B
GMR Implementation

In this appendix, we show the prototype implemented in each GMR. In particular, we explain:

- how we determine that a GMR is not fulfilled in merged ontologies, and
- how the unfulfilled GMR can be repaired.

We assume an ontology $\mathcal{O}$ contains a set of entities $\mathcal{E}$ including classes C, properties P, and instances I. The full list of used notations of this appendix has been shown in Table B.1.

TABLE B.1: The used notations and symbols in Appendix B.

Notation	Description
$\mathcal{O}_S$	source ontologies
$\mathcal{O}_M$	merged ontology
c_j	a class
c_j^T	a parent of a class
c_j^D	a child of a class
p_j	a property
c_j^p	a respective class (domain/range) of a property
I_j	an instance
c_j^I	the respective class of an instance
e_j	an entity
$\mathcal{E}$	all entities
X	all axioms
α	an axiom

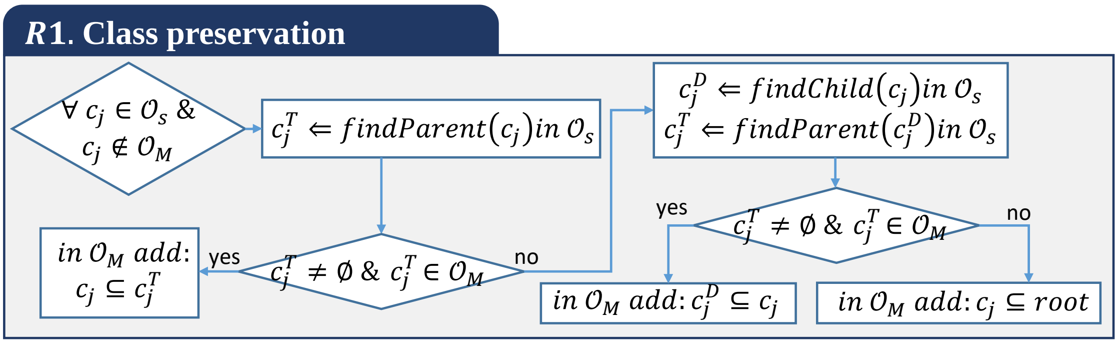
R1. Class preservation
$\forall c_j \in O_s$ & $c_j \notin O_M$
$c_j^T \Leftarrow findParent(c_j) in O_s$
$c_j^T \neq \emptyset$ & $c_j^T \in O_M$
yes
no
in O_M add: $c_j \subseteq c_j^T$
$c_j^D \Leftarrow findChild(c_j) in O_s$
$c_j^T \Leftarrow findParent(c_j^D) in O_s$
yes
$c_j^T \neq \emptyset$ & $c_j^T \in O_M$
no
in O_M add: $c_j^D \subseteq c_j$
in O_M add: $c_j \subseteq root$

FIGURE B.1: *R1*- Repair solution.

***R1*. Class preservation:**

- **Detecting *R1***: We check whether all classes (or their mapped classes) from the source ontologies exist in the merged ontology. If no, we mark them as missing classes.
- **Repairing *R1***: Any missed class (c_j) from source ontologies, i.e., $c_j \in \mathcal{O}_S$ but $c_j \notin \mathcal{O}_M$, should be added to the merged ontology. To perform this process:
  1. One parent (c_j^T) of this class in $\mathcal{O}_S$ should be found.
  2. If there exists a parent for it ($c_j^T \neq \emptyset$) and if this parent already exists in the merged ontology ($c_j^T \in \mathcal{O}_M$), this class is added as a child of its detected parent.
  3. If there is no parent for it ($c_j^T = \emptyset$), or the parent c_j^T does not exist in $\mathcal{O}_M$, then repeat this process by considering the child of c_j, i.e., one child of the missing class (c_j^D) should be found. If it exists in $\mathcal{O}_M$, c_j^D is added as a parent of c_j.
  4. Otherwise, it should be added to the root.

  Figure B.1 shows the repair process of *R1*.

