Graph Rewriting System for Consistent Evolution of RDF/S databases

Jacques Chabin, Cédric Eichler, Mirian Halfeld-Ferrari, Nicolas Hiot

Graph Rewriting System for Consistent Evolution of RDF/S databases *

Jacques Chabin, Cédric Eichler, Mirian Halfeld-Ferrari, and Nicolas Hiot

Université d'Orléans, INSA CVL, LIFO EA, Orléans, France
{jacques.chabin, mirian, nicolas.hiot}@univ-orleans.fr
cedric.eichler@insa-cvl.fr

1 Introduction

Today RDF (Resource Description Framework) is a standard model for data interchange on the Web and particularly for exporting Linked Open Data. These data, enriched by constraints, stored in a database (especially in a graph database or triple store), and available for querying systems are important sources for analysis and for guiding decisions. But only systems offering the capability of dealing with evolution of data instance and structure without violating the semantics of the RDF model can ensure sustainability. Since RDF databases can be canonically viewed as a typed graph, this technical report proposes a formalization of RDF datastores evolutions through the use of *graph rewriting rules*. RDF/S is the focus of the report because, currently, users interested in these facilities are mostly those dealing with ontology evolution.

We are interested in the maintenance of *valid* RDF/S knowledge graph, *i.e.*, data sets respecting RDF/S semantic constraints. When such a data set evolves (through instance or schema changes) we have to guarantee that the new set conforms to the constraints. To do so, we propose a set of graph rewriting rules formalizing atomic RDF/S evolution that *necessarily* preserve the database integrity. The next section briefly introduces RDF/S notations and the considered constraints. Section 3 presents their representation using a typed graph. Section 4 is dedicated to our consistency-preserving graph rewriting rules formalizing atomic RDF/S updates.

2 RDF databases and updates

The RDF data model describes (web) resources via triples of the format (a P b), which express the fact that a has b as value for property P. A collection of RDF statements intrinsically represents a typed attributed directed multi-graph, making the RDF model suited to certain kinds of knowledge representation [1]. Constraints on RDF facts can be expressed in RDFS (Resource Description Framework Schema), the schema language of RDF, which allows, for instance,

* Supported by the French National Research Agency, ANR, under grant ANR-18-CE23-0010; work also developed in the context of DOING-DIAMS network group.

declaring objects and subjects as instances of certain classes or expressing semantic relations between classes and between properties (*i.e.*, subclasses and sub-properties).

In [2] we find a set of logical rules expressing the semantics of RDF/S (rules concerning RDF or RDFS) models. We consider $\mathbf{A}_C = \{a, b, \ldots, a_1, a_2, \ldots\}$, a countably infinite set of constants and $var = \{X_1, X_2, \ldots, Y_1, \ldots\}$ an infinite set of variables ranging over elements in $\mathbf{A}_C$. A *term* is a constant or a variable. We classify predicates into two sets: (i) SchPred $= \{Cl, Pr, CSub, Psub, Dom, Rng\}$, used to define the database schema, standing respectively for classes, properties, sub-classes, sub-properties, property domain and range, and (ii) InstPred $= \{CI, PI, Ind\}$, used to define the database instance, standing respectively for class and property instances and individuals. An *atom* has the form $P(u)$, where P is a predicate, and u is a list of terms. When all the terms of an atom are in $\mathbf{A}_C$, we have a fact.

Definition 1 (Database). *An RDF database $\mathscr{D}$ is a set of facts composed by two subsets: the database instance $\mathscr{D}_I$ (facts with predicates in* InstPred*) and the database schema $\mathscr{D}_S$ (facts with predicates in* SchPred*). Denote by $\mathbb{G}$ the graph that represents the same database $\mathscr{D}$. The notation $\mathscr{D}/\mathbb{G}$ designates these two formats of a database.* □

Constraints presented in [2] are those in Fig. 1 which is borrowed from [4]. These constraints (that we denote by $\mathscr{C}$) are the basis of RDF semantics. Given $c \in \mathscr{C}$ we note $body(c)$ its left-hand side and $head(c)$ its right-hand side. For instance, the schema constraint (20) establishes transitivity between sub-properties and the instance constraint (27) ensures this transitivity on instances of a property (if z is a sub-property of w, all z's instances are property instances of w). We are interested in database that satisfy all constraints in $\mathscr{C}$.

Definition 2 (Consistent database $(\mathscr{D}, \mathscr{C})$). *A database $\mathscr{D}$ is consistent if it satisfies all constraints in $\mathscr{C}$ (i.e., in this paper, those in Fig. 1).* □

At this point, it is worth noting the dichotomy which usually exists when dealing with constraints on RDF. The web semantics world mostly adopts the open world assumption (OWA) and ontological constraints are, in fact, just inference rules. The database world usually adopts the closed world assumption (CWA) and constraints impose data restrictions. Rules are not supposed to infer a non-explicit knowledge. This paper adopts the database point of view and addresses the problem of updating an RDF database.

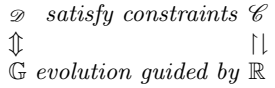

Fig. 2. Rewriting rules $\mathbb{R}$ and constraints $\mathscr{C}$.

Definition 3 (Update). *Let $\mathscr{D}/\mathbb{G}$ be a database. An update U on $\mathscr{D}$ (for $U = F$) is either (i) the insertion of F in $\mathscr{D}$ (an insertion is denoted by F) or (ii) the removal of F from $\mathscr{D}$ (a deletion is denoted by $\neg F$). To each update U corresponds a graph rewriting rule r.* □

• **Typing Constraints:**			
$Cl(x) \Rightarrow URI(x)$	(1)	$Pr(x) \Rightarrow URI(x)$	(2)
$Ind(x) \Rightarrow URI(x)$	(3)	$(Cl(x) \wedge Pr(x)) \Rightarrow \bot$	(4)
$(Cl(x) \wedge Ind(x)) \Rightarrow \bot$	(5)	$(Pr(x) \wedge Ind(x)) \Rightarrow \bot$	(6)
$CSub(x,y) \Rightarrow Cl(x) \wedge Cl(y)$	(7)	$PSub(x,y) \Rightarrow Pr(x) \wedge Pr(y)$	(8)
$Dom(x,y) \Rightarrow Pr(x) \wedge Cl(y)$	(9)	$Rng(x,y) \Rightarrow Pr(x) \wedge Cl(y)$	(10)
$CI(x,y) \Rightarrow Ind(x) \wedge Cl(y)$	(11)	$PI(x,y,z) \Rightarrow Ind(x) \wedge (Ind(y) \vee Lit(y)) \wedge Pr(z)$	(12)
$Cl(x) \Rightarrow CSub(x, rdfs{:}Resource)$	(13)	$Ind(x) \Rightarrow CI(x, rdfs{:}Resource)$	(14)
• **Schema Constraints:**			
$Pr(x) \Rightarrow (\exists y,z)(Dom(x,y) \wedge Rng(x,y))$	(15)	$((y \neq z) \wedge Dom(x,y) \wedge Dom(x,z)) \Rightarrow \bot$	(16)
$((y \neq z) \wedge Rng(x,y) \wedge Rng(x,z)) \Rightarrow \bot$	(17)		
$CSub(x,y) \wedge CSub(y,z) \Rightarrow CSub(x,z)$	(18)	$CSub(x,y) \wedge CSub(y,x) \Rightarrow \bot$	(19)
$PSub(x,y) \wedge PSub(y,z) \Rightarrow PSub(x,z)$	(20)	$PSub(x,y) \wedge PSub(y,x) \Rightarrow \bot$	(21)
$Psub(x,y) \wedge Dom(x,z) \wedge Dom(y,w) \wedge (z \neq w) \Rightarrow CSub(z,w)$			(22)
$Psub(x,y) \wedge Rng(x,z) \wedge Rng(y,w) \wedge (z \neq w) \Rightarrow CSub(z,w)$			(23)
• **Instance Constraints:**			
$Dom(z,w) \Rightarrow (PI(x,y,z) \Rightarrow CI(x,w))$	(24)	$Rng(z,w) \Rightarrow (PI(x,y,z) \Rightarrow CI(x,w))$	(25)
$CSub(y,z) \Rightarrow (CI(x,y) \Rightarrow CI(x,z))$	(26)	$PSub(z,w) \Rightarrow (PI(x,y,z) \Rightarrow PI(x,y,w))$	(27)

Fig. 1. Simplified and compacted form of RDF/S constraints

Updates can be classified according to the predicate of F, *i.e.*, the insertion (or the deletion) of a class, a class instance, a property, etc. For each update type, a rewriting rule r describes when and how to transform a graph database. This paper aims at proposing a set of graph rewriting rules, denoted by $\mathbb{R}$, which ensures consistent transformations on $\mathbb{G}$ due to any atomic update U. The set $\mathbb{R}$ is defined on the basis of $\mathscr{C}$ as illustrated in Fig. 2: on the logical level, $(\mathscr{D}, \mathscr{C})$ expresses consistent databases; on the knowledge graph level, $(\mathbb{G}, \mathbb{R})$ expresses graph evolution with rules in $\mathbb{R}$ encompassing constraints from $\mathscr{C}$. The idea is: given $\mathscr{D}/\mathbb{G}$ for $(\mathscr{D}, \mathscr{C})$ and update U corresponding to rule $r \in \mathbb{R}$; if $\mathbb{G}$' is the result of applying r on $\mathbb{G}$ then our goal is to have $(\mathscr{D}', \mathscr{C})$ for $\mathscr{D}'/\mathbb{G}'$.

