method, which takes a property and a resource as parameters, determines which individuals in the affected cells are to receive the specified pivot property based on the specified dimension property value of the respective cell. For example, in Listing 9, the pivoting operation applies to the cells at aircraft model, location segment, and day granularity. In these cells, individuals of type ManoeuvringAreaAvailability have the respective cell's `cube:hasLocation` property attached via the `obj:hasLocation` property.

Listing 9: Initialization and execution of a pivoting operation

```
Map<String, String> granularity = new HashMap<String,String>();

granularity.put(
  "cube:hasAircraft", "cube:Level_Aircraft_Model"
);

granularity.put(
  "cube:hasLocation", "cube:Level_Location_Segment"
);

granularity.put("cube:hasDate", "cube:Level_Date_Day");

granularity.put(
  "cube:hasImportance", "cube:Level_Importance_Importance"
);

Pivot pivot = new Pivot(granularity, tempRepository, prefixes);

pivot.setDimensionProperty("cube:hasLocation");

pivot.setSelectionCondition(
  "rdf:type", "obj:ManoeuvringAreaAvailability"
);

pivot.setPivotProperty("obj:hasLocation");

pivot.execute();
```

4.5 SPARQL queries

In the following, we illustrate the SPARQL realization of KG-OLAP operations using the examples from the previous sections, which we also use in the performance experiments. The `execute` and `executeInMemory` methods of the `Statement` class executes dynamically-generated SPARQL code depending on the specific subclass and the respective object's member variables that represent the query arguments. The operations are realized as SPARQL `SELECT` statements that return a "delta" table, i.e., a tuple query result where each tuple represents an RDF quad along with the indication of the operation (-, +), which specifies whether the quad must be added to or deleted from the target repository in order to obtain the result. The quad store takes care of optimization. Table 5 shows a delta table extract that deletes and inserts a tuple, which in that case corresponds to a triple-generating abstraction that replaces runway individuals by the airport individual that the runway is situated at. Statements can later be deleted and inserted according to the delta table. The delta queries can also easily be translated into SPARQL `DELETE`/`INSERT` statements, which is the purpose of the `prepareUpdate` method of the `Statement` class.

Table 5: An example delta table

?s	?p	?o	?g	?op
`obj:Runway16/34`	`obj:contaminant`	`obj:cont#265`	`cube:Ctx-1-mod`	"-"
`obj:airportLOWW`	`obj:contaminant`	`obj:cont#265`	`cube:Ctx-1-mod`	"+"

4.5.1 Slice and dice

A slice-and-dice operation translates into a `SELECT` statement executed on the source repository, which is typically the base repository. The resulting delta table, which contains only insertions, is then applied to the target repository. The `INSERT/DELETE` variant of the slice-and-dice realization employs the `SERVICE` clause. The example in Listing 10 shows the `SELECT` statement for a slice-and-dice operation that selects cells relevant to the LOVV region.

Listing 10 shows an example slice-and-dice operation that selects cells relevant for the LOVV region. The `WHERE` clause consists of several sub-`SELECT` statements conjoined by `UNION`, with each sub-`SELECT` accounting for different aspects of the KG-OLAP cube that need to be selected for insertion into the target repository. Each sub-`SELECT` statement returns tuples that represent RDF quads, using the bindings `?g`, `?s`, `?p`, and `?o` for graph name, subject, predicate, and object, respectively. The union of result tuples of the sub-`SELECT` statements is extended with an `?op` binding that is assigned the "+" literal, meaning the result quads are inserted into the target repository.

The first sub-`SELECT` (Listing 10, Lines 15-49) and the second sub-`SELECT` (Lines 51-84) select knowledge about contexts/cells – referred to by the `?ctx` binding – that are relevant according to the dice coordinates, including all contexts that have coordinates that roll up to the dice coordinates or that the dice coordinates roll up to, i.e., cells the location dimension attribute value of which is in a roll-up relationship with LOVV and the other dimension attribute values roll up to the respective dimension's all attribute value (Lines 27-48). The first sub-`SELECT` selects all triples where `?ctx` is the subject, the second sub-`SELECT` selects all triples where `?ctx` is the object. The cube knowledge of interest is in the `ckr:global` and `<ckr:global-inf>` graphs, which means that the triples are selected from these graphs (Line 19). The query selects triples where the contexts of interest are subject and the `covers` property is not predicate (Line 18). The inclusion of coverage statements in the first sub-`SELECT` would result in the selection of knowledge about coverage relationships of irrelevant contexts since a relevant context may cover an irrelevant context. The coverage statements, however, are included in the second sub-`SELECT`.

The third sub-`SELECT` (Listing 10, Lines 86-130) selects knowledge about the knowledge modules – referred to by the `?m` binding – that are associated with relevant contexts/cells. Relevant contexts are selected using the same condition as the first two sub-`SELECT` statements. The `?m` module must be (inferred or asserted) module of a relevant context (Lines 127-129). The third sub-`SELECT` statement selects triples where the modules are subject or object (Lines 87-97).

The fourth sub-`SELECT` (Listing 10, Lines 132-167) selects data from the (asserted or inferred) modules – bound to the `?g` variable – that are associated with relevant contexts/cells. Relevant contexts are selected using the same condition as the first three sub-`SELECT` statements. The `?g` module must be (inferred or asserted) module of a relevant context (Lines 164-166).

The fifth sub-`SELECT` (Listing 10, Lines 169-193) selects the elements of the dimensional model that apply to the relevant contexts/cells. The dimensional model elements are the dimension members, and the levels these members belong to, that roll up to the dice-coordinates or which the dice-coordinates roll up to (Lines 181-192).

The remaining sub-`SELECT` statements (Listing 10, Lines 195-242) select knowledge through inline definitions. This knowledge consists of OWL and RDFS definitions as well as the relationships between the global context and the global inference context.

Listing 10: Example SPARQL slice-and-dice operation

```
1  PREFIX owl: <http://www.w3.org/2002/07/owl#>
2  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3  PREFIX obj: <http://example.org/kgolap/object-model#>
4  PREFIX xml: <http://www.w3.org/XML/1998/namespace>
5  PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
6  PREFIX olap: <http://dkm.fbk.eu/ckr/olap-model#>
7  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
8  PREFIX cube: <http://example.org/kgolap/cube-model#>
9  PREFIX ckr: <http://dkm.fbk.eu/ckr/meta#>
10 PREFIX onto: <http://www.ontotext.com/>
11
12
13 SELECT DISTINCT ?s ?p ?o ?g ?op WHERE {
14   {
15     SELECT DISTINCT ?g ?s ?p ?o WHERE {
16       GRAPH ?g {
17         ?ctx ?p ?o .
18         FILTER(?p != olap:covers)
19       } VALUES ?g {ckr:global <ckr:global-inf>}
20       BIND(?ctx AS ?s)
21       GRAPH ckr:global {
22         ?ctx cube:hasAircraft ?d1 .
23         ?ctx cube:hasLocation ?d2 .
24         ?ctx cube:hasDate ?d3 .
25         ?ctx cube:hasImportance ?d4 .
26       }
27       GRAPH <ckr:global-inf> {{
28         {
29           ?d1 olap:rollsUpTo cube:Level_Aircraft_All-All .
30         } UNION {
31            cube:Level_Aircraft_All-All olap:rollsUpTo ?d1 .
32         }
33         {
34           ?d2 olap:rollsUpTo cube:Level_Location_Region-LOVV .
35         } UNION {
36           cube:Level_Location_Region-LOVV olap:rollsUpTo ?d2 .
37         }
38         {
```

```
        ?d3 olap:rollsUpTo cube:Level_Date_All-All .
      } UNION {
        cube:Level_Date_All-All olap:rollsUpTo ?d3 .
      }
      {
        ?d4 olap:rollsUpTo cube:Level_Importance_All-All .
      } UNION {
        cube:Level_Importance_All-All olap:rollsUpTo ?d4 .
      }
    }
  }
} UNION {
  SELECT DISTINCT ?g ?s ?p ?o WHERE {
    GRAPH ?g {
      ?s ?p ?ctx .
  } VALUES ?g {ckr:global <ckr:global-inf>}
    BIND(?ctx AS ?o)
    GRAPH ckr:global {
      ?ctx cube:hasAircraft ?d1 .
      ?ctx cube:hasLocation ?d2 .
      ?ctx cube:hasDate ?d3 .
      ?ctx cube:hasImportance ?d4 .
    }
    GRAPH <ckr:global-inf> {
      {
        ?d1 olap:rollsUpTo cube:Level_Aircraft_All-All .
      } UNION {
        cube:Level_Aircraft_All-All olap:rollsUpTo ?d1 .
      }
      {
        ?d2 olap:rollsUpTo cube:Level_Location_Region-LOVV .
      } UNION {
        cube:Level_Location_Region-LOVV olap:rollsUpTo ?d2 .
      }
      {
        ?d3 olap:rollsUpTo cube:Level_Date_All-All .
      } UNION {
        cube:Level_Date_All-All olap:rollsUpTo ?d3 .
      }
      {
        ?d4 olap:rollsUpTo cube:Level_Importance_All-All .
      } UNION {
        cube:Level_Importance_All-All olap:rollsUpTo ?d4 .
      }
    }
  }
} UNION {
  SELECT DISTINCT ?g ?s ?p ?o WHERE {
    {
      GRAPH ?g {
        ?m ?p ?o .
      } VALUES ?g {ckr:global <ckr:global-inf>}
      BIND(?m AS ?s)
    } UNION {
```

```
        GRAPH ?g {
          ?s ?p ?m .
        } VALUES ?g {ckr:global <ckr:global-inf>}
        BIND(?m AS ?o)
      }
      GRAPH ckr:global {
        ?ctx cube:hasAircraft ?d1 .
        ?ctx cube:hasLocation ?d2 .
        ?ctx cube:hasDate ?d3 .
        ?ctx cube:hasImportance ?d4 .
      }
      GRAPH <ckr:global-inf> {
        {
          ?d1 olap:rollsUpTo cube:Level_Aircraft_All-All .
        } UNION {
          cube:Level_Aircraft_All-All olap:rollsUpTo ?d1 .
        }
        {
          ?d2 olap:rollsUpTo cube:Level_Location_Region-LOVV .
        } UNION {
          cube:Level_Location_Region-LOVV olap:rollsUpTo ?d2 .
        }
        {
          ?d3 olap:rollsUpTo cube:Level_Date_All-All .
        } UNION {
          cube:Level_Date_All-All olap:rollsUpTo ?d3 .
        }
        {
          ?d4 olap:rollsUpTo cube:Level_Importance_All-All .
        } UNION {
          cube:Level_Importance_All-All olap:rollsUpTo ?d4 .
        }
      }
      # Take either asserted or derived modules.
      GRAPH <ckr:global-inf> {
        ?ctx ckr:hasModule ?m .
      }
    }
  } UNION {
    SELECT DISTINCT ?g ?s ?p ?o WHERE {
      GRAPH ?g {
        ?s ?p ?o .
      }
      GRAPH ckr:global {
        ?ctx cube:hasAircraft ?d1 .
        ?ctx cube:hasLocation ?d2 .
        ?ctx cube:hasDate ?d3 .
        ?ctx cube:hasImportance ?d4 .
      }
      GRAPH <ckr:global-inf> {
        {
          ?d1 olap:rollsUpTo cube:Level_Aircraft_All-All .
        } UNION {
          cube:Level_Aircraft_All-All olap:rollsUpTo ?d1 .
```

```
      }
      {
        ?d2 olap:rollsUpTo cube:Level_Location_Region-LOVV .
      } UNION {
        cube:Level_Location_Region-LOVV olap:rollsUpTo ?d2 .
      }
      {
        ?d3 olap:rollsUpTo cube:Level_Date_All-All .
      } UNION {
        cube:Level_Date_All-All olap:rollsUpTo ?d3 .
      }
      {
        ?d4 olap:rollsUpTo cube:Level_Importance_All-All .
      } UNION {
        cube:Level_Importance_All-All olap:rollsUpTo ?d4 -.
      }
    }
    GRAPH <ckr:global-inf> {
      ?ctx ckr:hasModule ?g .
    }
  }
} UNION {
  SELECT DISTINCT ?g ?s ?p ?o WHERE {
    {
      GRAPH ?g {
        ?d ?p ?o .
      } VALUES ?g {ckr:global <ckr:global-inf>}
      BIND(?d AS ?s)
    } UNION {
      GRAPH ?g {
        ?l ?p ?o .
      } VALUES ?g {ckr:global <ckr:global-inf>}
      BIND(?l AS ?s)
    }
    {
      GRAPH <ckr:global-inf> {
        {
          ?d olap:rollsUpTo ?r .
        } UNION {
          ?r olap:rollsUpTo ?d .
        }
      }
      GRAPH ckr:global {
        ?d olap:atLevel ?l .
      }
    } VALUES ?r {cube:Level_Aircraft_All-All
        cube:Level_Location_Region-LOVV
        cube:Level_Date_All-All}
  }
} UNION {
  SELECT ?g ?s ?p ?o WHERE {
    GRAPH ckr:global {
      ?s ?p ?o .
    }
```

```
      VALUES ?p {
        owl:allValuesFrom
        owl:onProperty
        owl:disjointWith
        owl:hasSelf
        owl:propertyChainAxiom
        rdf:first
        rdf:rest
        rdfs:subClassOf
        rdfs:subPropertyOf
        rdfs:range
        rdfs:domain
        rdfs:comment
      }
      BIND(ckr:global AS ?g)
    }
  } UNION {
    SELECT ?g ?s ?p ?o WHERE {
      GRAPH ckr:global {
        ?s ?p ?o .
      }
      VALUES ?o {
        owl:Ontology
        owl:Class
        owl:ObjectProperty
        owl:FunctionalProperty
      }
      BIND(ckr:global AS ?g)
    }
  } UNION {
    SELECT ?g ?s ?p ?o WHERE {
      BIND(<ckr:global-inf> AS ?g)
      BIND(<ckr:global-inf> AS ?s)
      BIND(ckr:derivedFrom AS ?p)
      BIND(<ckr:global-inf> AS ?o)
    }
  } UNION {
    SELECT ?g ?s ?p ?o WHERE {
      BIND(<ckr:global-inf> AS ?g)
      BIND(<ckr:global-inf> AS ?s)
      BIND(ckr:derivedFrom AS ?p)
      BIND(ckr:global AS ?o)
    }
  }
  BIND("+" AS ?op)
}
```