***R2*. Property preservation:**

- **Detecting *R2*:** We check whether all properties (or their mapped properties) from the source ontologies exist in the merged ontology. If no, we mark them as missing properties.
- **Repairing *R2*:** Any missed property (p_j) from source ontologies, i.e., $p_j \in \mathcal{O}_S$ but $p_j \notin \mathcal{O}_M$, should be added to the merged ontology. To perform this process:
  1. The respective class (c_j^p) for the missing property from $\mathcal{O}_S$ should be found. It can be a domain or range of that property. Note that, domains or ranges of the properties are the type of class.
  2. If c_j^p exists in the merged ontology, we add in $\mathcal{O}_M$: $c_j^P\ hasProperty\ p_j$.
  3. If c_j^p does not exist in the merged ontology, the property p_j cannot be added to the merged ontology. This situation will be warned to the user.

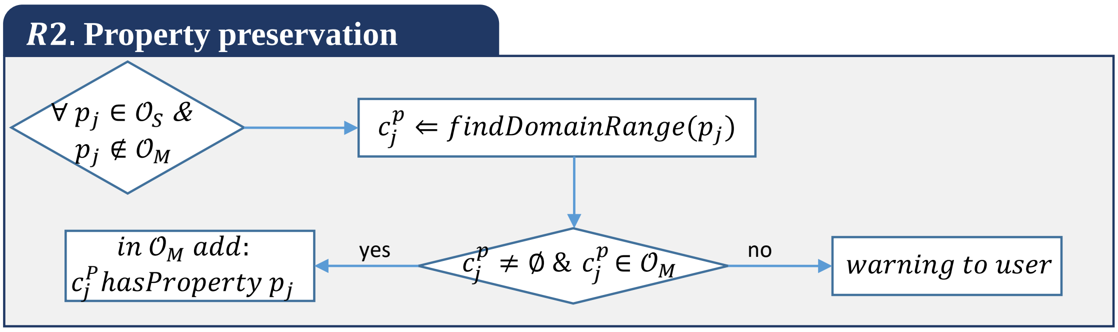
R2. Property preservation
∀ $p_j \in \mathcal{O}_S$ & $p_j \notin \mathcal{O}_M$
$c_j^p \Leftarrow findDomainRange(p_j)$
$c_j^p \neq \emptyset$ & $c_j^p \in \mathcal{O}_M$
yes
no
in $\mathcal{O}_M$ add: c_j^P hasProperty p_j
warning to user

FIGURE B.2: *R2*- Repair solution.

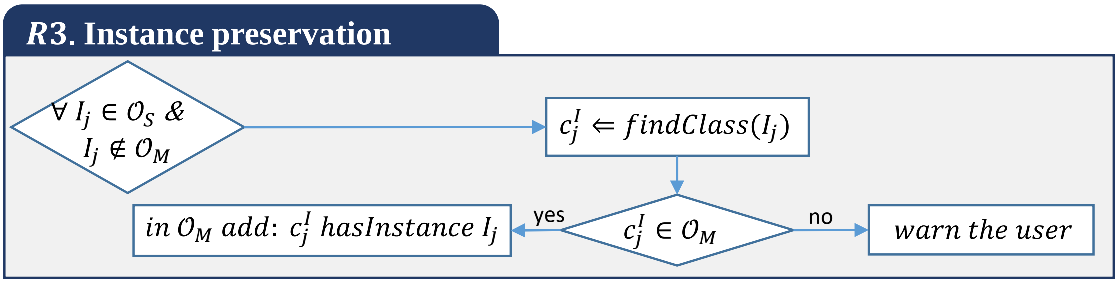
R3. Instance preservation
∀ $I_j \in \mathcal{O}_S$ & $I_j \notin \mathcal{O}_M$
$c_j^I \Leftarrow findClass(I_j)$
$c_j^I \in \mathcal{O}_M$
yes
no
in $\mathcal{O}_M$ add: c_j^I hasInstance I_j
warn the user

FIGURE B.3: *R3*- Repair solution.

Figure B.2 shows the repair process of *R2*. This process carries for all types of properties such as objects and data properties.