3 RDF/S databases as a typed graph

As stated int the previous section, RDF/S databases are formalized in two ways in this report: as classical triple-based RDF statements and as a typed graph. This section presents the latter.

RDF/S type graphs comprise 4 node types (Class, Individual, Literal, and Prop) and 6 edge types (CI, PI, domain, range, subclass, and subproperty). Each nodes have one attribute representing an URI, an URI, a value, and a name, respectively. PI-typed edges are the only ones with an attribute which represent the name of the property the edge is an instance of.

Fig 3 describes how each RDF triples are formalized in the typed graph model. The type and attributes of each graph element is indicated

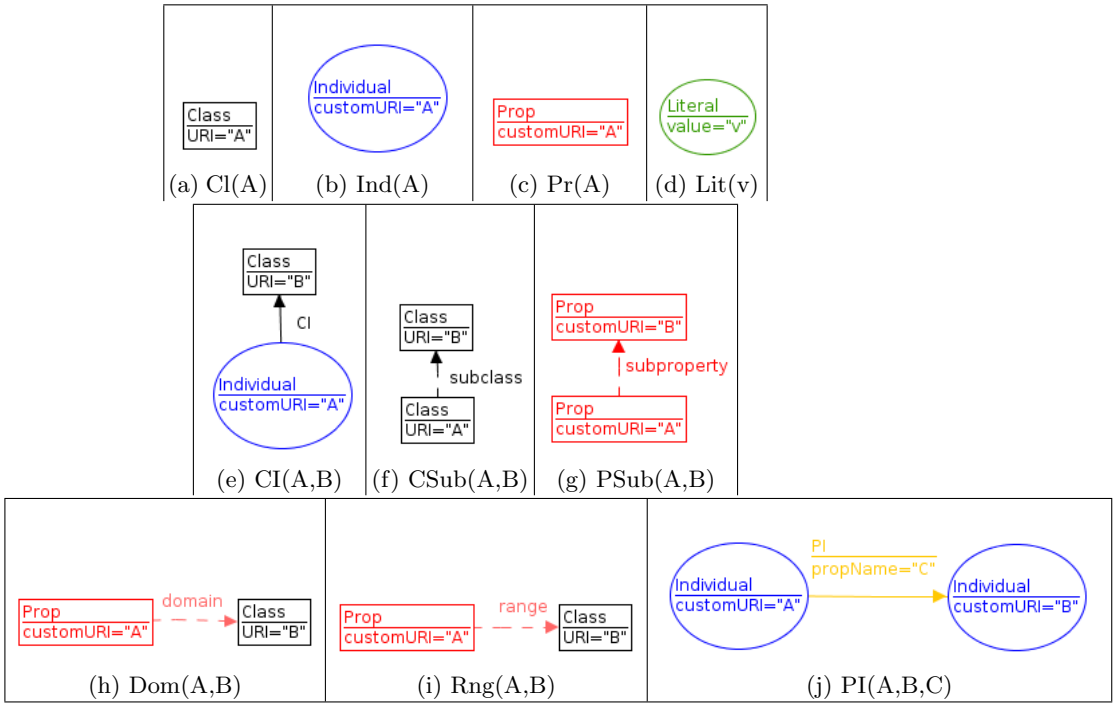

Fig. 3. RDF triples in the type graph model

4 Rewriting rules formalizing consistent RDF/S updates

To prevent the introduction of inconsistencies during updates, this section presents 1) formal specification of rules of $\mathbb{R}$ formalizing $\mathbb{G}$ evolution and 2) proofs that every rule in $\mathbb{R}$ ensures the preservation of every constraints in $\mathscr{C}$.

We adopt the SPO formalism [5] to specify rewriting rule as well as several of its extension to specify additional application conditions and restrict rule applicability: *Negative Application Conditions* (NACs) [3], *Positive Application Conditions* (PACs), and *General Application Conditions* (GACs) [6].

In this formalism, rewriting rules formalize both graph transformations and the context in which they may be applied. These rules may be *fully specified graphically*, enabling an easy-to-understand graphical view of the graph transformation that remains formal. We adopt the graphical conventions of AGG (The Attributed Graph Grammar System[1]) [7],

In total, 18 rules modelling consistent updates are presented herein. Their presentation follows a standard basic form filled by the main explanations of the rule. The proofs are in fact quite immediate. Indeed, rules have been specified to preserve consistency constraints by-design and their graphical specification, while formal, ease comprehension.

tiny

[1] `user.cs.tu-berlin.de/gragra/agg/index.html`

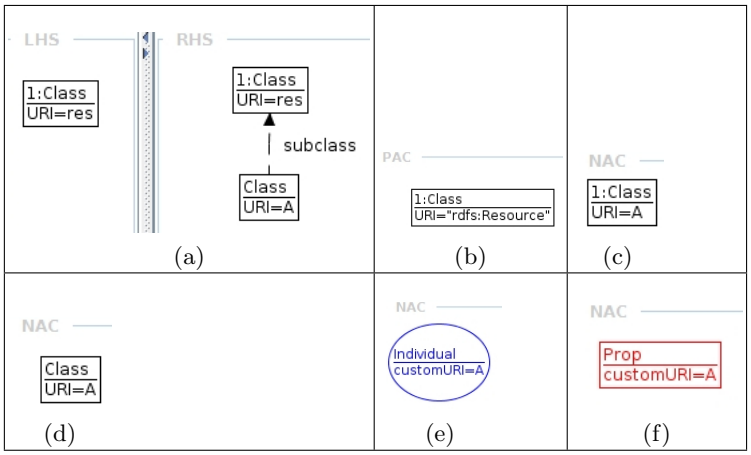

Fig. 4. Rewriting rule for the insertion of a class

4.1 Insertion of a Class (Fig. 4)

Update category: Schema evolution
User level: Only authorized users such as database administrators
Rule semantics: 1) SPO specification: (Fig. 4a)
LHS: A class-typed node with URI *res*. If this rule is applicable, there is such a class in the database;
RHS: LHS plus a class-typed node with the URI *A* as a sub-class of *Cl(res)*. The application of the rule will lead to their addition.
2) PAC specification: (Fig. 4b) variable *res* is assigned to "*rdfs:Resource*"; this PAC corresponds to constraint (13).
3)NAC_{res} and NAC_{cl}: (Fig. 4d and 4c respectively) these NACs are non-redundancy guarantee (ie, two classes may not have the same URI). A class *Cl(A)* may be inserted in the graph when: (*i*) *A* is not *rdfs:Resource* and (*ii*) another class with URI *A* does not already exist.
4) NAC_{ind} and NAC_{pr} (Fig.s 4e and 4f, respectively): guarantee that the sets of classes, properties, and individuals are disjoint (constraints 4 and 5). The rule is not applicable if *Pr(A)* or *Ind(A)* is in the database.

Proof of consistency preservation: It is clear from Fig. 1 that the addition of a class may activate constraints 4, 5, and 13 (*i.e.*, those having an atom with predicate *Cl* in their bodies). Thanks to the specification of NAC_{ind} and NAC_{pr}, constraints 4 and 5 are ensured. The PAC and SPO core of the rule in Fig. 4b and 4a impose the new class to be a subclass of *rdfs:Resource*, as constraint 13.

4.2 Deletion of a Class (Fig. 5)

Update category: Schema evolution
User level: Only authorized users such as database administrators

Rule semantics:
1) SPO specification: (Fig. 5a)
LHS: A class-typed node with URI A;
RHS: empty, rule's application leads to the deletion of the class with the URI A.
2) NAC_{res} : (Fig. 5b) states that the rule cannot be applied when A is *rdfs:Resource* – indeed the root of RDF class hierarchy cannot be deleted.
3)NAC_{dom} and NAC_{range}: (Fig. 5c and 5d respectively) impose that the class being deleted is neither the domain nor the range of any property.

Proof of consistency preservation: From Fig. 1, the deletion of a class may impact constraints 7, 9, 10, 11 (those having $Cl(A)$ on the RHS) together with constraints 13, 14, 15 (as consequences of possible deletion politics). Constraints 7 and 11 are preserved because $CSub$ and CI relations involving $Cl(A)$ are represented as edge incident to the node modelling $Cl(A)$. As in the SPO approach dangling edges are deleted, all $CSub$ and CI relations involving $Cl(A)$ are suppressed when this rule is applied. Constraint 9 (respect. 10) forbids the deletion of A as the domain (respect. as a range) of an existing property (which would also impact rule 15). Thanks to NAC_{dom} and NAC_{rng}, our graph rewriting rule is applicable only if the class to be deleted is neither the domain nor the range of any property. Finally as NAC_{res} forbids the deletion of class *rdfs:Resource*, constraints 13 and 14 are never violated by the deletion of a class.