4.5.2 Merge union

Listing 11 shows an example merge-union operation that rolls up facts to aircraft type, region, and all-date granularity. The `WHERE` clause consists of several sub-`SELECT` statements conjoined by `UNION`. The result delta table of the first sub-`SELECT` (Lines 14-114) selects and, if necessary, generates contexts and knowledge modules at the argument granularity level. The result delta table of the second sub-`SELECT` (Lines 116-195) updates the coverage relationships to include the newly-generated contexts. The result delta table of the third sub-`SELECT` (Lines 197-223) sets the merged contexts null by assigning them the `ckr:Null` type. The result delta table of the fourth sub-`SELECT` (Lines 225-286) performs the actual merge of the knowledge modules. The result delta table of the fifth and sixth sub-`SELECT` statements (Lines 288-409) deletes the knowledge in and about the modules associated with merged contexts.

A new context should only be generated if there is anything at all to merge (Listing 11, Lines 79-92) underneath the supposed context. Optionally, if a context at the roll-up granularity (and its knowledge modules) already exists (Lines 93-107), no context shall be generated and the existing context/modules shall be bound to the corresponding variables (Line 110).

Listing 11: Example SPARQL merge-union operation

```
1  PREFIX owl: <http://www.w3.org/2002/07/owl#>
2  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3  PREFIX obj: <http://example.org/kgolap/object-model#>
4  PREFIX xml: <http://www.w3.org/XML/1998/namespace>
5  PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
6  PREFIX olap: <http://dkm.fbk.eu/ckr/olap-model#>
7  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
8  PREFIX cube: <http://example.org/kgolap/cube-model#>
9  PREFIX ckr: <http://dkm.fbk.eu/ckr/meta#>
10 PREFIX onto: <http://www.ontotext.com/>
11
12 SELECT DISTINCT ?s ?p ?o ?g ?op WHERE {
13   {
14     SELECT ?g ?s ?p ?o ?op WHERE {
15       {
16         BIND(ckr:global AS ?g)
17         BIND(?ctx AS ?s)
18         BIND(rdf:type AS ?p)
19         BIND(olap:Cell AS ?o)
20       } UNION {
21         BIND(ckr:global AS ?g)
22         BIND(?ctx AS ?s)
23         BIND(ckr:hasAssertedModule AS ?p)
24         BIND(?mod AS ?o)
25       } UNION {
26         BIND(ckr:global AS ?g)
27         BIND(?ctx AS ?s)
28         BIND(cube:hasAircraft AS ?p)
29         BIND(?d1 AS ?o)
30       } UNION {
31         BIND(ckr:global AS ?g)
32         BIND(?ctx AS ?s)
33         BIND(cube:hasLocation AS ?p)
34         BIND(?d2 AS ?o)
35       } UNION {
36         BIND(ckr:global AS ?g)
37         BIND(?ctx AS ?s)
38         BIND(cube:hasDate AS ?p)
```

```
          BIND(?d3 AS ?o)
        } UNION {
          BIND(<ckr:global-inf> AS ?g)
          BIND(?ctx AS ?s)
          BIND(ckr:hasModule AS ?p)
          BIND(?mod AS ?o)
        } UNION {
          BIND(<ckr:global-inf> AS ?g)
          BIND(?ctx AS ?s)
          BIND(ckr:hasModule AS ?p)
          BIND(?inf AS ?o)
        } UNION {
          BIND(<ckr:global-inf> AS ?g)
          BIND(?inf AS ?s)
          BIND(ckr:closureOf AS ?p)
          BIND(?ctx AS ?o)
        } UNION {
          BIND(?inf AS ?g)
          BIND(?inf AS ?s)
          BIND(ckr:derivedFrom AS ?p)
          BIND(ckr:global AS ?o)
        } UNION {
          BIND(?inf AS ?g)
          BIND(?inf AS ?s)
          BIND(ckr:derivedFrom AS ?p)
          BIND(<ckr:global-inf> AS ?o)
        } UNION {
          BIND(?inf AS ?g)
          BIND(?inf AS ?s)
          BIND(ckr:derivedFrom AS ?p)
          BIND(?mod AS ?o)
        }
        {
          SELECT ?ctx ?mod ?inf ?d1 ?d2 ?d3 ?d4 WHERE {
            GRAPH ckr:global {
              ?d1 olap:atLevel cube:Level_Aircraft_Type .
              ?d2 olap:atLevel cube:Level_Location_Region .
              ?d3 olap:atLevel cube:Level_Date_All .
              ?d4 olap:atLevel cube:Level_Importance_All .
            }
            FILTER EXISTS {
              GRAPH ckr:global {
                ?covered cube:hasAircraft ?y1 .
                ?covered cube:hasLocation ?y2 .
                ?covered cube:hasDate ?y3 .
                ?covered cube:hasImportance ?y4 .
              }
              GRAPH <ckr:global-inf> {
                ?y1 olap:rollsUpTo ?d1 .
                ?y2 olap:rollsUpTo ?d2 .
                ?y3 olap:rollsUpTo ?d3 .
                ?y4 olap:rollsUpTo ?d4 .
              }
            }
```

```
        OPTIONAL {
          GRAPH ckr:global {
            ?ctx1 ckr:hasAssertedModule ?mod1 .
            ?ctx1 cube:hasAircraft ?d1 .
            ?ctx1 cube:hasLocation ?d2 .
            ?ctx1 cube:hasDate ?d3 .
            ?ctx1 olap:atLevel cube:Level_Importance_All .
          }
          OPTIONAL {
            GRAPH <ckr:global-inf> {
              ?ctx1 ckr:hasModule ?inf1 .
              ?inf1 ckr:closureOf ?ctx1 .
            }
          }
        }
        BIND(IF(!BOUND(?ctx1), IRI(CONCAT(STR(cube:), ’Ctx’,
            ’-’, IF(STRAFTER(STR(?d1), ’#’) != ’’,
            STRAFTER(STR(?d1), ’#’), STRAFTER(STR(?d1),
            ’urn:uuid:’)), ’-’, IF(STRAFTER(STR(?d2), ’#’)
            != ’’, STRAFTER(STR(?d2), ’#’),
            STRAFTER(STR(?d2), ’urn:uuid:’)), ’-’,
            IF(STRAFTER(STR(?d3), ’#’) != ’’,
            STRAFTER(STR(?d3), ’#’), STRAFTER(STR(?d3),
            ’urn:uuid:’)), ’-’, IF(STRAFTER(STR(?d4), ’#’)
            != ’’, STRAFTER(STR(?d4), ’#’),
            STRAFTER(STR(?d4), ’urn:uuid:’)))), ?ctx1) AS
            ?ctx)
        BIND(IF(!BOUND(?mod1), IRI(CONCAT(STR(cube:), ’Ctx’,
            ’-’, IF(STRAFTER(STR(?d1), ’#’) != ’’,
            STRAFTER(STR(?d1), ’#’), STRAFTER(STR(?d1),
            ’urn:uuid:’)), ’-’, IF(STRAFTER(STR(?d2), ’#’)
            != ’’, STRAFTER(STR(?d2), ’#’),
            STRAFTER(STR(?d2), ’urn:uuid:’)), ’-’,
            IF(STRAFTER(STR(?d3), ’#’) != ’’,
            STRAFTER(STR(?d3), ’#’), STRAFTER(STR(?d3),
            ’urn:uuid:’)), ’-’, IF(STRAFTER(STR(?d4), ’#’)
            != ’’, STRAFTER(STR(?d4), ’#’),
            STRAFTER(STR(?d4), ’urn:uuid:’)), ’-mod’)),
            ?mod1) AS ?mod)
        BIND(IF(!BOUND(?inf1), IRI(CONCAT(STR(cube:), ’Ctx’,
            ’-’, IF(STRAFTER(STR(?d1), ’#’) != ’’,
            STRAFTER(STR(?d1), ’#’), STRAFTER(STR(?d1),
            ’urn:uuid:’)), ’-’, IF(STRAFTER(STR(?d2), ’#’)
            != ’’, STRAFTER(STR(?d2), ’#’),
            STRAFTER(STR(?d2), ’urn:uuid:’)), ’-’,
            IF(STRAFTER(STR(?d3), ’#’) != ’’,
            STRAFTER(STR(?d3), ’#’), STRAFTER(STR(?d3),
            ’urn:uuid:’)), ’-’, IF(STRAFTER(STR(?d4), ’#’)
            != ’’, STRAFTER(STR(?d4), ’#’),
            STRAFTER(STR(?d4), ’urn:uuid:’)), ’-inf’)),
            ?inf1) AS ?inf)
      }
    }
    BIND("+" AS ?op)
```

```
    }
  } UNION {
    SELECT ?g ?s ?p ?o ?op WHERE {
      {
        SELECT ?ctx ?d1 ?d2 ?d3 ?d4 ?x1 ?x2 ?x3 ?x4 WHERE {
          GRAPH ckr:global {
            ?ctx cube:hasAircraft ?x1 .
            ?d1 olap:atLevel cube:Level_Aircraft_Type .
            ?ctx cube:hasLocation ?x2 .
            ?d2 olap:atLevel cube:Level_Location_Region .
            ?ctx cube:hasDate ?x3 .
            ?d3 olap:atLevel cube:Level_Date_All .
            ?ctx cube:hasImportance ?x4 .
            ?d4 olap:atLevel cube:Level_Importance_All .
          }
          FILTER NOT EXISTS {
            GRAPH ckr:global {
              ?ctx1 cube:hasAircraft ?d1 .
              ?ctx1 cube:hasLocation ?d2 .
              ?ctx1 cube:hasDate ?d3 .
              ?ctx1 cube:hasImportance ?d4 .
            }
          }
          FILTER EXISTS {
            GRAPH ckr:global {
              ?covered cube:hasAircraft ?y1 .
              ?covered cube:hasLocation ?y2 .
              ?covered cube:hasDate ?y3 .
              ?covered cube:hasImportance ?y4 .
            }
            GRAPH <ckr:global-inf> {
              ?y1 olap:rollsUpTo ?d1 .
              ?y2 olap:rollsUpTo ?d2 .
              ?y3 olap:rollsUpTo ?d3 .
              ?y4 olap:rollsUpTo ?d4 .
            }
          }
        }
      }
      {
        GRAPH <ckr:global-inf> {
          ?d1 olap:rollsUpTo ?x1 .
          ?d2 olap:rollsUpTo ?x2 .
          ?d3 olap:rollsUpTo ?x3 .
          ?d4 olap:rollsUpTo ?x4 .
        }

        BIND(<ckr:global-inf> AS ?g)
        BIND(?ctx AS ?s)
        BIND(olap:covers AS ?p)
        BIND(IRI(CONCAT(STR(cube:), 'Ctx', '-',
            IF(STRAFTER(STR(?d1), '#') != '',
            STRAFTER(STR(?d1), '#'), STRAFTER(STR(?d1),
            'urn:uuid:')), '-', IF(STRAFTER(STR(?d2), '#') !=
```

```
        '', STRAFTER(STR(?d2), '#'), STRAFTER(STR(?d2),
        'urn:uuid:')), '-', IF(STRAFTER(STR(?d3), '#') !=
        '', STRAFTER(STR(?d3), '#'), STRAFTER(STR(?d3),
        '-', IF(STRAFTER(STR(?d4), '#') != '',
        STRAFTER(STR(?d4), '#'), STRAFTER(STR(?d4),
        'urn:uuid:')))) AS ?o)
  } UNION {
    GRAPH <ckr:global-inf> {
      ?x1 olap:rollsUpTo ?d1 .
      ?x2 olap:rollsUpTo ?d2 .
      ?x3 olap:rollsUpTo ?d3 .
      ?x4 olap:rollsUpTo ?d4 .
    }

    BIND(<ckr:global-inf> AS ?g)
    BIND(IRI(CONCAT(STR(cube:), 'Ctx', '-',
        IF(STRAFTER(STR(?d1), '#') != '',
        STRAFTER(STR(?d1), '#'), STRAFTER(STR(?d1),
        'urn:uuid:')), '-', IF(STRAFTER(STR(?d2), '#') !=
        '', STRAFTER(STR(?d2), '#'), STRAFTER(STR(?d2),
        'urn:uuid:')), '-', IF(STRAFTER(STR(?d3), '#') !=
        '', STRAFTER(STR(?d3), '#'), STRAFTER(STR(?d3),
        '-', IF(STRAFTER(STR(?d4), '#') != '',
        STRAFTER(STR(?d4), '#'), STRAFTER(STR(?d4),
        'urn:uuid:')))) AS ?s)
    BIND(olap:covers AS ?p)
    BIND(?ctx AS ?o)
  } UNION {
    GRAPH <ckr:global-inf> {
      ?d1 olap:rollsUpTo ?x1 .
      ?d2 olap:rollsUpTo ?x2 .
      ?d3 olap:rollsUpTo ?x3 .
      ?d4 olap:rollsUpTo ?x4 .
    }

    GRAPH ckr:global {
      ?ctx ckr:hasAssertedModule ?mod .
    }

    BIND(<ckr:global-inf> AS ?g)
    BIND(IRI(CONCAT(STR(cube:), 'Ctx', '-',
        IF(STRAFTER(STR(?d1), '#') != '',
        STRAFTER(STR(?d1), '#'), STRAFTER(STR(?d1),
        'urn:uuid:')), '-', IF(STRAFTER(STR(?d2), '#') !=
        '', STRAFTER(STR(?d2), '#'), STRAFTER(STR(?d2),
        'urn:uuid:')), '-', IF(STRAFTER(STR(?d3), '#') !=