***R3*. Instance preservation:**

- **Detecting *R3*:** We check whether all instances from the source ontologies exist in the merged ontology. If no, we mark them as missing instances.
- **Repairing *R3*:** Any missed instance (I_j) from the source ontologies, i.e., $I_j \in \mathcal{O}_S$ but $I_j \notin \mathcal{O}_M$, should be added to the merged ontology. To perform this process:
  1. The respective class c_j^I of the missing instance I_j should be found.
  2. If c_j^I exists in the merged ontology, the instance (I_j) is added to its detected class (c_j^I).
  3. If c_j^I does not exist, we warn to the user that this instance could not be added to the merged ontology.

  Figure B.3 shows the repair process of *R3*.

***R4*. Correspondence preservation:**

- **Detecting *R4*:** We check whether all corresponding entities are integrated into one entity in the merged ontology or not. If no, we mark them.
- **Repairing *R4*:** For those entities which have some correspondences, but they did not merge into one entity, we combine them in an integrated entity. We add this

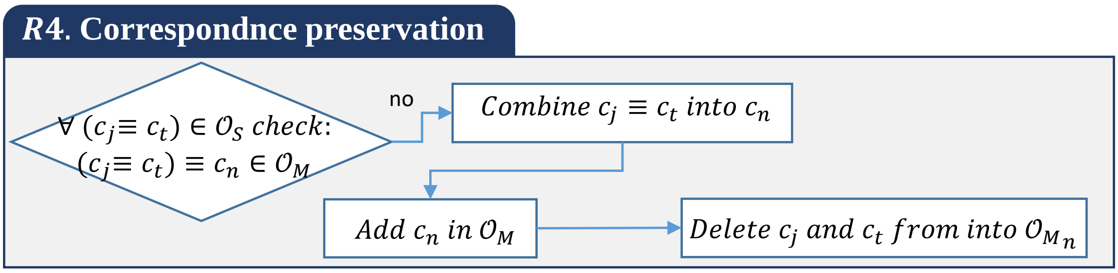
$R4$. Correspondnce preservation
$\forall\ (c_j \equiv c_t) \in O_S$ check:
$(c_j \equiv c_t) \equiv c_n \in O_M$
no
Combine $c_j \equiv c_t$ into c_n
Add c_n in O_M
Delete c_j and c_t from into O_{M_n}

FIGURE B.4: *R4*- Repair solution.

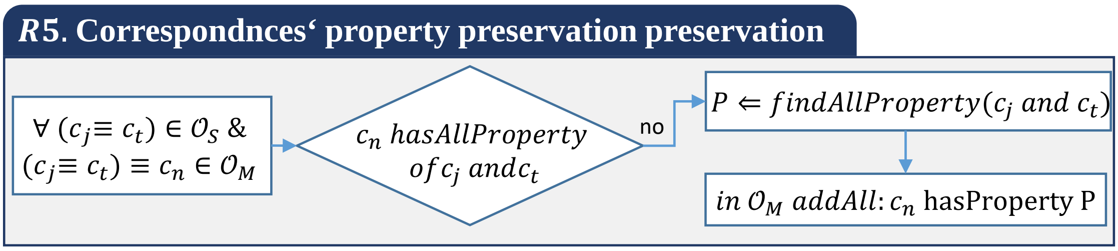
$R5$. Correspondnces' property preservation preservation
$\forall\ (c_j \equiv c_t) \in O_S$ &
$(c_j \equiv c_t) \equiv c_n \in O_M$
c_n hasAllProperty of c_j and c_t
no
$P \Leftarrow findAllProperty(c_j$ and $c_t)$
in O_M addAll: c_n hasProperty P

FIGURE B.5: *R5*- Repair solution.

new entity to the merged ontology, then delete those entities from the merged ontology. Figure B.4 shows the repair process of *R4*.

***R5.* Correspondence's property preservation:**

- **Detecting *R5*:** For all corresponding classes that they merged into an integrated entity in $\mathcal{O}_M$, we check whether this integrated entity has all properties of its corresponding entities. If no, we mark them.
- **Repairing *R5*:** For those marked entities, we add the properties of the corresponding entities to the integrated entity in $\mathcal{O}_M$. Figure B.5 shows the repair process of *R5*.

***R6.* Value preservation:**

- **Detecting *R6*:** For all corresponding entities with two different values, we check whether their integrated entity has both values. Moreover, if both values have a conflict, we mark them.
- **Repairing *R6*:** If an integrated entity does not have both values of its corresponding entities, we set both values for the integrated one. However, if their values have a conflict with each other, we need user interaction to solve it. Figure B.6 shows the repair process of *R6*.