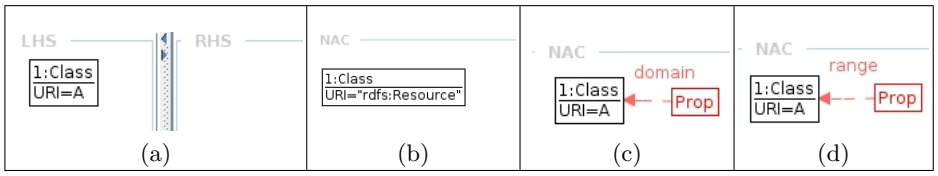

Fig. 5. Rule concerning the deletion of a class

4.3 Insertion of a Class Instance (Fig. 6

Update category: Instance evolution
User level: Any user
Rule semantics: 1) SPO specification: (Fig. 6a)
LHS: two nodes, with type class and URI B and type individual and URI A;
RHS: LHS plus a CI-typed edge from *Ind(A)* to *Cl(B)*.
2) NAC_{red} : (Fig. **??**) forbids the application of the rule if $CI(A, B)$ already exists in the database.
3)GAC: (Fig. 6b) this GAC is of the form "for all pattern GacTransCI there exists a pattern NestCond". GacTransCI is similar to LHS plus an unattributed (i.e. any attribute can be matched) node of type class and a subclass edge from $Cl(B)$ to said node. NestCond is GacTransCI plus a CI-typed edge from $Ind(A)$ to this new class. It ensures that the rule is applicable only if A is an instance of all super-classes of B.

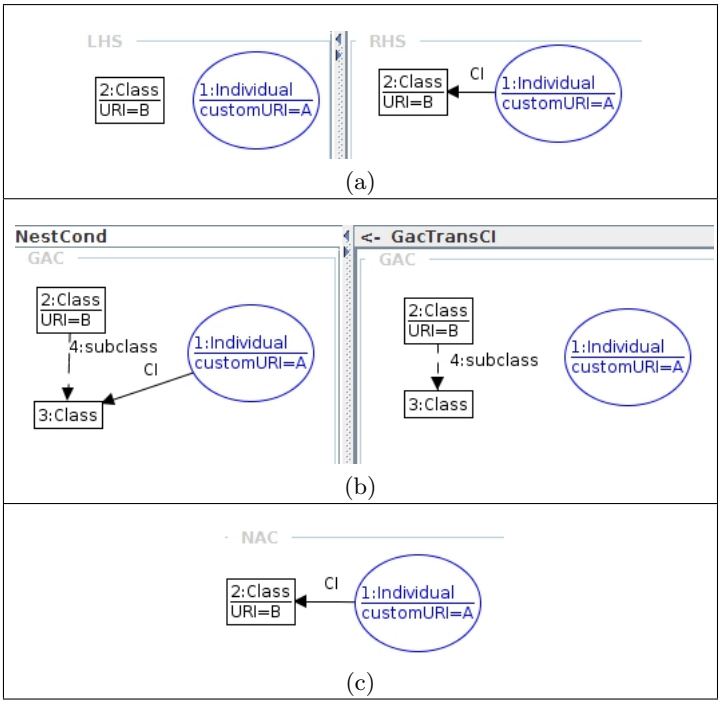

Fig. 6. Rule concerning the insertion of a class instance

Proof of consistency preservation: From Fig. 1, constraints 11 and 26 (having atoms with CI on their body) are impacted. Our graph rewriting rule ensures that the insertion of a class instance is performed only when the individual and its type already exist in the database (constraint 11). According to *GacTrans*, if there exists some super-class C of B and A is not an instance of C, then the class instance relation $CI(A, B)$ cannot be added (ensuring constraint 26).

4.4 Deletion of a class instance (Fig. 7)

Update category: Instance evolution
User level: Any user
Rule semantics:
1) SPO specification: (Fig. 7a)
LHS: An individual is linked to a class by an edge typed CI, *i.e.*, in the database, the individual is an instance of this given class.
RHS: The CI edge is removed (the individual still exists but is not an instance of the given class anymore).
2) NACs: The individual considered here is an instance of the given class. The NAC in Fig. 7b forbids the application of the rule when this individual is also connected to another individual by a property (*i.e.*, as part of a property instance) whose domain is the given class. The NACs in Figs. 7c and 7d are similar to the previous one, treating the cases where the individual is connected to a literal or to itself, respectively. The NACs in Figs. 7f and 7g impose similar prohibition when the given class is the range of the property. The NAC in Fig 7g ensures that no instance of resource is removed – since an individual is always an instance of class Resource. The NAC in Fig 7h disallows the rule application when the individual is an instance of a subclass of the given class.

Proof of consistency preservation: From Fig. 1, we remark that constraints 14, 24, 25, and 26 are concerned by the deletion of a class instance since an atom with predicate CI appear in their right-hand sides. The NAC in Fig 7g ensures the satisfaction of constraint 14. The NACs in Figs. 7b– 7f ensures the satisfaction of constraints 24 and 25. Finally the NAC in Fig 7h guarantees that constraint 26 is not violated.

4.5 Insertion of an individual (Fig. 8)

Update category: Instance evolution
User level: Any user
Rule semantics:
1) SPO specification: (Fig. 8a)
LHS: The class Resource;
RHS: LHS plus an individual with the URI A and a CI edge from the from the former to the latter. The application of the rule inserts the individual as an instance of class Resource.
2)NACs: The NACs defined in Fig. 8c and 8d guarantee that the sets of classes,

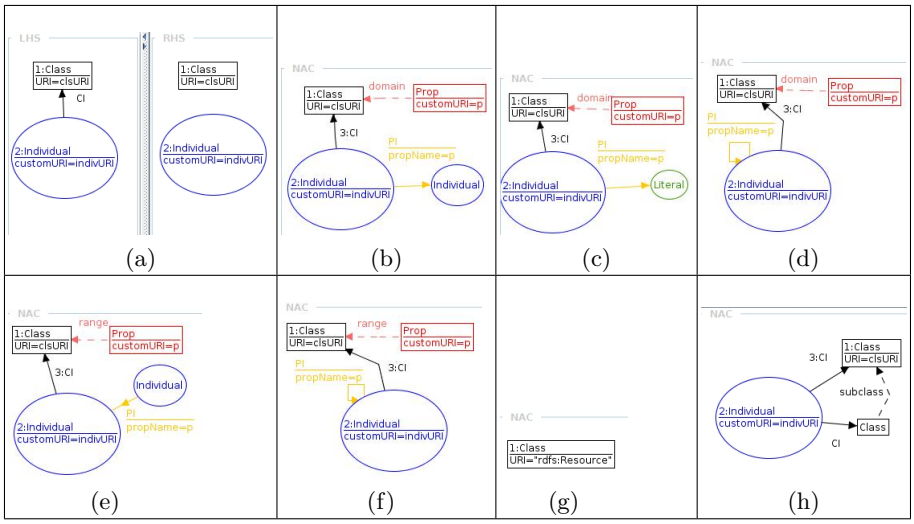

Fig. 7. Rule concerning the deletion of a class instance.

properties, and individuals are disjoint (constraints 5 and 6 in Fig. 1). The Nac from Fig. 8b forbids the addition of the individual if an individual with the same URI already exists.

Proof of consistency preservation: The addition of an individual triggers constraints 3 (Fig. 1) requiring an URL (given as a rule parameter) and constraints 5 and 6 which are guaranteed by the two NACs. 8c and 8d. Unicity is guaranteed by NAC 8b .

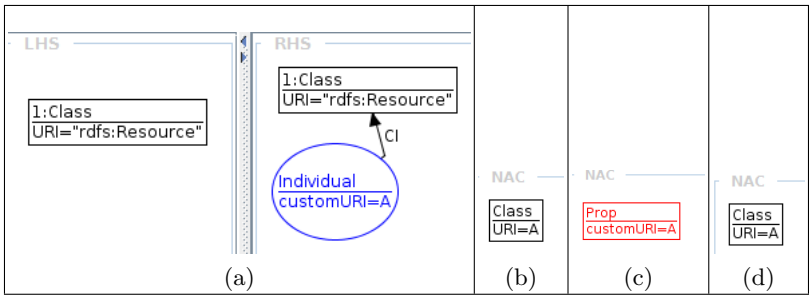

Fig. 8. Rewriting rule for the insertion of an individual

4.6 Deletion of an individual (Fig. 9)

Update category: Instance evolution
User level: Any user
Rule semantics:
1) SPO specification:
LHS: An individual with URI A;
RHS: the empty graph: rule's application leads to the deletion of the individual with the URI A (and all edge incident to it).