        '', STRAFTER(STR(?d3), '#'), STRAFTER(STR(?d3),
        '-', IF(STRAFTER(STR(?d4), '#') != '',
        STRAFTER(STR(?d4), '#'), STRAFTER(STR(?d4),
        'urn:uuid:')), '-inf')) AS ?s)
    BIND(ckr:derivedFrom AS ?p)
    BIND(?mod AS ?o)
  }
  BIND("+" AS ?op)
```

```
    }
  } UNION {
    SELECT ?g ?s ?p ?o ?op WHERE {
      {
        GRAPH ckr:global {
          ?s cube:hasAircraft/olap:atLevel ?l1 .
          ?s cube:hasLocation/olap:atLevel ?l2 .
          ?s cube:hasDate/olap:atLevel ?l3 .
          ?s cube:hasImportance/olap:atLevel ?l4 .
        }
        GRAPH <ckr:global-inf> {
          ?l1 olap:rollsUpTo cube:Level_Aircraft_Type .
          ?l2 olap:rollsUpTo cube:Level_Location_Region .
          ?l3 olap:rollsUpTo cube:Level_Date_All .
          ?l4 olap:rollsUpTo cube:Level_Importance_All .
        }
      } MINUS {
        GRAPH ckr:global {
          ?s cube:hasAircraft/olap:atLevel
              cube:Level_Aircraft_Type .
          ?s cube:hasLocation/olap:atLevel
              cube:Level_Location_Region .
          ?s cube:hasDate/olap:atLevel cube:Level_Date_All .
          ?s cube:hasImportance/olap:atLevel
              cube:Level_Importance_All .
        }
      }
      BIND(ckr:global AS ?g)
      BIND(rdf:type AS ?p)
      BIND(ckr:Null AS ?o)
      BIND("+" AS ?op)
    }
  } UNION {
    SELECT DISTINCT ?s ?p ?o ?g ?op WHERE {
      GRAPH ?m {
        ?s ?p ?o .
      }
      FILTER(?p != ckr:derivedFrom)
      {
        SELECT ?g ?m WHERE {
          {
            SELECT ?ctx ?g ?m WHERE {
              GRAPH <ckr:global-inf> {
                ?ctx olap:covers? ?ctx1 .
              }
              GRAPH ckr:global {
                ?ctx1 ckr:hasAssertedModule ?m .
              }
              OPTIONAL {
                GRAPH ckr:global {
                  ?ctx ckr:hasAssertedModule ?mod .
                }
              }
```

```
        BIND(IF(!BOUND(?mod), IRI(CONCAT(STR(?ctx),
            '-mod')), ?mod) AS ?g)
      }
    } UNION {
      SELECT ?ctx ?g ?m WHERE {
        GRAPH <ckr:global-inf> {
          ?ctx olap:covers? ?ctx1 .
        }
        MINUS {
          GRAPH ckr:global {
            ?ctx1 ckr:hasAssertedModule ?m .
          }
        }
        OPTIONAL {
          GRAPH <ckr:global-inf> {
            ?ctx ckr:hasModule ?inf .
          }
          MINUS {
            GRAPH ckr:global {
              ?ctx ckr:hasAssertedModule ?inf .
            }
          }
        }
        BIND(IF(!BOUND(?inf), IRI(CONCAT(STR(?ctx),
            '-inf')), ?inf) AS ?g)
      }
    }
    {
      SELECT ?ctx WHERE {
        GRAPH ckr:global {
          ?ctx cube:hasAircraft ?d1 .
          ?ctx cube:hasLocation ?d2 .
          ?ctx cube:hasDate ?d3 .
          ?ctx cube:hasImportance ?d4 .
        }
        GRAPH ckr:global {
          ?d1 olap:atLevel cube:Level_Aircraft_Type .
          ?d2 olap:atLevel cube:Level_Location_Region .
          ?d3 olap:atLevel cube:Level_Date_All .
          ?d4 olap:atLevel cube:Level_Importance_All .
        }
      }
    }
  }
} UNION {
  SELECT ?g ?m WHERE {
    {
      SELECT ?ctx ?g ?m WHERE {
        {
          SELECT ?m ?d1 ?d2 ?d3 WHERE {
            GRAPH ckr:global {
              ?d1 olap:atLevel cube:Level_Aircraft_Type .
              ?d2 olap:atLevel
                  cube:Level_Location_Region .
```

```
                    ?d3 olap:atLevel cube:Level_Date_All .
                    ?d4 olap:atLevel cube:Level_Importance_All
                        .
                  }
                  FILTER NOT EXISTS {
                    GRAPH ckr:global {
                      ?ctx cube:hasAircraft ?d1 .
                      ?ctx cube:hasLocation ?d2 .
                      ?ctx cube:hasDate ?d3 .
                      ?ctx cube:hasImportance ?d4 .
                    }
                  }
                  GRAPH ckr:global {
                    ?covered ckr:hasAssertedModule ?m .
                    ?covered cube:hasAircraft ?y1 .
                    ?covered cube:hasLocation ?y2 .
                    ?covered cube:hasDate ?y3 .
                    ?covered cube:hasImportance ?y4 .
                  }
                  GRAPH <ckr:global-inf> {
                    ?y1 olap:rollsUpTo ?d1 .
                    ?y2 olap:rollsUpTo ?d2 .
                    ?y3 olap:rollsUpTo ?d3 .
                    ?y4 olap:rollsUpTo ?d4 .
                  }
                }
              }

              BIND(IRI(CONCAT(STR(cube:), 'Ctx', '-',
                  IF(STRAFTER(STR(?d1), '#') != '',
                  STRAFTER(STR(?d1), '#'), STRAFTER(STR(?d1),
                  'urn:uuid:')), '-', IF(STRAFTER(STR(?d2),
                  '#') != '', STRAFTER(STR(?d2), '#'),
                  STRAFTER(STR(?d2), 'urn:uuid:')), '-',
                  IF(STRAFTER(STR(?d3), '#') != '',
                  STRAFTER(STR(?d3), '#'), STRAFTER(STR(?d3),
                  '-', IF(STRAFTER(STR(?d4), '#') != '',
                  STRAFTER(STR(?d4), '#'), STRAFTER(STR(?d4),
                  'urn:uuid:')))) AS ?ctx)
              BIND(IRI(CONCAT(STR(cube:), 'Ctx', '-',
                  IF(STRAFTER(STR(?d1), '#') != '',
                  STRAFTER(STR(?d1), '#'), STRAFTER(STR(?d1),
                  'urn:uuid:')), '-', IF(STRAFTER(STR(?d2),
                  '#') != '', STRAFTER(STR(?d2), '#'),
                  STRAFTER(STR(?d2), 'urn:uuid:')), '-',
                  IF(STRAFTER(STR(?d3), '#') != '',
                  STRAFTER(STR(?d3), '#'), STRAFTER(STR(?d3),
                  '-', IF(STRAFTER(STR(?d4), '#') != '',
                  STRAFTER(STR(?d4), '#'), STRAFTER(STR(?d4),
                  'urn:uuid:')), '-mod')) AS ?g)
            }
          } UNION {
            SELECT ?ctx ?g ?m WHERE {
              {
```

```
    SELECT ?m ?d1 ?d2 ?d3 ?d4 WHERE {
      GRAPH ckr:global {
        ?d1 olap:atLevel cube:Level_Aircraft_Type .
        ?d2 olap:atLevel
            cube:Level_Location_Region .
        ?d3 olap:atLevel cube:Level_Date_All .
        ?d4 olap:atLevel cube:Level_Importance_All
            .
      }
      FILTER NOT EXISTS {
        GRAPH ckr:global {
          ?ctx cube:hasAircraft ?d1 .
          ?ctx cube:hasLocation ?d2 .
          ?ctx cube:hasDate ?d3 .
          ?ctx cube:hasImportance ?d4 .
        }
      }
      GRAPH ckr:global {
        ?covered cube:hasAircraft ?y1 .
        ?covered cube:hasLocation ?y2 .
        ?covered cube:hasDate ?y3 .
        ?covered cube:hasImportance ?y4 .
      }
      GRAPH <ckr:global-inf> {
        ?y1 olap:rollsUpTo ?d1 .
        ?y2 olap:rollsUpTo ?d2 .
        ?y3 olap:rollsUpTo ?d3 .
        ?y4 olap:rollsUpTo ?d4 .
        ?covered ckr:hasModule ?m .
      }
      MINUS {
        GRAPH ckr:global {
          ?covered ckr:hasAssertedModule ?m .
        }
      }
    }
  }

  BIND(IRI(CONCAT(STR(cube:), 'Ctx', '-',
      IF(STRAFTER(STR(?d1), '#') != '',
      STRAFTER(STR(?d1), '#'), STRAFTER(STR(?d1),
      'urn:uuid:')), '-', IF(STRAFTER(STR(?d2),
      '#') != '', STRAFTER(STR(?d2), '#'),
      STRAFTER(STR(?d2), 'urn:uuid:')), '-',
      IF(STRAFTER(STR(?d3), '#') != '',
      STRAFTER(STR(?d3), '#'), STRAFTER(STR(?d3),
      'urn:uuid:'))))) AS ?ctx)
  BIND(IRI(CONCAT(STR(cube:), 'Ctx', '-',
      IF(STRAFTER(STR(?d1), '#') != '',
      STRAFTER(STR(?d1), '#'), STRAFTER(STR(?d1),
      'urn:uuid:')), '-', IF(STRAFTER(STR(?d2),
      '#') != '', STRAFTER(STR(?d2), '#'),
      STRAFTER(STR(?d2), 'urn:uuid:')), '-',
      IF(STRAFTER(STR(?d3), '#') != '',
```

```
                STRAFTER(STR(?d3), ’#’), STRAFTER(STR(?d3),
                ’-’, IF(STRAFTER(STR(?d4), ’#’) != ’’,
                STRAFTER(STR(?d4), ’#’), STRAFTER(STR(?d4),
                ’urn:uuid:’)), ’-inf’)) AS ?g)
          }
        }
      }
    }
    BIND("+" AS ?op)
  }
} UNION {
  SELECT ?g ?s ?p ?o ?op WHERE {
    {
      GRAPH ckr:global {
        ?ctx cube:hasAircraft/olap:atLevel ?l1 .
        ?ctx cube:hasLocation/olap:atLevel ?l2 .
        ?ctx cube:hasDate/olap:atLevel ?l3 .
        ?ctx cube:hasImportance/olap:atLevel ?l4 .
      }
      GRAPH <ckr:global-inf> {
        ?l1 olap:rollsUpTo cube:Level_Aircraft_Type .
        ?l2 olap:rollsUpTo cube:Level_Location_Region .
        ?l3 olap:rollsUpTo cube:Level_Date_All .
        ?l4 olap:rollsUpTo cube:Level_Importance_All .
      }
    } MINUS {
      GRAPH ckr:global {
        ?ctx cube:hasAircraft/olap:atLevel
            cube:Level_Aircraft_Type .
        ?ctx cube:hasLocation/olap:atLevel
            cube:Level_Location_Region .
        ?ctx cube:hasDate/olap:atLevel cube:Level_Date_All .
        ?ctx cube:hasImportance/olap:atLevel
            cube:Level_Importance_All .
      }
    }
    {
      GRAPH ckr:global {
        ?ctx ckr:hasAssertedModule ?g .
      }
    } UNION {
      GRAPH <ckr:global-inf> {
        ?ctx ckr:hasModule ?g .
      }
    }
    GRAPH ?g {
      ?s ?p ?o
    }
    BIND("-" AS ?op)
  }
} UNION {
  SELECT ?g ?s ?p ?o ?op WHERE {
    {
      GRAPH ckr:global {
```

```
          ?ctx cube:hasAircraft/olap:atLevel ?l1 .
          ?ctx cube:hasLocation/olap:atLevel ?l2 .
          ?ctx cube:hasDate/olap:atLevel ?l3 .
          ?ctx cube:hasImportance/olap:atLevel ?l4 .
        }
        GRAPH <ckr:global-inf> {
          ?l1 olap:rollsUpTo cube:Level_Aircraft_Type .
          ?l2 olap:rollsUpTo cube:Level_Location_Region .
          ?l3 olap:rollsUpTo cube:Level_Date_All .
          ?l4 olap:rollsUpTo cube:Level_Importance_All .
        }
      } MINUS {
        GRAPH ckr:global {
          ?ctx cube:hasAircraft/olap:atLevel
              cube:Level_Aircraft_Type .
          ?ctx cube:hasLocation/olap:atLevel
              cube:Level_Location_Region .
          ?ctx cube:hasDate/olap:atLevel cube:Level_Date_All .
          ?ctx cube:hasImportance/olap:atLevel
              cube:Level_Importance_All .
        }
      }
      {
        GRAPH ckr:global {
          ?ctx ckr:hasAssertedModule ?m .
        }
      } UNION {
        GRAPH <ckr:global-inf> {
          ?ctx ckr:hasModule ?m .
        }
      }
      {
        GRAPH ?g {
          ?m ?p ?o
        }
        BIND(?m AS ?s)
      } UNION {
        GRAPH ?g {
          ?s ?p ?m
        }
        BIND(?m AS ?o)
      }
      BIND("-" AS ?op)
    }
  }
}
```