***R7.* Structure preservation:**

- **Detecting *R7*:** In the merged ontology, we check whether each class has the same ancestor as the source ontologies. If no, we mark them.

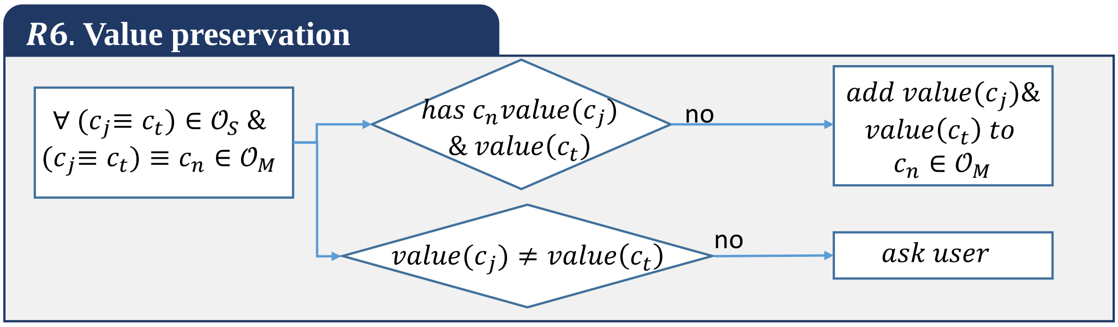
*R*6. Value preservation
$\forall\ (c_j \equiv c_t) \in \mathcal{O}_S\ \&$ $(c_j \equiv c_t) \equiv c_n \in \mathcal{O}_M$
$has\ c_n value(c_j)\ \&\ value(c_t)$
no
$add\ value(c_j)\&\ value(c_t)\ to\ c_n \in \mathcal{O}_M$
$value(c_j) \neq value(c_t)$
no
$ask\ user$

FIGURE B.6: *R6*- Repair solution.

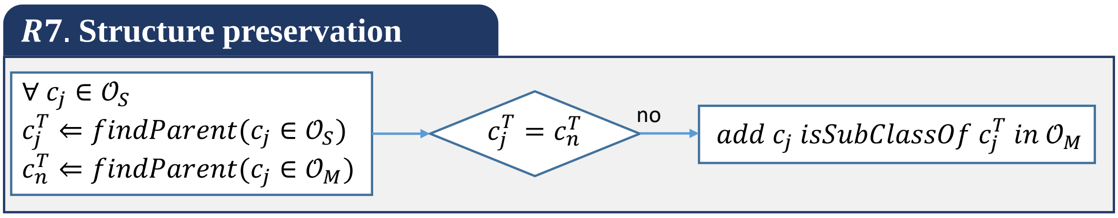
*R*7. Structure preservation
$\forall\ c_j \in \mathcal{O}_S$
$c_j^T \Leftarrow findParent(c_j \in \mathcal{O}_S)$
$c_n^T \Leftarrow findParent(c_j \in \mathcal{O}_M)$
$c_j^T = c_n^T$
no
$add\ c_j\ isSubClassOf\ c_j^T\ in\ \mathcal{O}_M$

FIGURE B.7: *R7*- Repair solution.

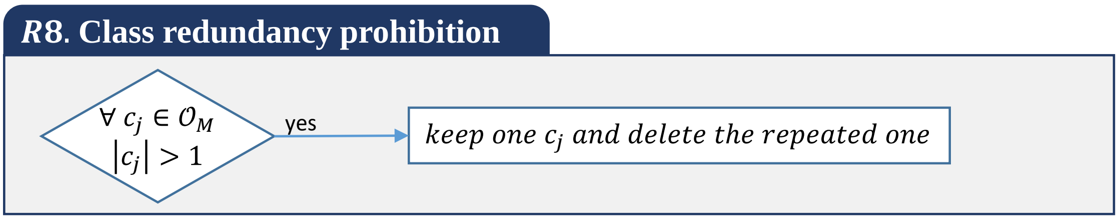
*R*8. Class redundancy prohibition
$\forall\ c_j \in \mathcal{O}_M$
$|c_j| > 1$
yes
$keep\ one\ c_j\ and\ delete\ the\ repeated\ one$

FIGURE B.8: *R8*- Repair solution.