Proof of consistency preservation: From Fig. 1, the deletion of an individual may impact constraints 11 and 12. These constraints are still preserved because CI and PI relations involving an individual A are represented as an edge incident to the node modelling $Ind(A)$. In the SPO approach, dangling edges are deleted, thus all CI and PI relations involving $Ind(A)$ are suppressed when this rule is applied.

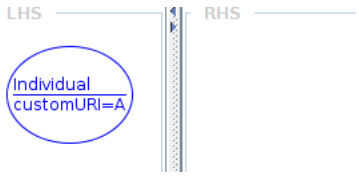

Fig. 9. Rewriting rule for the deletion of an individual

4.7 Insertion of a literal (Fig. 10)

Update category: Instance evolution
User level: Any user
Rule semantics:
1) SPO specification: (Fig. 10a)
LHS: Empty
RHS: The application of the rule inserts a node corresponding to the literal and its associated value in the graph.
2)NACs: (Fig. 10b) the NAC guarantees that such a literal does not exist yet.

Proof of consistency preservation: Property or class values such as textual strings are examples of RDF literals. The addition of a literal does not trigger any constraint (Fig. 1), just allowing its future use –as a value for property for instance–. The NAC avoids literal redundancy.

4.8 Deletion of a literal (Fig. 11)

Update category: Instance evolution
User level: Any user

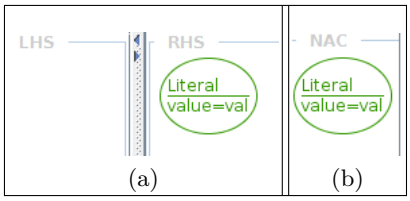

Fig. 10. Rewriting rule for the insertion of a literal

Rule semantics:
1) SPO specification: (Fig. 11a)
LHS: The node corresponding to the literal.
RHS: Empty.
2)NACs: (Fig. 11b) the NAC guarantees that the literal *rdfs:Literal* node is not the one been deleted; this node is a modelling artefact used as range when the range of a property is a literal and should not be deleted.

Proof of consistency preservation: From Fig. 1, the deletion of a literal is only concerned by constraint 12 when it is the value of a PI. In the SPO approach, dangling edges are deleted, thus all *PI* relations involving the literal are suppressed when this rule is applied.

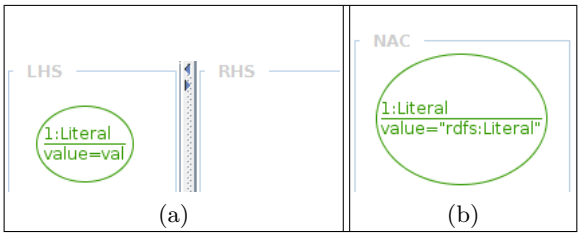

Fig. 11. Rewriting rule for the deletion of a literal

4.9 Insertion of a property

Two rules formalize the insertion of a property depending on the nature of its range.

• Insertion of a property having a class as its range (Fig. 12a)
Update category: Schema evolution
User level: Only authorized users such as database administrators
Rule semantics:
1) SPO specification: (Fig. 12a)
LHS: the LHS is composed of two classes with URI domain (denoted by *domain*

class) and range (denoted by *range class*);
RHS: LHS plus a node representing a *property*, whose URI is A, which is connected to the domain class by a *domain*-typed edge and to the range class by a *range*-typed edge. Thus, the application of this rule inserts a property between existing classes which are specified as the domain and range of that property.
2)NACs: NACs of Figs. 13a, 13c and 13d guarantee that there exist no class with URI A (Fig. 13c), including the range (Fig. 13a) and the domain (Fig.13d) classes. NAC of Fig 13e prohibits the existence of an individual whose URI is A. Again, the NACs ensure that classes, properties, and individuals are disjoint sets (constraints 4 and 5 in Fig. 1). Finally, NAC 13b guarantees that a property with the same URI does not already exists, guaranteeing unicity.

Proof of consistency preservation: The addition of a property concerns constraints 2, 4 and 6 of Fig. 1. The NACs in Figs. 13a, 13c and 13d of our rewriting rule ensure that these three constraints are respected. Notice that classes on LHS of our rule are not required to be distinguishable. Constraint 15 in Fig. 1 is also concerned by the insertion of a property. It requires the existence of a domain and a range for every property. On the LHS, our rewriting rule imposes the existence of two classes, while in its RHS, it establishes these classes as the property's domain and range. Constraint 15 is respected even when the same class is defined as the domain and the range of a given property.

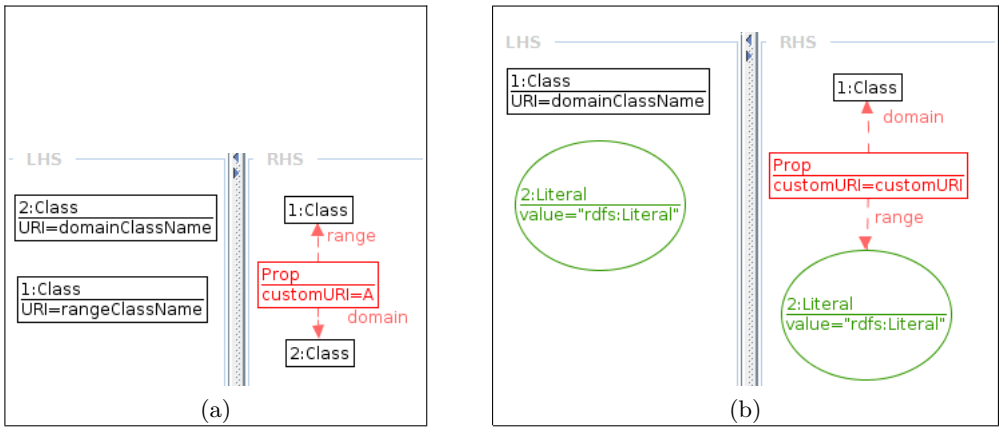

Fig. 12. Rewriting rules for inserting properties come in two versions according to the the type of the property's range.

• Insertion of a property having a literal as its range (Fig. 12b)
Update category: Schema evolution
User level: Only authorized users such as database administrators
Rule semantics:
1) SPO specification: (Fig. 12b)

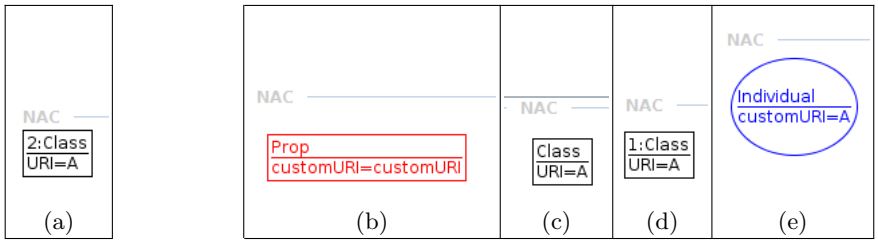

Fig. 13. NACs for the insertion of a property.

LHS: The LHS is composed of a class and a literal node attributed "rdfs:Literal". The latter is a special node used solely to specify that the range of a property is a literal ;
RHS: LHS plus a node representing a *property*, whose URI is A, which is connected to the class by a *domain*-typed edge and to "rdfs:Literal" by a *range*-typed edge. Thus, the application of the rule inserts a property between a class and "rdfs:Literal" which are specified, respectively, as the domain and range of that property.
2)NACs: This rule is concerned only by the four NACs defined in Fig. 13d, 13c, 13e, and 13b. These two first NACs guarantee that there exist no class with URI A (Fig. 13c), including the domain (Fig.13d) class. NAC of Fig 13e prohibits the existence of an individual whose URI is A. Again, the NACs ensure that classes, properties, and individuals are disjoint sets (constraints 4 and 5 in Fig. 1). Finally, NAC 13b guarantees that a property with the same URI does not already exists, guaranteeing unicity.

Proof of consistency preservation: The proof is similar to the previous one, the only difference is that the range is not a class, but a literal.

4.10 Deletion of a property (Fig. 14)

Update category: Schema evolution
User level: Only authorized users such as database administrators
Rule semantics:
1) SPO specification: (Fig. 14a)
LHS: a property with URI A;
RHS: the empty graph. Rule's application leads to the deletion of the property with the URI A.
2) NACs: NACs ensure that a property having instances cannot be deleted. Indeed, a property instance is a PI-typed edge between individuals (Fig. 14b where the instances of the property are individuals), between an individual and a literal (Fig. 14c) or an atomic loop (Fig. 14d).

Proof of consistency preservation: From Fig. 1, we can remark that constraints 8, 9, 10 and 12 are concerned by the deletion of a property. Constraints 8, 9, 10

are still respected after the application of the rule because, when the node corresponding to the property is deleted, all dangling edges are deleted. Here these edges indicate sub-property relationship (constraint 8), property domain (constraint 9) or property range (constraint 10). Constraint 12 is preserved because NACs prohibit the deletion of a property having instances.