4.5.3 Triple-generating abstraction

Consider a triple-generating abstraction that replaces the ManoeuvringAreaUsage individuals with their usage type in all cells at a particular granularity. The SPARQL query in Listing 12, Lines 17-30, selects from all cells at aircraft model, location segment, and day granularity the statements with a predicate other than `usageType` –

the grouping property. In these statements, ManoeuvringAreaUsage instances are to be replaced by their usage type. The ?a binding refers to the subjects of these statements, the ?d binding to the objects, and for both the usage type is retrieved, given there is any (Lines 31-74). The query only keeps those statements where either subject or object has a usage type (Line 77). Non-usageType statements are then returned (Lines 16-75) along with the potential grouping, as ?x and ?y, respectively. Then, the statements of this result set are then duplicated, with one entry for insertion and the other for deletion (Line 80). Updates only apply to a context's asserted module (Line 18) but the determination of which individuals are grouped, i.e., only those of type ManoeuvringAreaUsage, considers inherited and inferred knowledge.

Listing 12: Example SPARQL triple-generation abstraction operation

```
1  PREFIX owl: <http://www.w3.org/2002/07/owl#>
2  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3  PREFIX obj: <http://example.org/kgolap/object-model#>
4  PREFIX xml: <http://www.w3.org/XML/1998/namespace>
5  PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
6  PREFIX olap: <http://dkm.fbk.eu/ckr/olap-model#>
7  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
8  PREFIX cube: <http://example.org/kgolap/cube-model#>
9  PREFIX ckr: <http://dkm.fbk.eu/ckr/meta#>
10 PREFIX onto: <http://www.ontotext.com/>
11
12 SELECT DISTINCT ?g ?s ?p ?o ?op WHERE {
13   {
14     SELECT ?m ?a ?b ?c ?d ?e ?op ?ctx WHERE {
15       {
16         SELECT ?m ?a ?x ?c ?d ?y ?ctx WHERE {
17           GRAPH ckr:global {
18             ?ctx ckr:hasAssertedModule ?m .
19           }
20           GRAPH ckr:global {
21             ?ctx cube:hasAircraft/olap:atLevel
                   cube:Level_Aircraft_Model .
22             ?ctx cube:hasLocation/olap:atLevel
                   cube:Level_Location_Segment .
23             ?ctx cube:hasDate/olap:atLevel cube:Level_Date_Day
                   .
24             ?ctx cube:hasImportance/olap:atLevel
                   cube:Level_Importance_Importance .
25           }
26           GRAPH ?m {
27             ?a ?c ?d .
28             FILTER(?c != rdf:type)
29             FILTER(?c != obj:usageType)
30           }
31           OPTIONAL {
32             SELECT ?a ?x ?ctx WHERE {
33               GRAPH <ckr:global-inf> {
34                 ?inf1 ckr:closureOf ?ctx .
35               }
36               GRAPH ?inf1 {
```

```
                ?inf1 ckr:derivedFrom ?m1 .
              }
              GRAPH ?m1 {
                ?a obj:usageType ?x .
              }
              GRAPH <ckr:global-inf> {
                ?inf3 ckr:closureOf ?ctx .
              }
              GRAPH ?inf3 {
                ?inf3 ckr:derivedFrom ?m3 .
              }
              GRAPH ?m3 {
                ?a rdf:type obj:ManoeuvringAreaUsage .
              }
            }
          }
          OPTIONAL {
            SELECT ?d ?y ?ctx WHERE {
              GRAPH <ckr:global-inf> {
                ?inf2 ckr:closureOf ?ctx .
              }
              GRAPH ?inf2 {
                ?inf2 ckr:derivedFrom ?m2 .
              }
              GRAPH ?m2 {
                ?d obj:usageType ?y .
              }
              GRAPH <ckr:global-inf> {
                ?inf4 ckr:closureOf ?ctx .
              }
              GRAPH ?inf4 {
                ?inf4 ckr:derivedFrom ?m4 .
              }
              GRAPH ?m4 {
                ?d rdf:type obj:ManoeuvringAreaUsage .
              }
            }
          }
        }
      }
      FILTER(BOUND(?x) || BOUND(?y))
      BIND(IF(!BOUND(?x), ?a, ?x) AS ?b)
      BIND(IF(!BOUND(?y), ?d, ?y) AS ?e)
      VALUES ?op { "-" "+" }
    }
  }
  BIND(?m AS ?g)
  BIND(IF(?op = "-", ?a, ?b) AS ?s)
  BIND(?c AS ?p)
  BIND(IF(?op = "-", ?d, ?e) AS ?o)
}
```

4.5.4 Individual-generating abstraction

Listing 13 shows the SPARQL query for individual-generating abstraction that groups individuals by properties, generating a grouping for individuals with the same group-by property values. In the example, all individuals with the same operationalStatus value are grouped together.

The SPARQL query for individual-generating abstraction (grouping by properties) consists of two sub-`SELECT` statements. The first sub-`SELECT` (Listing 13, Lines 14-47) retrieves insertions into the contexts' respective knowledge module, namely the definition of grouping properties for the grouped individuals. The second sub-`SELECT` (Lines 49-110) performs the actual abstraction, i.e., replace grouped individuals in the subjects and objects of statements.

The SPARQL query for individual-generating abstraction follows a similar scheme as the SPARQL query for triple-generating abstraction, the main difference being that a group individual is generated. For the generated groups, IRIs must be defined. The string representation of IRIs of grouping property values are concatenated and a hash function (`SHA512`) applied. The concatenation that generates the IRI could also include the context name in order to disallow the same groups to be generated in different contexts. Furthermore, an externally provided universally unique identifier, generated at each invocation, could be included in the generated individual IRIs in order to ensure uniqueness across query operations.

Listing 13: Example SPARQL individual-generating abstraction operation

```
1  PREFIX owl: <http://www.w3.org/2002/07/owl#>
2  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3  PREFIX obj: <http://example.org/kgolap/object-model#>
4  PREFIX xml: <http://www.w3.org/XML/1998/namespace>
5  PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
6  PREFIX olap: <http://dkm.fbk.eu/ckr/olap-model#>
7  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
8  PREFIX cube: <http://example.org/kgolap/cube-model#>
9  PREFIX ckr: <http://dkm.fbk.eu/ckr/meta#>
10 PREFIX onto: <http://www.ontotext.com/>
11
12 SELECT DISTINCT ?g ?s ?p ?o ?op WHERE {
13   {
14     SELECT ?g ?s ?p ?o ?op ?ctx WHERE {
15       {
16         SELECT ?m ?r ?gr ?ctx WHERE {
17           GRAPH ckr:global {
18             ?ctx ckr:hasAssertedModule ?m .
19           }
20           GRAPH ckr:global {
21             ?ctx cube:hasAircraft/olap:atLevel
                   cube:Level_Aircraft_Model .
22             ?ctx cube:hasLocation/olap:atLevel
                   cube:Level_Location_Segment .
23             ?ctx cube:hasDate/olap:atLevel cube:Level_Date_Day
                   .
24             ?ctx cube:hasImportance/olap:atLevel
                   cube:Level_Importance_Importance .
25           }
```

```
          {
            SELECT ?r ?gr1 ?ctx WHERE {
              GRAPH <ckr:global-inf> {
                ?inf1 ckr:closureOf ?ctx .
              }
              GRAPH ?inf1 {
                ?inf1 ckr:derivedFrom ?m1 .
              }
              GRAPH ?m1 {
                ?r obj:operationalStatus ?gr1 .
              }
            }
          }
          BIND(IRI(CONCAT(STR(obj:),
              SHA512(CONCAT(STR(?gr1))))) AS ?gr)
        }
      }
      BIND(?m AS ?g)
      BIND(?r AS ?s)
      BIND(obj:grouping AS ?p)
      BIND(?gr AS ?o)
      BIND("+" AS ?op)
    }
  } UNION {
    SELECT ?g ?s ?p ?o ?op ?ctx WHERE {
      {
        SELECT ?m ?a ?b ?c ?d ?e ?op ?ctx WHERE {
          {
            SELECT ?m ?a ?x ?c ?d ?y ?ctx WHERE {
              GRAPH ckr:global {
                ?ctx ckr:hasAssertedModule ?m .
              }
              GRAPH ckr:global {
                ?ctx cube:hasAircraft/olap:atLevel
                    cube:Level_Aircraft_Model .
                ?ctx cube:hasLocation/olap:atLevel
                    cube:Level_Location_Segment .
                ?ctx cube:hasDate/olap:atLevel
                    cube:Level_Date_Day .
                ?ctx cube:hasImportance/olap:atLevel
                    cube:Level_Importance_Importance .
              }
              GRAPH ?m {
                ?a ?c ?d .
                FILTER(?c != rdf:type)
                FILTER(?c != obj:grouping)
              }
              OPTIONAL {
                SELECT ?a ?x ?ctx WHERE {
                  {
                    SELECT ?a ?x1 ?ctx WHERE {
                      GRAPH <ckr:global-inf> {
                        ?inf1 ckr:closureOf ?ctx .
                      }
```

```
                      GRAPH ?inf1 {
                        ?inf1 ckr:derivedFrom ?m1 .
                      }
                      GRAPH ?m1 {
                        ?a obj:operationalStatus ?x1 .
                      }
                    }
                  }
                  BIND(IRI(CONCAT(STR(obj:),
                      SHA512(CONCAT(STR(?x1))))) AS ?x)
                }
              }
              OPTIONAL {
                SELECT ?d ?y ?ctx  WHERE {
                  {
                    SELECT ?d ?y1 ?ctx WHERE {
                      GRAPH <ckr:global-inf> {
                        ?inf3 ckr:closureOf ?ctx .
                      }
                      GRAPH ?inf3 {
                        ?inf3 ckr:derivedFrom ?m3 .
                      }
                      GRAPH ?m3 {
                        ?d obj:operationalStatus ?y1 .
                      }
                    }
                  }
                  BIND(IRI(CONCAT(STR(obj:),
                      SHA512(CONCAT(STR(?y1))))) AS ?y)
                }
              }
            }
          }
          FILTER(BOUND(?x) || BOUND(?y))
          BIND(IF(!BOUND(?x), ?a, ?x) AS ?b)
          BIND(IF(!BOUND(?y), ?d, ?y) AS ?e)
          VALUES ?op { "-" "+" }
        }
      }
      BIND(?m AS ?g)
      BIND(IF(?op = "-", ?a, ?b) AS ?s)
      BIND(?c AS ?p)
      BIND(IF(?op = "-", ?d, ?e) AS ?o)
    }
  }
}
```

Listing 14 shows an example of a query with multiple grouping properties, one of which is a role composition. All individuals with the same operational status that reference a ManoeuvringAreaUsage individual of the same usage type are grouped together. For each grouping property, a separate sub-`SELECT` returns the values that the individuals have for that property. The string representations of the individual values are then concatenated and hashed.