- **Repairing *R7*:** For any marked class c_j, we add a new is-a relationship from class c_j to its respective parent (belong to $\mathcal{O}_S$). This process is carried, only if the parent of c_j exists in $\mathcal{O}_M$. Figure B.7 shows the repair process of *R7*.

***R8*. Class redundancy prohibition:**

- **Detecting *R8*:** If there is any class c_j, which is redundant (duplicated) in the merged ontology, we mark it.
- **Repairing *R8*:** For any marked class c_j, we keep one of them and delete the repeated one. Figure B.8 shows the repair process of *R8*.

***R9*. Property redundancy prohibition:**

- **Detecting *R9*:** If there is any property p_j, which is redundant (duplicated) in the merged ontology, we mark it.
- **Repairing *R9*:** For any marked property p_j, we keep one of them and delete the repeated one. Figure B.9 shows the repair process of *R9*.

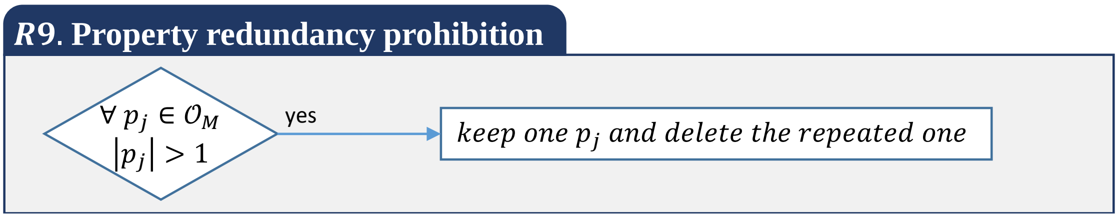
R9. Property redundancy prohibition
∀ $p_j \in O_M$
$|p_j| > 1$
yes
keep one p_j and delete the repeated one

FIGURE B.9: *R9*- Repair solution.

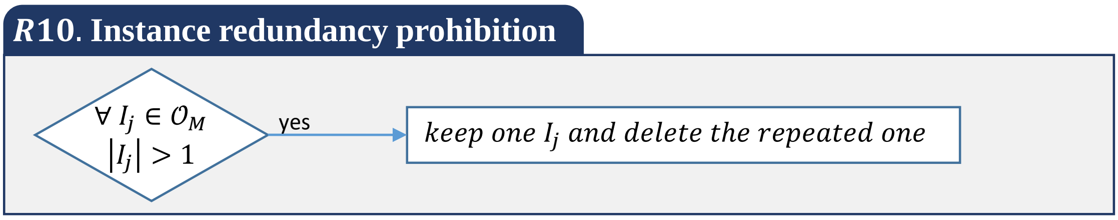
R10. Instance redundancy prohibition
∀ $I_j \in O_M$
$|I_j| > 1$
yes
keep one I_j and delete the repeated one

FIGURE B.10: *R10*- Repair solution.

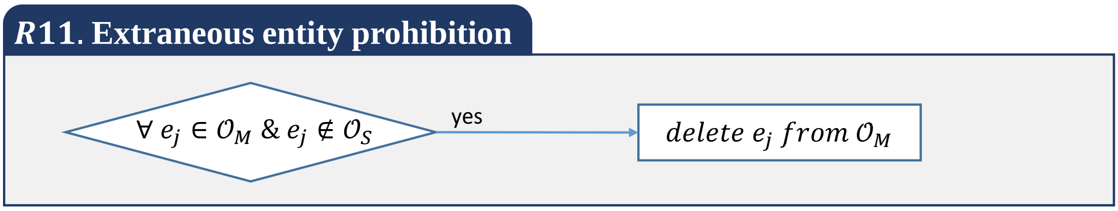
R11. Extraneous entity prohibition
∀ $e_j \in O_M$ & $e_j \notin O_S$
yes
delete e_j from O_M

FIGURE B.11: *R11*- Repair solution.