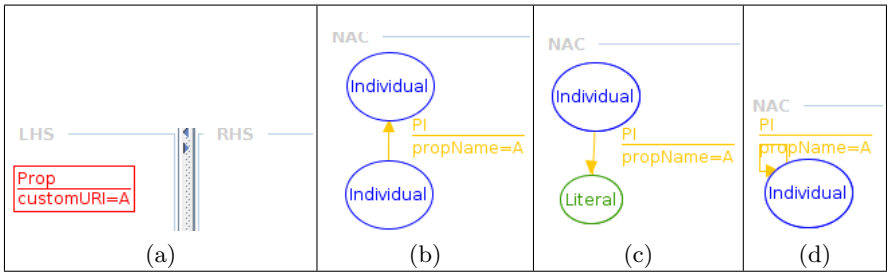

Fig. 14. Rule concerning the deletion of a property (with associated NACs).

4.11 Insertion of a property instance

We have two rules for the insertion of properties, we similarly have to consider different situations for the insertion of property instances.

• Insertion of a property instance for a property having a class as its range (Fig. 15)
Update category: Instance evolution
User level: Any user
Rule semantics:
1) SPO specification: (Fig. 15a)
LHS: the LHS is composed of a property (identified by *propURI*) having classes as its domain and range. Each of these classes has an individual as an instance.
RHS: LHS plus an edge connecting the individuals. The edge represents the property instance, indicating that the individuals are related to each other by the property *propURI*.
2)GAC: (Fig. 15b) In the schema of the graph database, if the property *propURI* is a sub-property of property *supPr* (for all pattern GacTransPI), then the two individuals should be already instances of *supPr* (NestCond), *i.e.*, the rule is applied only if the individuals involved in the property instance been inserted are already related by instances of all its super-properties.
3)NACs: (Fig. 15c) The NAC guarantees that the individuals are not already instances of the property *propURI*.

Proof of consistency preservation: The addition of a property instance concerns constraints 12, 24 and 25 of Fig. 1 which are ensured by the SPO specification. Let us denote by source (respectively, target) of a PI edge the node (individual) being the start point (respectively, the ending point) of the PI edge. The LHS

guarantees: (i) the existence of two individuals and the property in the graph (constraint 12), (ii) that the source of the PI is an instance of its class domain (constraint 24) and (iii) that the target of the PI is an instance of its class range (constraint 25). Constraint 27 is ensured by the GAC. An instance of P can be inserted between two individuals only if their is between the two an instance of all the super-properties of P.

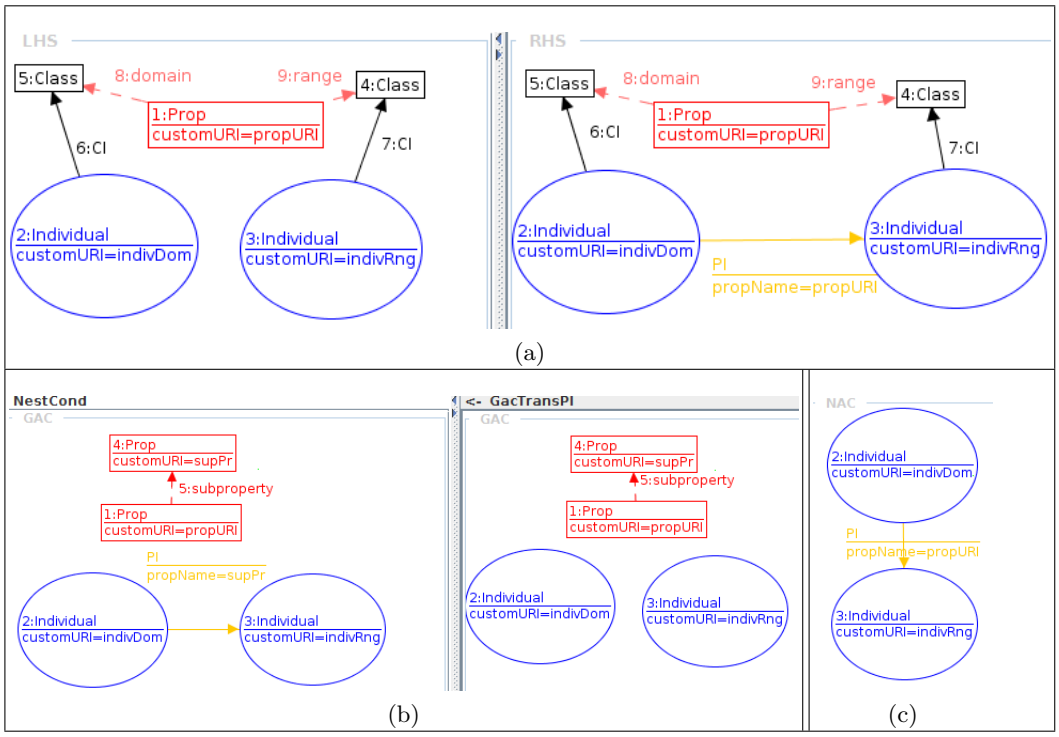

Fig. 15. Rewriting rule for the insertion of a property instance when the property range is a class.

• Insertion of a property instance for a property having a literal as its range (Fig. 16)
Update category: Instance evolution
User level: Any user
Rule semantics:
1) SPO specification: (Fig. 16a)
LHS: the LHS is composed of a property (identified by *propURI*) having a class as its domain and the literal class as its range, one individual (instance of the domain) and a literal.
RHS: LHS plus an edge from the individual to the literal. The edge represents the property instance, indicating that the individual and the literal are related

to each other by the property *propURI*.
2)GAC: (Fig. 16c) In the graph database schema, if the property *propURI* is a sub-property of property *supPr* (GacTransPI), then the individual and the literal should be already instances of *supPr* (NestCond), *i.e.*, the rule is applied only if the individual and the literal involved in the property instance been inserted are already involved in instances of all its super-properties.
3)NACs: (Fig. 16b) The NAC guarantees that the individual and the literal are not already linked as an instance of the property *propURI*.

Proof of consistency preservation: Similar to the proof in the previous item.

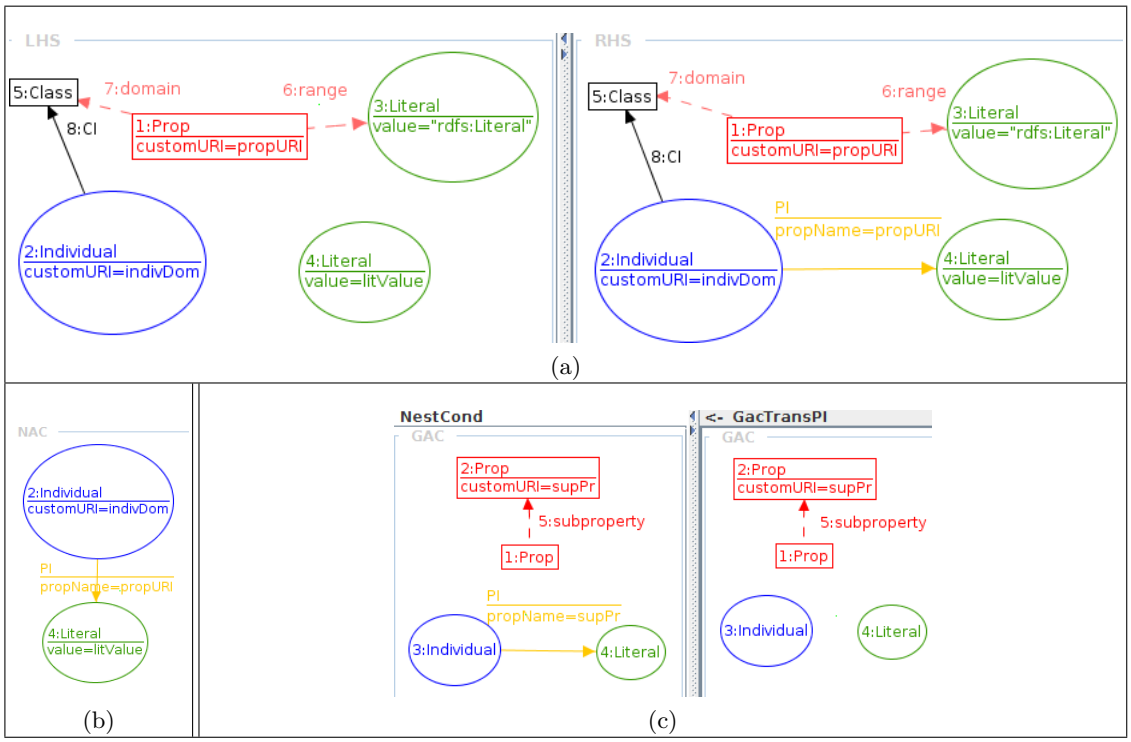

Fig. 16. Rewriting rule for the insertion of a property instance when the property range is a literal.

4.12 Deletion of a property instance

Similarly, we have to consider two different situations for the deletion of property instances.