Listing 14: Example SPARQL individual-generating abstraction operation with multiple grouping properties

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX obj: <http://example.org/kgolap/object-model#>
PREFIX xml: <http://www.w3.org/XML/1998/namespace>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX olap: <http://dkm.fbk.eu/ckr/olap-model#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX cube: <http://example.org/kgolap/cube-model#>
PREFIX ckr: <http://dkm.fbk.eu/ckr/meta#>
PREFIX onto: <http://www.ontotext.com/>

SELECT DISTINCT ?g ?s ?p ?o ?op WHERE {
  {
    SELECT ?g ?s ?p ?o ?op ?ctx WHERE {
      {
        SELECT ?m ?r ?gr ?ctx WHERE {
          GRAPH ckr:global {
            ?ctx ckr:hasAssertedModule ?m .
          }
          GRAPH ckr:global {
            ?ctx cube:hasAircraft/olap:atLevel
                cube:Level_Aircraft_Model .
            ?ctx cube:hasLocation/olap:atLevel
                cube:Level_Location_Segment .
            ?ctx cube:hasDate/olap:atLevel cube:Level_Date_Day
                .
            ?ctx cube:hasImportance/olap:atLevel
                cube:Level_Importance_Importance .
          }
          {
            SELECT ?r ?gr1 ?ctx WHERE {
              GRAPH <ckr:global-inf> {
                ?inf1 ckr:closureOf ?ctx .
              }
              GRAPH ?inf1 {
                ?inf1 ckr:derivedFrom ?m1 .
              }
              GRAPH ?m1 {
                ?r obj:operationalStatus ?gr1 .
              }
            }
          }
          {
            SELECT ?r ?gr2 ?ctx WHERE {
              GRAPH <ckr:global-inf> {
                ?inf1 ckr:closureOf ?ctx .
              }
              GRAPH ?inf1 {
                ?inf1 ckr:derivedFrom ?m1 .
              }
              GRAPH ?m1 {
                ?r obj:usage/obj:usageType ?gr2 .
```

```
              }
            }
          }
          BIND(IRI(CONCAT(STR(obj:), SHA512(CONCAT(STR(?gr1), STR(?gr2))))) AS ?gr)
        }
      }
      BIND(?m AS ?g)
      BIND(?r AS ?s)
      BIND(obj:grouping AS ?p)
      BIND(?gr AS ?o)
      BIND("+" AS ?op)
    }
  } UNION {
    SELECT ?g ?s ?p ?o ?op ?ctx WHERE {
      {
        SELECT ?m ?a ?b ?c ?d ?e ?op ?ctx WHERE {
          {
            SELECT ?m ?a ?x ?c ?d ?y ?ctx WHERE {
              GRAPH ckr:global {
                ?ctx ckr:hasAssertedModule ?m .
              }
              GRAPH ckr:global {
                ?ctx cube:hasAircraft/olap:atLevel cube:Level_Aircraft_Model .
                ?ctx cube:hasLocation/olap:atLevel cube:Level_Location_Segment .
                ?ctx cube:hasDate/olap:atLevel cube:Level_Date_Day .
                ?ctx cube:hasImportance/olap:atLevel cube:Level_Importance_Importance .
              }
              GRAPH ?m {
                ?a ?c ?d .
                FILTER(?c != rdf:type)
                FILTER(?c != obj:grouping)
              }
              OPTIONAL {
                SELECT ?a ?x ?ctx WHERE {
                  {
                    SELECT ?a ?x1 ?ctx WHERE {
                      GRAPH <ckr:global-inf> {
                        ?inf1 ckr:closureOf ?ctx .
                      }
                      GRAPH ?inf1 {
                        ?inf1 ckr:derivedFrom ?m1 .
                      }
                      GRAPH ?m1 {
                        ?a obj:operationalStatus ?x1 .
                      }
                    }
                  }
                  {
                    SELECT ?a ?x2 ?ctx WHERE {
```

```
            GRAPH <ckr:global-inf> {
              ?inf1 ckr:closureOf ?ctx .
            }
            GRAPH ?inf1 {
              ?inf1 ckr:derivedFrom ?m1 .
            }
            GRAPH ?m1 {
              ?a obj:usage/obj:usageType ?x2 .
            }
          }
        }
        BIND(IRI(CONCAT(STR(obj:),
            SHA512(CONCAT(STR(?x1), STR(?x2))))) AS
            ?x)
      }
    }
    OPTIONAL {
      SELECT ?d ?y ?ctx  WHERE {
        {
          SELECT ?d ?y1 ?ctx WHERE {
            GRAPH <ckr:global-inf> {
              ?inf3 ckr:closureOf ?ctx .
            }
            GRAPH ?inf3 {
              ?inf3 ckr:derivedFrom ?m3 .
            }
            GRAPH ?m3 {
              ?d obj:operationalStatus ?y1 .
            }
          }
        }
        {
          SELECT ?d ?y2 ?ctx WHERE {
            GRAPH <ckr:global-inf> {
              ?inf3 ckr:closureOf ?ctx .
            }
            GRAPH ?inf3 {
              ?inf3 ckr:derivedFrom ?m3 .
            }
            GRAPH ?m3 {
              ?d obj:usage/obj:usageType ?y2 .
            }
          }
        }
        BIND(IRI(CONCAT(STR(obj:),
            SHA512(CONCAT(STR(?y1), STR(?y2))))) AS
            ?y)
      }
    }
  }
}
FILTER(BOUND(?x) || BOUND(?y))
BIND(IF(!BOUND(?x), ?a, ?x) AS ?b)
BIND(IF(!BOUND(?y), ?d, ?y) AS ?e)
```

```
          VALUES ?op { "-" "+" }
        }
      }
      BIND(?m AS ?g)
      BIND(IF(?op = "-", ?a, ?b) AS ?s)
      BIND(?c AS ?p)
      BIND(IF(?op = "-", ?d, ?e) AS ?o)
    }
  }
}
```

4.5.5 Value-generating abstraction

Value-generating abstraction is an aggregation of data property values from the same individual. Listing 15 shows a SPARQL query for calculating the average wingspan for each AircraftCharacteristic individual. All wingspan data properties from the same individual are grouped together (Lines 23-25) and the average aggregaton function applied (Lines 27-37).

Listing 15: SPARQL code for a value-generating abstraction

```
1  PREFIX owl: <http://www.w3.org/2002/07/owl#>
2  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3  PREFIX obj: <http://example.org/kgolap/object-model#>
4  PREFIX xml: <http://www.w3.org/XML/1998/namespace>
5  PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
6  PREFIX olap: <http://dkm.fbk.eu/ckr/olap-model#>
7  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
8  PREFIX cube: <http://example.org/kgolap/cube-model#>
9  PREFIX ckr: <http://dkm.fbk.eu/ckr/meta#>
10 PREFIX onto: <http://www.ontotext.com/>
11
12 SELECT DISTINCT ?g ?s ?p ?o ?op WHERE {
13   {
14     SELECT ?m ?s ?d ?e ?op ?ctx WHERE {
15       {
16         SELECT ?m ?s ?d ?e ?ctx WHERE {
17           GRAPH <ckr:global-inf> {
18             ?inf ckr:closureOf ?ctx .
19           }
20           GRAPH ?inf {
21             ?inf ckr:derivedFrom ?m .
22           }
23           GRAPH ?m {
24             ?s obj:wingspan ?d .
25           }
26           {
27             SELECT ?s (AVG(?d) AS ?e) ?ctx WHERE {
28               GRAPH <ckr:global-inf> {
29                 ?inf ckr:closureOf ?ctx .
30               }
31               GRAPH ?inf1 {
32                 ?inf ckr:derivedFrom ?m .
```

```
              }
              GRAPH ?m {
                ?s obj:wingspan ?d .
              }
            } GROUP BY ?s ?ctx
          }
          {
            SELECT ?s ?ctx WHERE {
              GRAPH <ckr:global-inf> {
                ?inf ckr:closureOf ?ctx .
              }
              GRAPH ?inf {
                ?inf ckr:derivedFrom ?m .
              }
              GRAPH ?m {
                ?s rdf:type obj:AircraftCharacteristic .
              }
            }
          }
        }
      }
      VALUES ?op { "-" "+" }
    }
  }
  GRAPH ckr:global {
    ?ctx ckr:hasAssertedModule ?n .
  }
  BIND(IF(?op = "-", ?m, ?n) AS ?g)
  BIND(obj:wingspan AS ?p)
  BIND(IF(?op = "-", ?d, ?e) AS ?o)
  {
    SELECT ?ctx WHERE {
      ?ctx cube:hasAircraft/olap:atLevel
          cube:Level_Aircraft_Model .
      ?ctx cube:hasLocation/olap:atLevel
          cube:Level_Location_Segment .
      ?ctx cube:hasDate/olap:atLevel cube:Level_Date_Day .
      ?ctx cube:hasImportance/olap:atLevel
          cube:Level_Importance_Importance .
    }
  }
}
```

4.5.6 Reification

Listing 16 shows the SPARQL query for reifying the usage predicate in all contexts at aircraft model, location segment, and day granularity. We use the SPARQL `UUID` function to generate a unique IRI for the reification individuals.

Listing 16: SPARQL code for a reification operation

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
PREFIX obj: <http://example.org/kgolap/object-model#>
PREFIX xml: <http://www.w3.org/XML/1998/namespace>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX olap: <http://dkm.fbk.eu/ckr/olap-model#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX cube: <http://example.org/kgolap/cube-model#>
PREFIX ckr: <http://dkm.fbk.eu/ckr/meta#>
PREFIX onto: <http://www.ontotext.com/>

SELECT ?g ?s ?p ?o ?op WHERE {
  {
    {
      BIND(?m AS ?g)
      BIND(?x AS ?s)
      BIND(rdf:subject AS ?p)
      BIND(?subj AS ?o)
      BIND("+" AS ?op)
    } UNION {
      BIND(?m AS ?g)
      BIND(?x AS ?s)
      BIND(rdf:type AS ?p)
      BIND(IRI(CONCAT(STR(?pred), '-type')) AS ?o)
      BIND("+" AS ?op)
    } UNION {
      BIND(?m AS ?g)
      BIND(?x AS ?s)
      BIND(rdf:object AS ?p)
      BIND(?obj AS ?o)
      BIND("+" AS ?op)
    }
  }
  {
    SELECT ?m ?x ?subj ?pred ?obj WHERE {
      {
        SELECT ?ctx ?m WHERE {
          ?ctx ckr:hasAssertedModule ?m .
          ?ctx cube:hasAircraft ?x1 .
          ?x1 olap:atLevel cube:Level_Aircraft_Model .
          ?ctx cube:hasLocation ?x2 .
          ?x2 olap:atLevel cube:Level_Location_Segment .
          ?ctx cube:hasImportance ?x3 .
          ?x3 olap:atLevel cube:Level_Importance_Importance .
          ?ctx cube:hasDate ?x4 .
          ?x4 olap:atLevel cube:Level_Date_Day .
        }
      }
      {
        SELECT ?ctx ?m ?subj ?pred ?obj WHERE {
          GRAPH ckr:global {
            ?ctx ckr:hasAssertedModule ?m .
          }
          GRAPH ?m {
            ?subj ?pred ?obj .
            FILTER(?pred = obj:usage)
```

```
          }
        }
      }
      BIND(UUID() AS ?x)
    }
  }
}
```

4.5.7 Pivot

Listing 17 shows the SPARQL `SELECT` statement that computes the delta table for an application of the pivoting operation on cells at aircraft model, location segment, and day granularity. Unlike the formalization, this query performs pivoting on multiple contexts and attaches a property to an individual in a specific context only if the individual is part of a triple in that particular context.