***R10.* Instance redundancy prohibition:**

- **Detecting *R10*:** If there is any instance I_j, which is redundant (duplicated) in the merged ontology, we mark it.
- **Repairing *R10*:** For any marked instance I_j, we keep one of them and delete the repeated one. Figure B.10 shows the repair process of *R10*.

***R11*. Extraneous entities prohibition:**

- **Detecting *R11*:** For all entities belong to $\mathcal{O}_M$, we check whether they exist in $\mathcal{O}_S$. If no, we mark them.
- **Repairing *R11*:** Any extra marked entity is deleted from $\mathcal{O}_M$. Figure B.11 shows the repair process of *R11*.

***R12*. Entailments deduction satisfaction:**

- **Detecting *R12*:** This is related to subsumption and equivalence entailments. For both, we follow the same process. First, we get subclass and equivalence axioms from the source ontologies. Then, we ask a reasoner to check whether the merged ontology can entail those axioms. If no, we mark them.

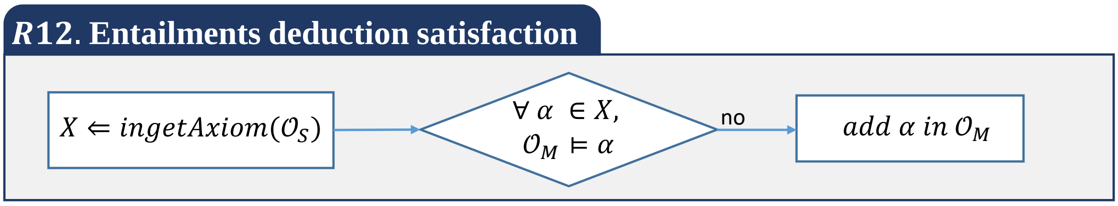
$R12$. Entailments deduction satisfaction
$X \Leftarrow ingetAxiom(\mathcal{O}_S)$
$\forall \alpha \in X,$
$\mathcal{O}_M \vDash \alpha$
no
$add\ \alpha\ in\ \mathcal{O}_M$

FIGURE B.12: *R12*- Repair solution.

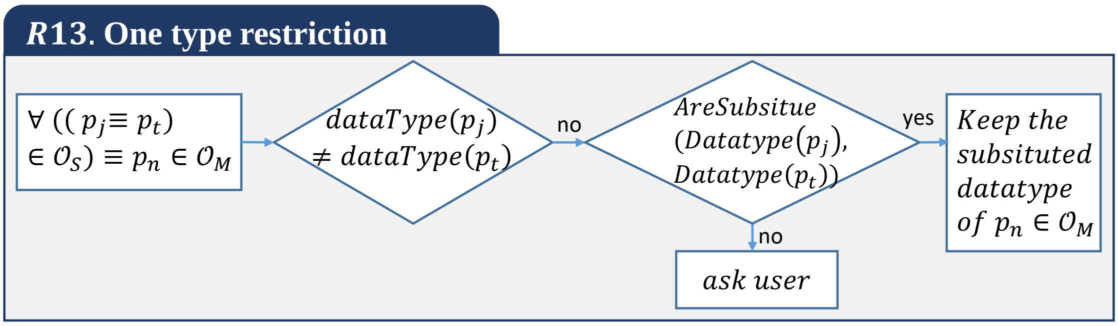

FIGURE B.13: *R13*- Repair solution.

- **Repairing *R12*:** We add those not-entailed axioms in $\mathcal{O}_M$. Figure B.12 shows the repair process of *R12*.

***R13*. One type restriction:**

- **Detecting *R13*:** We check for each integrated data type property in the merged ontology $(((p_j \equiv p_t) \in \mathcal{O}_S) \equiv p_n \in \mathcal{O}_M)$, whether they have the same datatype properties. If in the source ontologies, they have different values $(dataType(p_j) \neq dataType(p_t))$, the new integrated one p_n, cannot have both types at the same time. So, we mark it.

- **Repairing *R13*:** For any marked property, we check if both types are homogenous together, we only keep the substitute one. e.g., CHAR and STRING are homogenous together and we keep only the more general type, i.e., type STRING for p_n. If no, we ask the user. Figure B.13 shows the repair process of *R13*.