• Deletion of a property instance for a property having a class as its range (Fig. 17)
Update category: Instance evolution

User level: Any user
Rule semantics:
1) SPO specification: (Fig. 17a)
LHS: the LHS is composed of two individuals denoted by *indivDom* and *indivRng* linked by a PI-typed edge attributed with *propURI*, *i.e.*, there is an instance of property *propURI* whose object is *indivDom* and value *indivRng*.
RHS: LHS minus the edge, the rule application leads to the removal of the property instance.
2)NACs: (Fig. 17b) If property *propURI* has at least one sub-property the individuals are also instances of the NAC forbids the rule application.

Proof of consistency preservation: The deletion of a property instance concerns constraint 27 of Fig. 1. The NAC ensures this constraint since the rule cannot be triggered if there exist sub-property instance links between the individuals.

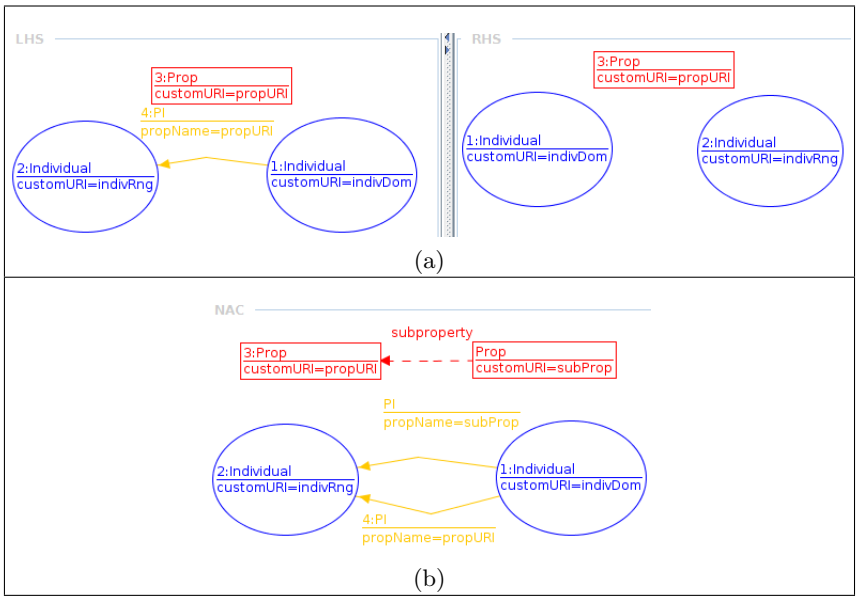

Fig. 17. Rewriting rule for the deletion of a property instance when the property range is a class.

• Deletion of a property instance for a property having a literal as its range (Fig. 18)
Update category: Instance evolution
User level: Any user
Rule semantics:
1) SPO specification: (Fig. 18a)
LHS: the LHS is composed of an individuals denoted by *indivDom* and a literal linked by a PI-typed edge with attribute *propURI*, *i.e.*, they are instances of

property *propURI*.
RHS: LHS minus the edge, the rule application leads to the removal of the edge linking the the individual to the literal.
2)NACs: (Fig. 18b) If property *propURI* has a sub-property with an instance involving *indivDom* and the literal, then the NAC forbids the rule application.

Proof of consistency preservation: Similar to the proof in the previous item.

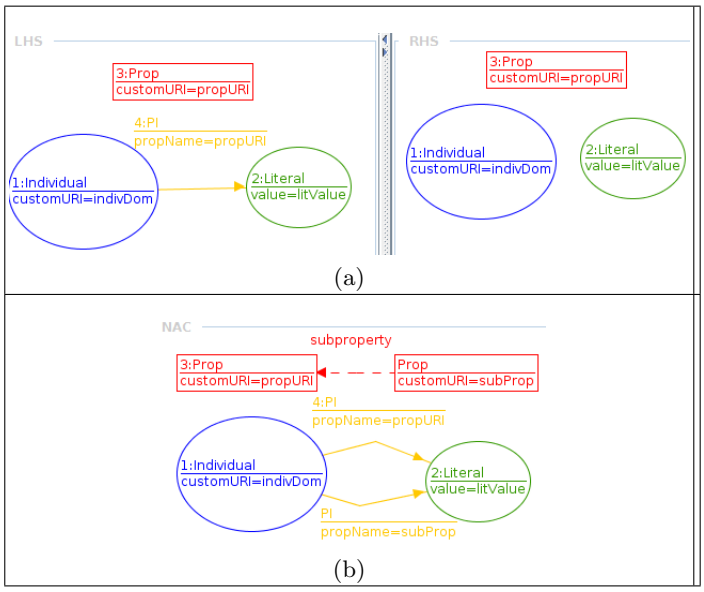

Fig. 18. Rewriting rule for the deletion of a property instance when the property range is a literal.

4.13 Insertion of a subclass relation (Figs. 19 and 20)

Update category: Schema evolution
User level: Only authorized users such as database administrators
Rule semantics:
1) SPO specification: Fig 19a
LHS: Two class-typed nodes with URI A and B;
RHS: An edge of type "subclass" from class B to class A is added, indicating that B is a subclass of A.
2) NACs: The NAC in Fig 19b ensures that class B is not a subclass of class A yet, while the NAC in Fig 19c prohibits the construction of cyclic subclass relationships – if A is already a subclass of B, the insertion of a subclass relationship from B to A is not possible. The NAC in Fig 19d forbids reflexive

subclass relationship.
3) GACs: Fig. 20
The tree in Fig 20a shows the entire logical combination of conditions imposed to the graph for the application of the rule. The right branch of the tree refers to GAC in Fig. 19b. In the graph database, all individuals which are instances of class B (`GacTransCI`) should also be instances of class A (`NestCond1`) for the rule to be applicable, *i.e.*, the rule is applicable only if all instance of B are already instances of A. The left sub-tree in Fig. 20 gathers two conditions. The leftmost one corresponds to Fig 20c. Each superclass of A (`GacTransCsub`) is a superclass of B (NestCond), *i.e.*, the application of the rule is possible only if there exists a subclass relationship between B and all superclass of A. The last condition is the one in Fig.20d. It states that all subclass of class B (`GacTransCsub2`) is a subclass of A (`NestCond2`).

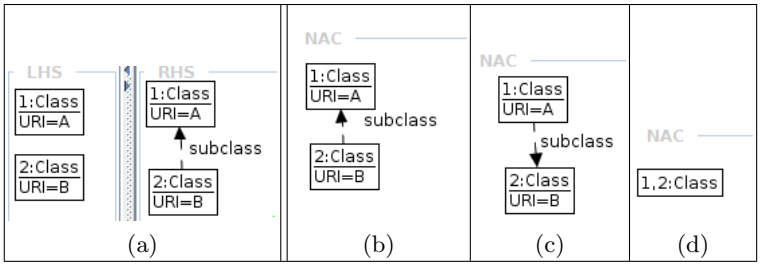

Fig. 19. Rule concerning the insertion of a subclass (with associated NACs).

Proof of consistency preservation: From Fig. 1, we remark that constraints 7, 18, 19 and 26 are concerned by the insertion of a subclass. Constraint 7 is not violated since the LHS of the SPO specification (Fig. 19a) imposes the existence of two classes before the addition of the edge representing the subclass relationship. GACs in Figs. 20c and 20d ensures the satisfaction of constraint 18 of Fig. 1, since they guarantee the application of the rule only if the class hierarchy stays consistent. Constraint 19 is implemented by the NACs which ensure that a cyclic subclass hierarchy is not possible. Constraint 26 is ensured by GAC in Fig. 20b which imposes the application of the rule only if all instances of class B are already instances of class A.

4.14 Deletion of a subclass relation (Fig. 21)

Update category: Schema evolution
User level: Only authorized users such as database administrators
Rule semantics:
1) SPO specification: Fig 21a

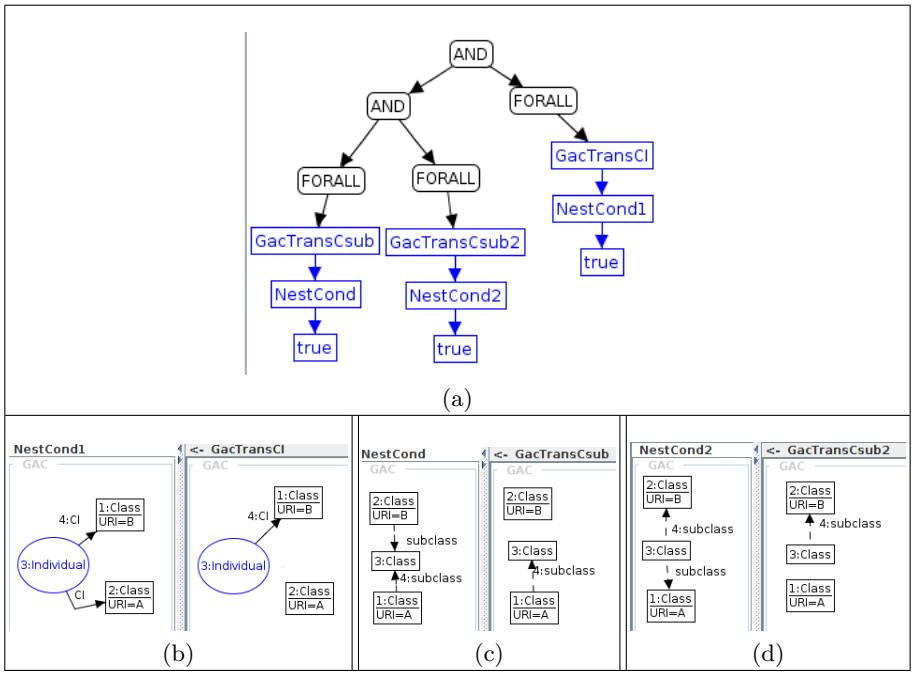

Fig. 20. GAGs concerning the insertion of a sub-class property.