Listing 17: SPARQL code for a pivot operation

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX obj: <http://example.org/kgolap/object-model#>
PREFIX xml: <http://www.w3.org/XML/1998/namespace>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX olap: <http://dkm.fbk.eu/ckr/olap-model#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX cube: <http://example.org/kgolap/cube-model#>
PREFIX ckr: <http://dkm.fbk.eu/ckr/meta#>

SELECT ?g ?s ?p ?o ?op WHERE {
  {
    SELECT ?m ?r ?x WHERE {
      {
        SELECT ?m ?r ?x WHERE {
          GRAPH ckr:global {
            ?ctx ckr:hasAssertedModule ?m .
            ?ctx cube:hasLocation ?x .
            ?ctx cube:hasAircraft ?x1 .
            ?x1 olap:atLevel cube:Level_Aircraft_Model .
            ?ctx cube:hasLocation ?x2 .
            ?x2 olap:atLevel cube:Level_Location_Segment .
            ?ctx cube:hasDate ?x3 .
            ?x3 olap:atLevel cube:Level_Date_Day .
            ?ctx cube:hasImportance ?x4 .
            ?x4 olap:atLevel cube:Level_Importance_Importance .
          }
          GRAPH <ckr:global-inf> {
            ?inf ckr:closureOf ?ctx .
          }
          GRAPH ?inf {
            ?inf ckr:derivedFrom ?d .
          }
          GRAPH ?d {
            ?r rdf:type obj:ManoeuvringAreaAvailability .
```

```
          }
        }
      }
      {
        GRAPH ?m {
          ?r ?p ?o .
        }
      } UNION {
        GRAPH ?m {
          ?o ?p ?r .
        }
      }
    }
  }
  BIND(?m AS ?g)
  BIND(?r AS ?s)
  BIND(obj:hasLocation AS ?p)
  BIND(?x AS ?o)
  BIND("+" AS ?op)
}
```

5 Performance evaluation

We investigate feasibility by measuring performance of queries on datasets with varying properties. Even though the proof-of-concept implementation is not tweaked for performance, the experiments demonstrate feasibility of the approach.

5.1 System configuration

We conducted performance experiments on a virtual CentOS 6.8 machine (using openVZ[6]) with 128 GB main memory using four cores of an Intel Xeon CPU E5-2640 v4 machine with 2.4 GHz. The KG-OLAP system employed a GraphDB Free[7] 8.9 instance. Listing 18 shows the configuration (see [9] for more information) of the temporary repository, which was the same for the base repository. In summary, indexes were enabled while inferencing was disabled since RDFpro took care of reasoning.

Listing 18: GraphDB repository configuration

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix rep: <http://www.openrdf.org/config/repository#>.
@prefix sr: <http://www.openrdf.org/config/repository/sail#>.
@prefix sail: <http://www.openrdf.org/config/sail#>.
@prefix owlim: <http://www.ontotext.com/trree/owlim#>.

[] a rep:Repository ;
  rep:repositoryID "Temp" ;
  rdfs:label "" ;
  rep:repositoryImpl [
    rep:repositoryType "graphdb:FreeSailRepository" ;
```

[6] https://openvz.org/
[7] http://graphdb.ontotext.com/

```
    sr:sailImpl [
      sail:sailType "graphdb:FreeSail" ;

      owlim:base-URL "http://dkm.fbk.eu/ckr/meta#" ;
      owlim:defaultNS "" ;
      owlim:entity-index-size "30000000" ;
      owlim:entity-id-size  "32" ;
      owlim:imports "" ;
      owlim:repository-type "file-repository" ;
      owlim:ruleset "empty" ;
      owlim:storage-folder "storage" ;

      owlim:enable-context-index "true" ;

      owlim:enablePredicateList "true" ;

      owlim:in-memory-literal-properties "true" ;
      owlim:enable-literal-index "true" ;

      owlim:check-for-inconsistencies "false" ;
      owlim:disable-sameAs  "true" ;
      owlim:query-timeout  "0" ;
      owlim:query-limit-results  "0" ;
      owlim:throw-QueryEvaluationException-on-timeout "false" ;
      owlim:read-only "false" ;
    ]
  ].
```

5.2 Datasets

The datasets vary with respect to the number of dimensions (3D and 4D), cells/contexts (1365, 2501, and 3906 contexts), and statements (10-35 millions). The three-dimensional datasets have aircraft, location, and date dimensions, the four-dimensional datasets have an additional importance dimension. The contexts are situated at five granularities (plus the root context), with more contexts towards the finer granularities. Table 6 shows the main characteristics of the different datasets employed for performance testing. Table 7 shows the main characteristics of the different datasets employed for performance testing of rule evaluation with domain/range reasoning.

The datasets are generated in the spirit of the air traffic management (ATM) use case [10, 11]. Figure 3 shows the context structure of the sample datasets. The three-dimensional datasets lack the importance dimension. We start with a root context at the all-granularity for each dimension and recursively add descendants at different granularities. The root context has children at region granularity (and all-importance). For large context size the root context has five children, four otherwise. Each context at region granularity then has children at region-year granularity (and package importance). For large context size, each region-granularity context has five children, four otherwise. Each context at region-year granularity has children at segment-month-type granularity (and package importance). For large and medium context size, each context at region-year granularity has five children, four otherwise. Each context at segment-month-type granularity has children at segment-day-type granularity (and importance importance). For large and medium context size, each context at segment-month-type granularity has five children, four otherwise. Each context at segment-day-type granularity has children at segment-day-model granularity (and importance importance). For large and medium context size, each context at segment-day-type granularity has five children, four otherwise. The baseline datasets are derived from the three-dimensional datasets with large context size.

The contexts at different granularities comprise different types of knowledge. The root context consists mainly of terminological knowledge defining domain and range of properties as well as subclass relationships (Listing 19). In general, the terminological knowledge defines classes and properties describing airport infrastructure as well as messages related to availability of infrastructure and contamination thereof. The root context also defines individuals of the ContaminationType class as well as contaminantGrouping relationships between these individuals, i.e., a contamination type may be the grouping of another contamination type. The contamination types and groupings are referred to in the lower-level contexts.

The region contexts define Airport individuals as well as Runway and Taxiway individuals for each airport; the runways and taxiways are linked to an airport via the isSituatedAt property. The lower-level context then define additional knowledge about the runways and taxiways. The region contexts also define VOR (Very High Frequency Omni-Directional Range) representing a type of navigational aid, which have a randomly assigned longitude and latitude data property. Then, the region-year contexts define randomly assigned frequency data property values to the previously defined VOR individuals.

The segment-month-type contexts define ManoeuvringAreaAvailability individuals and link those individuals to the previously defined taxiways, linked to the Taxiway individuals via availability property. Each of the ManoeuvringAreaAvailability individuals has warning property referring to a Warning individual and a randomly assigned warningAdjacent boolean data property value, which indicates a warning either on or adjacent to the taxiway. In addition, the segment-month-type contexts have randomly generated triples that serve as “fillers” in order to create a repository with a larger number of statements for performance evaluation purposes. These “filler” triples also constitutes a graph of interlinked individuals.

The segment-day-type contexts define SurfaceContamination individuals and link those individuals to the previously defined runways and taxiways, linked to the Runway and Taxiway individuals via contaminant property. Each of the SurfaceContamination individuals has a depth data property and three layer properties linking to SurfaceContaminationLayer individuals. Each of the SurfaceContaminationLayer individuals, in turn, has a contaminationType property linking to a ContaminationType individual defined in the root context.

The segment-day-model contexts define ManoeuvringAreaAvailability individuals and link those individuals to the previously defined runways and taxiways. Each of the ManoeuvringAreaAvailability individuals has an operationalStatus and a usage property linking to a Status and ManoeuvringAreaUsage individual, respectively. Each of the ManoeuvringAreaUsage individuals has a usageType, an operation, and an aircraft property linking to a UsageType, Operation, and AircraftCharacteristic individual, respectively. The AircraftCharacteristic individuals either have a wingspan data property and a obj:wingspanInterpretation property linking to an Interpretation individual or a weight data property and a weightInterpretation property. The segment-day-model contexts also have “filler” triples.

Listing 19: Terminological knowledge in the root context

```
obj:Airport rdfs:subClassOf obj:AirportHeliport .

obj:Runway rdfs:subClassOf obj:RunwayTaxiway .
obj:Taxiway rdfs:subClassOf obj:RunwayTaxiway .

obj:isSituatedAt rdfs:range obj:AirportHeliport .

obj:availability rdfs:range obj:ManoeuvringAreaAvailability .

obj:warning rdfs:domain obj:ManoeuvringAreaAvailability .

obj:warningAdjacent rdfs:domain
    obj:ManoeuvringAreaAvailability .
obj:warningAdjacent rdfs:range xsd:boolean .

obj:operationalStatus rdfs:domain
    obj:ManoeuvringAreaAvailability .

obj:usage rdfs:domain obj:ManoeuvringAreaAvailability .
obj:usage rdfs:range obj:ManoeuvringAreaUsage .

obj:usageType rdfs:domain obj:ManoeuvringAreaUsage .

obj:operation rdfs:domain obj:ManoeuvringAreaUsage .

obj:aircraft rdfs:domain obj:ManoeuvringAreaUsage .
obj:aircraft rdfs:range obj:AircraftCharacteristic .

obj:weight rdfs:domain obj:AircraftCharacteristic .

obj:weightInterpretation rdfs:domain
    obj:AircraftCharacteristic .

obj:wingspan rdfs:domain obj:AircraftCharacteristic .

obj:wingspanInterpretation rdfs:domain
    obj:AircraftCharacteristic .

obj:contaminant rdfs:range obj:SurfaceContamination .

obj:depth rdfs:domain obj:SurfaceContamination .

obj:layer rdfs:domain obj:SurfaceContamination .
obj:layer rdfs:range obj:SurfaceContaminationLayer .

obj:contaminationType rdfs:domain
    obj:SurfaceContaminationLayer .

obj:frequency rdfs:domain obj:VOR .
```