LHS: Two class-typed nodes with URI *subClass* and *SuperClass* with a subclass-typed edge from the former to the latter;
RHS: LHS minus the edge, indicating its deletion.
2) NACs: The NAC in Fig 21b ensures that class *SuperClass* is not "rdfs:Resource", since all classes are subclasses of the root. The NAC in Fig 21c (resp. Fig 21d) ensures that *SuperClass* is not the domain (resp. the range) of a property which has a sub-property whose domain (resp. range) is *subClass*. If such properties exist, the rule is not applicable. The NAC in Fig 21e forbids the existence of a class that is both a subclass of *SuperClass* and a superclass of *subClass*, ensuring consistency with regard to transitivity.

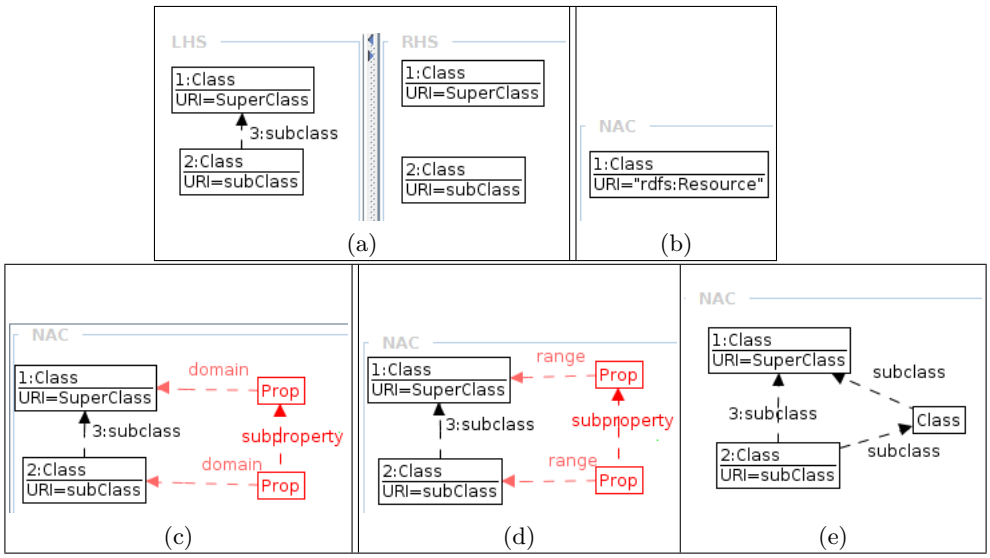

Fig. 21. Rule concerning the deletion of a subclass (with associated NACs).

Proof of consistency preservation: From Fig. 1, we remark that only constraints 13, 18, 22 and 23 are concerned by the deletion of a subclass relation. Constraint 13 is preserved thanks to the NAC defined in Fig. 21b that forbids the suppression of the sub-class relation to the root of the class hierarchy. The NAC of Fig. 21e ensures that the transitivity of the sub-class relation is respected, guaranteeing the respect of constraint 18. Finally, constraints 22 and 23 are ensured by NACs depicted in Figs. 21c and 21d, respectively. The sub-class relationship can not be deleted if it is required for the subsumption between two properties to reflect in their domains and ranges.

4.15 Insertion of a sub-property relation (Fig. 22, 23, 24, and 25)

Update category: Schema evolution
User level: Only authorized users such as database administrators
Rule semantics:
1) SPO specification: Fig 22a
LHS: Two property-typed nodes with URI *superProp* and *subProp*;
RHS: LHS plus a subproperty-typed edge from the node with URI *subProp* to the one with URI *superProp*, indicating its addition.
2) NACs: The NACs in Fig 22b and 22d ensure that the sub-property relation is neither reflexive nor symmetric, respectively. The NAC formalized in Fig. 22c forbids the insertion of the relation if it already exists.
3) GACs: The logical formula for the GACs is presented in Fig 23 while the GACs are formalized in Fig. 24 and 25. The logical formula states that all of the following conditions must be fulfilled for the rule to be applicable:

- $SameDom \vee SubDom$ (Fig. 24a and 24b); the properties have the same domain or the domain of *superProp* is a super-class of *subProp*'s domain;
- $SameRng \vee SameRngLit \vee SubRng$ (Fig. 24c), 24d, and 24e; the properties have the same range or the range of *superProp* is a super-class of *subProp*'s domain;
- for all patterns $GacTransPI$, $NestCond$ is true (Fig. 25a); all couple of individual related with an instance of *superProp* also have an instance of *subProp*.
- for all patterns $GacTransPISelf$, $NCselfPI$ is true (Fig. 25c); all individual with a reflexive instance of *superProp* also has an instance of *subProp*.
- for all patterns $GacTransPILit$, $NCtransPIlit$ is true (Fig. 25b); all couple of individual and literal with an instance of *superProp* also have an instance of *subProp*.
- for all patterns of $GacTransPsub$, $NCtransPsub$ is true (Fig.25d) (resp. $GacTransPsub2$, $NCtransub2$); all super-property of *superProp* is also a super-property of *subProp* (resp. all sub-property of *subProp* is also a sub-property of *superProp*).

Proof of consistency preservation: From Fig. 1, we remark that only constraints 8, 20, 21, 27, 22, and 23 are concerned by the deletion of a subproperty relation.

The typing of the relation (constraint 8) are guaranteed by the SPO part of the rule that may match only property-typed nodes.

Constraints 21 is preserved thanks to the NACs defined in Fig. 22b and 22d that forbid the introduction of a cycle in the sub-property relation. The GACs of Fig. 25d and 25e ensure the preservation of the relation transitivity (constraint 20).

The preservation of property instance propagation (constraint 27) is ensured by the GACs represented in Fig. 25c, **??**, and **??**.

Finally, constraints 22 and 23 are ensured by GACs depicted in Figs. 24a and 24b and 24c, 24d, and 24e, respectively. The sub-property relationship may be added only if the two properties have the same domain (resp. same range) or

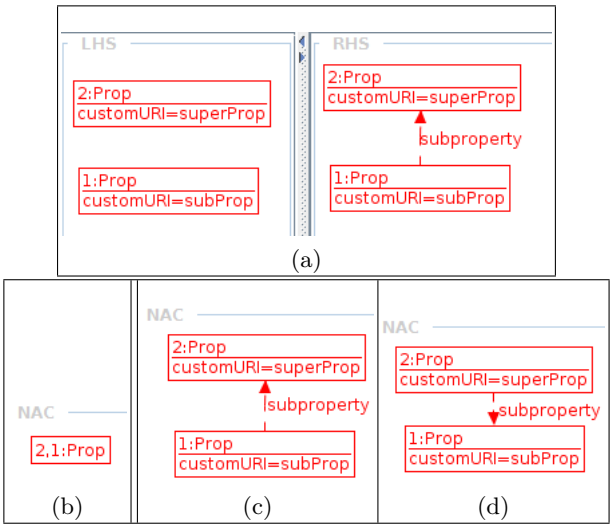

Fig. 22. Rule concerning the insertion of a subproperty relation subclass (with associated NACs).

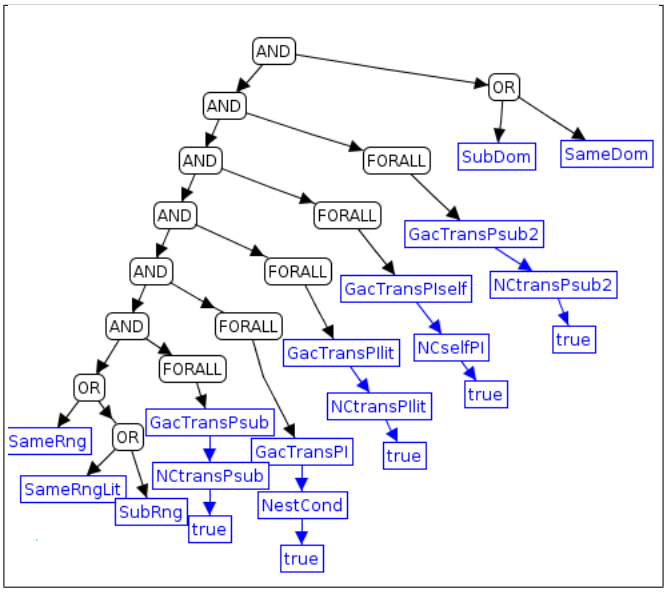

Fig. 23. Logical relations for GACs regarding the insertion of a subproperty relation subclass.

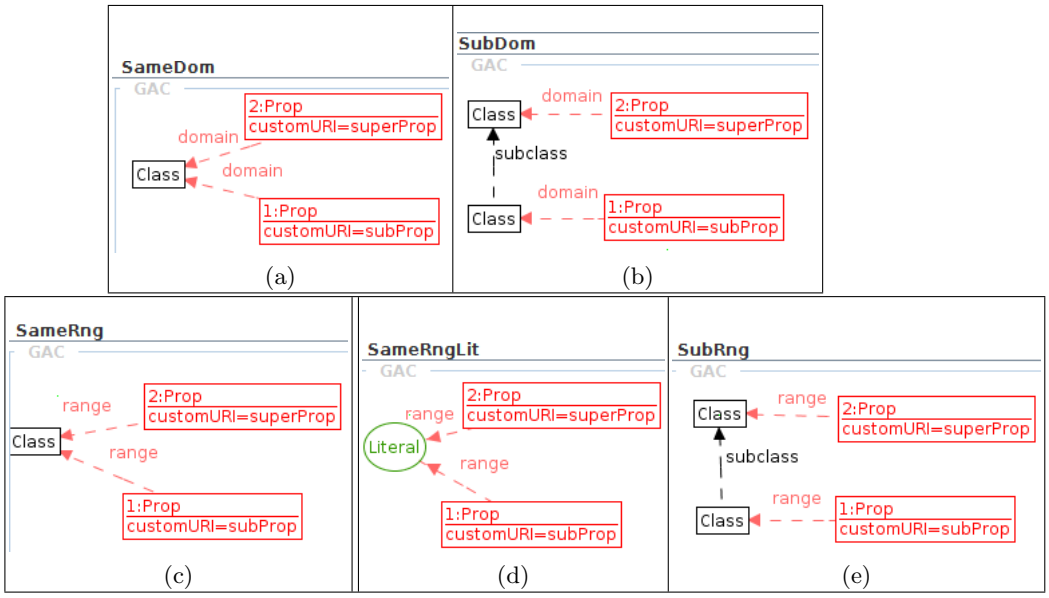

Fig. 24. GACs for the insertion of a subproperty relation subclass.

if their respective domains (resp. ranges) are related with an adequate sub-class relationship.

4.16 Deletion of a subproperty relation (Fig. 26)

Update category: Schema evolution
User level: Only authorized users such as database administrators
Rule semantics:
1) SPO specification: Fig 26a
LHS: Two property-typed nodes with URI *superProp* and *subProp* with a subproperty-typed edge from the former to the latter;
RHS: LHS minus the edge, indicating its deletion.
2) NAC: The NAC in Fig 26b ensures that there exists no third property which is both a super-property of *subProp* and a sub-property of *superProp*.

Proof of consistency preservation: From Fig. 1, we remark that only constraint 20 is concerned by the deletion of a suproperty relation. Its conservation is ensured by the NAC of Fig. 26b that forbids deletion of the relation if it has to exist due to transitivity.

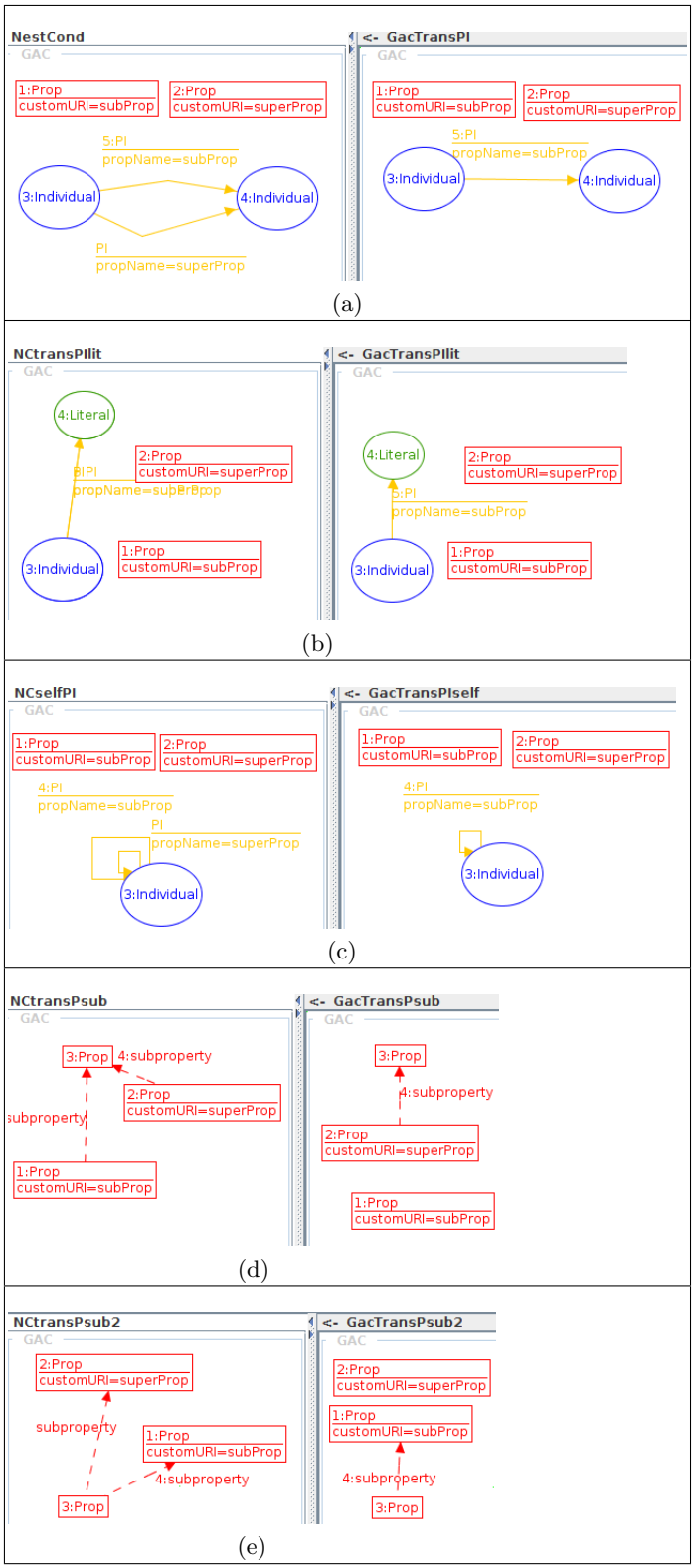

Fig. 25. GACs for the insertion of a subproperty relation subclass (cont').

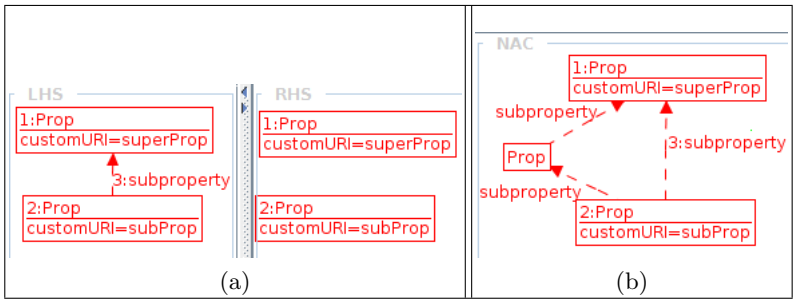

Fig. 26. Rule concerning the deletion of a subproperty relation (with associated NAC).

References

1. Abiteboul, S., Manolescu, I., Rigaux, P., Rousset, M.C., Senellart, P.: Web data management. Cambridge University Press (2011)
2. Flouris, G., Konstantinidis, G., Antoniou, G., Christophides, V.: Formal foundations for RDF/S KB evolution. Knowl. Inf. Syst. **35**(1), 153–191 (2013)
3. Habel, A., Heckel, R., Taentzer, G.: Graph grammars with negative application conditions. Fundam. Inf. **26**(3,4), 287–313 (Dec 1996), `http://dl.acm.org/citation.cfm?id=2379538.2379542`
4. Halfeld Ferrari, M., Laurent, D.: Updating RDF/S databases under constraints. In: Advances in Databases and Information Systems - 21st European Conference, ADBIS, Nicosia, Cyprus, Proceedings. pp. 357–371 (2017)
5. Löe, M.: Algebraic approach to single-pushout graph transformation. Theoretical Computer Science **109**(12), 181 – 224 (1993)
6. Runge, O., Ermel, C., Taentzer, G.: Agg 2.0 – new features for specifying and analyzing algebraic graph transformations. In: Schürr, A., Varró, D., Varró, G. (eds.) Applications of Graph Transformations with Industrial Relevance. pp. 81–88. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
7. Taentzer, G.: Agg: A graph transformation environment for modeling and validation of software. In: AGTIVE (2003)