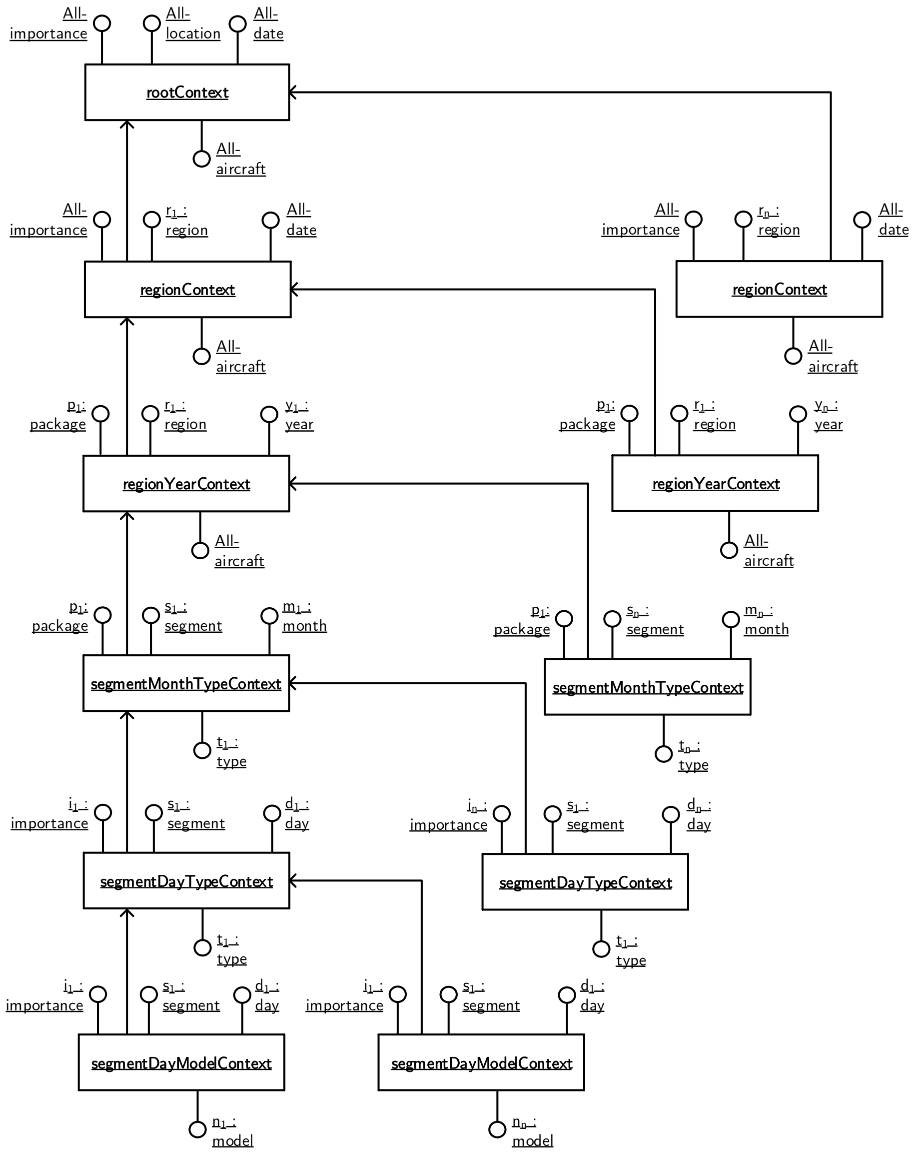

Figure 3: Contexts in the sample datasets

Table 6: Characteristics of the datasets employed in the performance experiments

Dataset	Contexts	Asserted Statements	Total Statements
3D – Small/Small	1 365	10 281 383	10 651 129
3D – Small/Small (Abstracted)	1 365	9 847 207	10 216 953
3D – Small/Medium	1 365	18 859 335	19 511 429
3D – Small/Medium (Abstracted)	1 365	18 048 327	18 700 421
3D - Small/Large	1 365	26 397 895	27 295 509
3D - Small/Large (Abstracted)	1 365	25 259 207	26 156 821
3D – Medium/Small	2 501	11 236 343	11 666 413
3D – Medium/Small (Abstracted)	2 501	10 740 343	11 170 413
3D – Medium/Medium	2 501	21 702 967	22 448 037
3D – Medium/Medium (Abstracted)	2 501	20 758 967	21 504 037
3D – Medium/Large	2 501	31 732 791	32 792 861
3D – Medium/Large (Abstracted)	2 501	30 340 791	31 400 861
3D – Large/Small	3 906	11 327 542	11 788 420
3D – Large/Small (Abstracted)	3 906	10 852 542	11 313 420
3D – Large/Medium	3 906	22 618 792	23 431 120
3D – Large/Medium (Abstracted)	3 906	21 643 792	22 456 120
3D – Large/Large	3 906	33 910 042	35 073 820
3D – Large/Large (Abstracted)	3 906	32 435 041	33 598 819
4D – Small/Small	1 365	10 283 854	10 656 594
4D – Small/Small (Abstracted)	1 365	9 849 678	10 222 418
4D – Small/Medium	1 365	18 861 806	19 516 894
4D – Small/Medium (Abstracted)	1 365	18 050 798	18 705 886
4D - Small/Large	1 365	26 400 366	27 300 974
4D - Small/Large (Abstracted)	1 365	25 261 677	26 162 285
4D – Medium/Small	2 501	11 240 526	11 675 590
4D – Medium/Small (Abstracted)	2 501	10 744 526	11 179 590
4D – Medium/Medium	2 501	21 707 150	22 457 214
4D – Medium/Medium (Abstracted)	2 501	20 763 150	21 513 214
4D – Medium/Large	2 501	31 736 974	32 802 038
4D – Medium/Large (Abstracted)	2 501	30 344 974	31 410 038
4D – Large/Small	3 906	11 334 066	11 802 738
4D – Large/Small (Abstracted)	3 906	10 859 066	11 327 738
4D – Large/Medium	3 906	22 625 316	23 445 438
4D – Large/Medium (Abstracted)	3 906	21 650 316	22 470 438
4D – Large/Large	3 906	33 916 566	35 088 138
4D – Large/Large (Abstracted)	3 906	32 441 566	33 613 138
Baseline Small	1	11 291 397	11 291 905
Baseline Small (Abstracted)	1	10 816 397	10 816 905
Baseline Medium	1	22 582 647	22 583 605
Baseline Medium (Abstracted)	1	21 607 647	21 608 605
Baseline Large	1	33 873 897	33 875 305
Baseline Large (Abstracted)	1	32 398 897	32 400 305

Table 7: Characteristics of the datasets employed in the performance experiments for rule evaluation with domain/range reasoning

Dataset	Contexts	Asserted Statements	Total Statements
3D – Small/Small (Domain/Range)	1 365	6 199 259	10 273 483
3D – Small/Medium (Domain/Range)	1 365	6 844 003	15 639 991
3D - Small/Large (Domain/Range)	1 365	6 859 522	24 640 242
3D – Medium/Small (Domain/Range)	2 501	11 388 515	17 312 399
3D – Medium/Medium (Domain/Range)	2 501	13 124 147	24 079 999
3D – Medium/Large (Domain/Range)	2 501	13 682 772	33 874 042
3D – Large/Small (Domain/Range)	3 906	15 938 755	23 471 039
3D – Large/Medium (Domain/Range)	3 906	19 196 291	32 312 007
3D – Large/Large (Domain/Range)	3 906	20 506 022	43 107 842
4D – Small/Small (Domain/Range)	1 365	6 201 730	10 280 314
4D – Small/Medium (Domain/Range)	1 365	11 390 986	17 319 230
4D - Small/Large (Domain/Range)	1 365	15 941 226	23 477 870
4D – Medium/Small (Domain/Range)	2 501	6 848 186	15 651 670
4D – Medium/Medium (Domain/Range)	2 501	13 128 330	24 091 678
4D – Medium/Large (Domain/Range)	2 501	19 200 474	32 323 686
4D – Large/Small (Domain/Range)	3 906	6 866 046	24 658 467
4D – Large/Medium (Domain/Range)	3 906	13 689 296	33 892 267
4D – Large/Large (Domain/Range)	3 906	20 512 546	43 126 067

5.3 Queries

We evaluate performance of slice and dice, merge union, replacement by grouping (triple-generating abstraction), grouping by properties (individual-generating abstraction), aggregation of data property values (value-generating abstraction), reification, and pivoting; we also investigate performance of rule evaluation. The queries correspond to the example queries in the previous sections. For performance evaluation, we keep the delta statements produced by the query operations in memory – rather than storing them on disk – and then perform bulk deletions and insertions. We note that bulk updates with delta files are a common-place performance optimization technique: "High performance bulk-revision of existing data (...) is best achieved by finding the difference (the 'delta') between an existing graph or dataset and the new graph or dataset being loaded, and then applying that differential or 'graph delta' to the quad store" [12].

5.4 Results

Results demonstrate general feasibility of the approach. In the following, Tables 8-14 give detailed results for the run times of the different query operations on various datasets; Table 15 gives results of rule evaluation. We give median run time and mean run time as well as standard deviation (SD) and standard error (SE) of the mean over N iterations. Results for the queries are for computation of delta tables only and results for rule evaluation are without loading statements and writing the statements back to the repository since such write operations are not specific to KG-OLAP and may vary between quad stores. We note that performing the actual updates can take a couple of minutes. In case of the merge union operation on the largest dataset, for example, insertion of approximately 35 million statements takes about 20 minutes, deleting approximately 35 million statements takes about 15 minutes. For the smallest dataset, insertion of approximately 10 million statements takes about five minutes, deleting approximately 10 million statements takes about four minutes.

Figure 4 illustrates the results of performance experiments related to the reification operation. The run times for the reification operations grow linearly with the repository size. For the pivoting operation (Figure 5), there was an important influence of context size. In particular, there was a stark difference between performing the pivoting operation on a single context (the baseline datasets) and performing the pivoting operation on all contexts at a specific level.

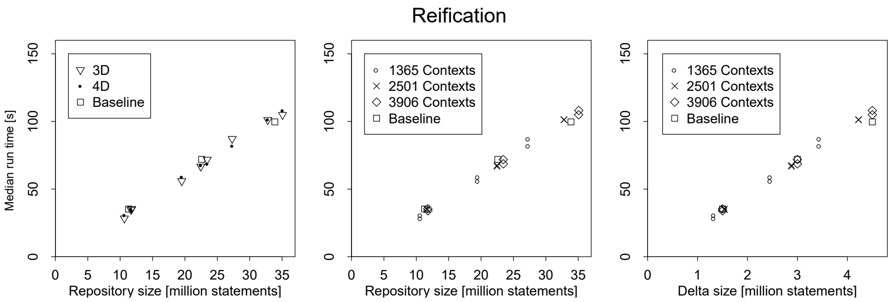

Figure 4: Performance of reification

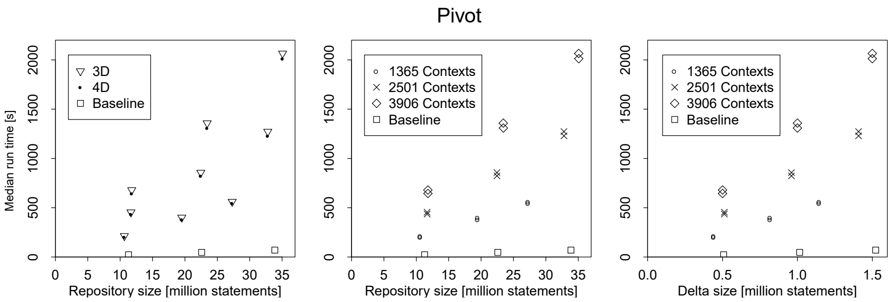

Figure 5: Performance of pivoting

Table 8: Run time in seconds for the computation of delta tables for slice/dice

Dataset	Median	Mean	SD	N	SE
3D – Small/Small	21.82	21.72	0.70	15	0.18
3D – Small/Medium	40.26	40.25	0.78	15	0.20
3D – Small/Large	56.38	56.04	1.16	15	0.30
3D – Medium/Small	24.19	24.37	1.06	15	0.27
3D – Medium/Medium	47.25	47.02	0.68	15	0.17
3D – Medium/Large	68.39	76.00	12.22	15	3.15
3D – Large/Small	18.16	18.27	0.39	15	0.10
3D – Large/Medium	39.60	39.52	0.83	15	0.21
3D – Large/Large	58.47	58.40	1.13	15	0.29
4D – Small/Small	22.40	22.36	0.44	15	0.11
4D – Small/Medium	41.11	41.05	0.93	15	0.24
4D – Small/Large	56.43	56.64	0.82	15	0.21
4D – Medium/Small	24.27	24.36	0.67	15	0.17
4D – Medium/Medium	47.30	47.64	1.47	15	0.38
4D – Medium/Large	68.07	69.54	6.61	15	1.70
4D – Large/Small	18.62	18.63	0.43	15	0.11
4D – Large/Medium	39.55	39.91	1.13	15	0.29
4D – Large/Large	58.66	58.85	1.16	15	0.30

Table 9: Run time in seconds for the computation of delta tables for merge union

Dataset	Median	Mean	SD	N	SE
3D – Small/Small	301.15	300.40	5.68	15	1.47
3D – Small/Medium	520.40	522.06	6.20	15	1.60
3D – Small/Large	724.95	728.78	9.33	15	2.41
3D – Medium/Small	433.37	431.35	7.02	17	1.70
3D – Medium/Medium	709.38	702.21	13.81	15	3.56
3D – Medium/Large	952.76	956.38	12.54	15	3.24
3D – Large/Small	698.63	699.00	5.39	15	1.39
3D – Large/Medium	1032.83	1030.70	9.93	15	2.56
3D – Large/Large	1356.34	1354.58	8.38	15	2.16
4D – Small/Small	327.98	325.60	5.15	15	1.33
4D – Small/Medium	548.56	551.24	7.56	15	1.95
4D – Small/Large	763.32	765.36	9.81	15	2.53
4D – Medium/Small	468.17	470.27	7.11	15	1.84
4D – Medium/Medium	772.17	769.03	12.06	15	3.11
4D – Medium/Large	1022.70	1024.79	9.40	15	2.43
4D – Large/Small	828.39	828.51	4.00	15	1.03
4D – Large/Medium	1175.04	1177.32	9.01	15	2.33
4D – Large/Large	1517.09	1515.73	12.07	15	3.12

Table 10: Run time in seconds for the computation of delta tables for triple-generating abstraction

Dataset	Median	Mean	SD	N	SE
3D – Small/Small	224.11	224.51	9.83	15	2.54
3D – Small/Medium	423.53	424.30	9.00	15	2.32
3D – Small/Large	591.83	593.03	9.77	15	2.52
3D – Medium/Small	262.16	262.27	5.57	15	1.44
3D – Medium/Medium	505.19	505.35	17.28	15	4.46
3D – Medium/Large	737.02	736.29	24.44	15	6.31
3D – Large/Small	259.27	260.97	10.10	15	2.61
3D – Large/Medium	507.04	510.75	16.26	15	4.20
3D – Large/Large	740.36	749.63	24.80	15	6.27
4D – Small/Small	219.44	218.94	4.25	15	1.10
4D – Small/Medium	430.28	431.81	11.28	15	2.91
4D – Small/Large	629.25	630.08	15.83	15	4.08
4D – Medium/Small	262.05	265.05	6.04	15	1.56
4D – Medium/Medium	507.02	509.23	13.43	15	3.47
4D – Medium/Large	741.71	743.81	19.19	15	4.95
4D – Large/Small	264.96	265.33	9.64	15	2.49
4D – Large/Medium	511.39	516.21	15.28	15	3.94
4D – Large/Large	748.45	752.92	13.90	15	3.59
Baseline Small	276.76	275.31	8.72	15	2.25
Baseline Medium	549.89	549.83	19.23	15	4.97
Baseline Large	793.24	795.20	30.42	16	7.61

Table 11: Run time in seconds for the computation of delta tables for individual-generating abstraction

Dataset	Median	Mean	SD	N	SE
3D – Small/Small	270.23	273.09	5.16	15	1.33
3D – Small/Medium	526.16	527.48	7.22	15	1.86
3D – Small/Large	744.89	746.52	8.01	15	2.07
3D – Medium/Small	323.44	326.44	7.12	15	1.84
3D – Medium/Medium	610.58	611.55	9.14	15	2.36
3D – Medium/Large	888.77	889.94	5.48	15	1.41
3D – Large/Small	313.22	314.35	5.84	15	1.51
3D – Large/Medium	626.65	627.32	8.90	15	2.30
3D – Large/Large	938.89	940.95	15.53	15	4.01
4D – Small/Small	290.99	289.00	6.48	15	1.67
4D – Small/Medium	548.03	547.95	5.95	15	1.54
4D – Small/Large	752.01	752.78	8.46	15	2.18
4D – Medium/Small	327.40	327.94	5.07	15	1.31
4D – Medium/Medium	622.44	626.77	15.11	15	3.90
4D – Medium/Large	906.97	909.39	14.98	15	3.87
4D – Large/Small	320.41	323.20	6.34	15	1.64
4D – Large/Medium	621.34	627.26	15.85	15	4.09
4D – Large/Large	928.77	930.30	11.95	15	3.09
Baseline Small	304.39	301.64	11.27	15	2.91
Baseline Medium	627.05	633.64	11.74	15	3.03
Baseline Large	957.64	956.76	10.53	15	2.72

Table 12: Run time in seconds for the computation of delta tables for value-generating abstraction

Dataset	Median	Mean	SD	N	SE
3D – Small/Small (Abstracted)	36.12	36.11	0.46	15	0.12
3D – Small/Medium (Abstracted)	67.77	70.28	4.86	15	1.26
3D – Small/Large (Abstracted)	89.44	92.97	6.72	15	1.73
3D – Medium/Small (Abstracted)	48.61	50.32	3.05	15	0.79
3D – Medium/Medium (Abstracted)	80.84	82.65	4.22	15	1.09
3D – Medium/Large (Abstracted)	115.94	120.54	9.28	15	2.40
3D – Large/Small (Abstracted)	58.95	59.81	2.92	15	0.75
3D – Large/Medium (Abstracted)	95.72	97.53	5.56	15	1.43
3D – Large/Large (Abstracted)	133.73	134.07	2.06	15	0.53
4D – Small/Small (Abstracted)	37.95	38.00	1.25	15	0.32
4D – Small/Medium (Abstracted)	74.80	75.24	2.85	15	0.74
4D – Small/Large (Abstracted)	100.52	100.84	4.63	15	1.20
4D – Medium/Small (Abstracted)	52.29	52.84	1.90	15	0.49
4D – Medium/Medium (Abstracted)	85.13	86.59	3.93	15	1.02
4D – Medium/Large (Abstracted)	121.40	121.72	3.12	15	0.81
4D – Large/Small (Abstracted)	61.95	62.08	1.88	15	0.49
4D – Large/Medium (Abstracted)	99.74	101.31	4.71	15	1.22
4D – Large/Large (Abstracted)	139.56	142.53	5.60	15	1.45
Baseline Small (Abstracted)	28.60	29.56	1.91	15	0.49
Baseline Medium (Abstracted)	60.00	61.47	4.10	15	1.06
Baseline Large (Abstracted)	92.82	94.77	5.85	15	1.51

Table 13: Run time in seconds for the computation of delta tables for reification

Dataset	Median	Mean	SD	N	SE
3D – Small/Small	28.40	28.74	1.45	15	0.37
3D – Small/Medium	56.00	57.59	3.38	15	0.87
3D – Small/Large	87.18	86.55	3.14	15	0.81
3D – Medium/Small	34.83	34.99	0.77	15	0.20
3D – Medium/Medium	66.76	69.46	4.37	15	1.13
3D – Medium/Large	101.15	101.71	3.99	15	1.03
3D – Large/Small	35.42	35.42	0.72	15	0.19
3D – Large/Medium	71.82	72.05	1.82	15	0.47
3D – Large/Large	104.93	105.68	3.89	15	1.00
4D – Small/Small	30.75	30.35	1.39	15	0.36
4D – Small/Medium	58.95	58.26	1.69	15	0.44
4D – Small/Large	81.94	82.25	1.45	15	0.37
4D – Medium/Small	35.23	35.23	0.80	15	0.21
4D – Medium/Medium	67.57	69.88	9.00	15	2.32
4D – Medium/Large	101.39	102.40	4.31	15	1.11
4D – Large/Small	33.98	33.67	0.96	15	0.25
4D – Large/Medium	68.66	68.56	1.79	16	0.45
4D – Large/Large	108.07	109.97	7.32	16	1.83
Baseline Small	35.27	35.55	1.22	15	0.31
Baseline Medium	71.98	72.18	2.92	15	0.75
Baseline Large	99.64	100.17	2.13	15	0.55

Table 14: Run time in seconds for the computation of delta tables for pivoting

Dataset	Median	Mean	SD	N	SE
3D – Small/Small	214.13	214.36	4.52	15	1.17
3D – Small/Medium	401.04	401.58	6.20	15	1.60
3D – Small/Large	561.73	558.60	6.97	15	1.80
3D – Medium/Small	454.53	455.37	6.09	15	1.57
3D – Medium/Medium	856.51	855.60	8.80	15	2.27
3D – Medium/Large	1272.57	1272.90	10.96	15	2.83
3D – Large/Small	679.77	649.63	11.90	15	3.07
3D – Large/Medium	1358.35	1358.76	13.47	15	3.48
3D – Large/Large	2065.26	2064.63	34.59	15	8.93
4D – Small/Small	202.97	203.34	1.73	15	0.45
4D – Small/Medium	381.54	381.67	2.59	15	0.67
4D – Small/Large	546.27	546.37	4.00	15	1.03
4D – Medium/Small	435.52	435.21	2.98	15	0.77
4D – Medium/Medium	823.48	824.23	4.29	15	1.11
4D – Medium/Large	1229.32	1228.71	10.88	15	2.81
4D – Large/Small	644.55	644.13	2.39	15	0.62
4D – Large/Medium	1308.71	1308.69	8.48	15	2.19
4D – Large/Large	2012.39	2016.94	22.88	15	5.91
Baseline Small	22.85	23.10	0.60	15	0.16
Baseline Medium	47.02	46.93	1.18	15	0.31
Baseline Large	69.98	70.01	2.63	15	0.68

Table 15: Run time in seconds for evaluating rules with membership reasoning under consideration of subclass relationships

Dataset	Median	Mean	SD	N	SE
3D – Small/Small	171.66	171.34	5.73	11	1.73
3D – Small/Medium	185.00	183.37	9.99	11	3.01
3D – Small/Large	183.36	185.19	6.14	11	1.85
3D – Medium/Small	686.43	696.13	60.61	12	17.50
3D – Medium/Medium	679.89	685.83	27.68	11	8.35
3D – Medium/Large	676.19	674.90	28.43	12	8.21
3D – Large/Small	1617.46	1638.97	81.40	11	24.54
3D – Large/Medium	1643.51	1655.20	52.65	11	15.87
3D – Large/Large	1647.26	1669.66	59.90	13	16.61
4D – Small/Small	200.20	202.21	9.14	11	2.76
4D – Small/Medium	221.57	218.95	13.92	11	4.20
4D – Small/Large	216.25	219.25	9.82	11	2.96
4D – Medium/Small	772.22	777.65	34.90	9	11.63
4D – Medium/Medium	767.23	761.55	19.16	9	6.39
4D – Medium/Large	778.78	783.54	34.36	9	11.45
4D – Large/Small	1879.00	1861.65	70.76	9	23.59
4D – Large/Medium	1855.82	1856.77	55.78	10	17.64
4D – Large/Large	1983.90	1981.09	93.96	12	27.12

Table 16: Run time in seconds for evaluating rules with membership reasoning under consideration of subclass relationships as well as domain/range

Dataset	**Median**	**Mean**	**SD**	**N**	**SE**
3D – Small/Small (Domain/Range)	487.33	481.77	22.88	10	7.23
3D – Small/Medium (Domain/Range)	523.15	523.11	36.18	10	11.44
3D – Small/Large (Domain/Range)	533.82	543.35	44.72	10	14.14
3D – Medium/Small (Domain/Range)	2151.67	2145.78	138.17	10	43.69
3D – Medium/Medium (Domain/Range)	2762.48	2750.21	184.56	10	58.36
3D – Medium/Large (Domain/Range)	2763.04	2799.56	224.07	10	70.85
3D – Large/Small (Domain/Range)	7458.16	7427.12	820.63	10	259.51
3D – Large/Medium (Domain/Range)	7806.48	7892.62	565.61	10	178.86
3D – Large/Large (Domain/Range)	8286.55	8332.46	762.81	10	241.22
4D – Small/Small (Domain/Range)	484.91	498.30	46.59	10	14.73
4D – Small/Medium (Domain/Range)	587.36	596.91	34.79	10	11.00
4D – Small/Large (Domain/Range)	558.44	568.69	51.05	10	16.15
4D – Medium/Small (Domain/Range)	2383.81	2441.62	304.05	10	96.15
4D – Medium/Medium (Domain/Range)	2760.17	2778.86	158.65	17	38.48
4D – Medium/Large (Domain/Range)	2583.83	2580.10	179.65	10	56.81
4D – Large/Small (Domain/Range)	7637.13	7763.63	829.15	10	262.20
4D – Large/Medium (Domain/Range)	7724.61	7922.17	638.20	10	201.82
4D – Large/Large (Domain/Range)	8175.12	8388.14	685.41	10	216.75

References

[1] C.G. Schuetz, L. Bozzato, B. Neumayr, M. Schrefl and L. Serafini, Knowledge Graph OLAP: A Multidimensional Model and Query Operations for Contextualized Knowledge Graphs, *Semantic Web* (2020). `http://www.semantic-web-journal.net/content/knowledge-graph-olap-multidimensional-model-and-query-operations-contextualized-knowledge-0`.

[2] F. Baader, D. Calvanese, D. McGuinness, D. Nardi and P. Patel-Schneider (eds), *The Description Logic Handbook*, Cambridge University Press, 2003.

[3] L. Bozzato and L. Serafini, Materialization calculus for contexts in the Semantic Web, in: *DL 2013*, CEUR Workshop Proceedings, Vol. 1014, CEUR-WS.org, 2013. `http://ceur-ws.org/Vol-1014/paper_51.pdf`.

[4] L. Bozzato, T. Eiter and L. Serafini, Enhancing context knowledge repositories with justifiable exceptions, *Artificial Intelligence* **257** (2018), 72–126. doi:10.1016/j.artint.2017.12.005.

[5] I. Horrocks, O. Kutz and U. Sattler, The Even More Irresistible $\mathcal{SROIQ}$, in: *Procs. of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 2006)*, AAAI Press, 2006, pp. 57–67.

[6] B. Motik, A. Fokoue, I. Horrocks, Z. Wu, C. Lutz and B.C. Grau, OWL 2 Web Ontology Language Profiles, W3C Recommendation, W3C, 2009. `http://www.w3.org/TR/2009/REC-owl2-profiles-20091027/`.

[7] M. Krötzsch, Efficient Inferencing for OWL EL, in: *JELIA 2010*, LNCS, Vol. 6341, Springer, 2010, pp. 234–246. doi:10.1007/978-3-642-15675-5_21.

[8] A. Vaisman and E. Zimányi, *Data Warehouse Systems – Design and Implementation*, Springer, Berlin Heidelberg, 2014.

[9] Ontotext, Configuring a repository, `http://graphdb.ontotext.com/documentation/8.9/free/configuring-a-repository.html` (Accessed: 20 October 2020).

[10] C.G. Schuetz, B. Neumayr, M. Schrefl, E. Gringinger and S. Wilson, Semantics-based summarisation of ATM information: Managing information overload in pilot briefings using semantic data containers, *The Aeronautical Journal* **123**(1268) (2019), 1639–1665. doi:10.1017/aer.2019.74.

[11] C.G. Schuetz, B. Neumayr, M. Schrefl, E. Gringinger, A. Vennesland and S. Wilson, The Case for Contextualized Knowledge Graphs in Air Traffic Management, in: *CKG 2018*, CEUR Workshop Proceedings, Vol. 2317, CEUR-WS.org, 2018. `http://ceur-ws.org/Vol-2317/article-10.pdf`.

[12] OpenLink Software, *Virtuoso Open-Source Wiki: Delta-aware bulk loading of datasets into Virtuoso*, `http://vos.openlinksw.com/owiki/wiki/VOS/VirtRDFBulkLoaderWithDelete` (Accessed: 20 October 2020).