***R14*. Property value's constraint:**

- **Detecting *R14*:** We check all following constraint types:

  ObjectMaxCardinality, ObjectMinCardinality, ObjectExactCardinality, DataMaxCardinality, DataMinCardinality, DataExactCardinality, ObjectSomeValuesFrom, ObjectAllValuesFrom.

  For all entities belonging to source ontologies, we check the value of property of each constraint type, then we check the value of its mapped entity in the merged ontology. If they have different values, we mark them.

- **Repairing *R14*:** For any marked entity e_j, we keep the substitute value in $\mathcal{O}_M$. Figure B.14 shows the repair process of *R14*.

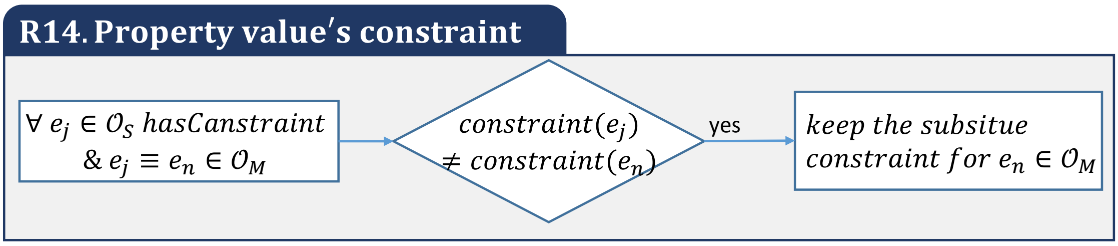
R14. Property value's constraint
$\forall\, e_j \in O_S$ hasCanstraint $\&\, e_j \equiv e_n \in O_M$
$constraint(e_j) \neq constraint(e_n)$
yes
keep the subsitue constraint for $e_n \in O_M$

FIGURE B.14: *R14*- Repair solution.

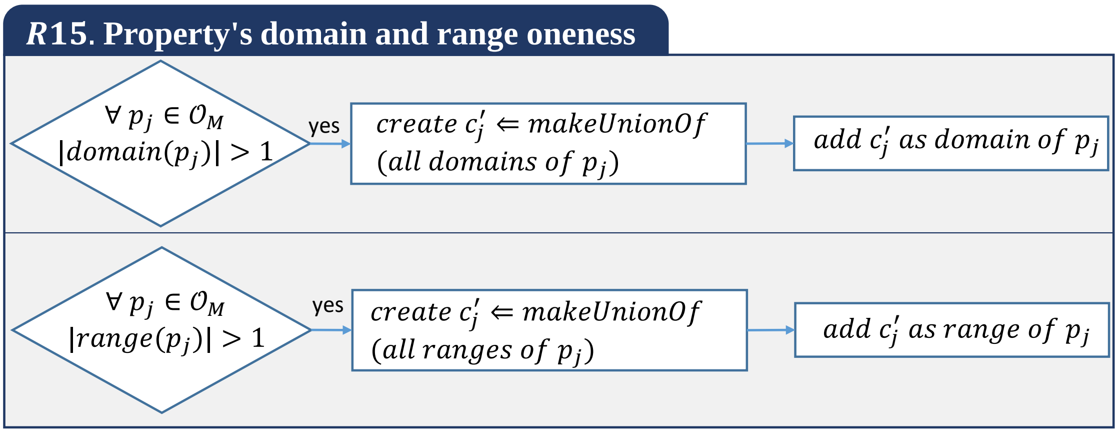
R15. Property's domain and range oneness
$\forall\, p_j \in O_M$ $|domain(p_j)| > 1$
yes
create $c'_j \Leftarrow makeUnionOf$ (all domains of p_j)
add c'_j as domain of p_j
$\forall\, p_j \in O_M$ $|range(p_j)| > 1$
yes
create $c'_j \Leftarrow makeUnionOf$ (all ranges of p_j)
add c'_j as range of p_j

FIGURE B.15: *R15*- Repair solution.

***R15*. Property's domain and range oneness:**

- **Detecting *R15*:** If a property p_j in the merged ontology has multiple domains or ranges ($|domainRange(p_j)| > 1$), we mark it.
- **Repairing *R15*:** For any marked property p_j, we create a new class as the union of all its domains or ranges. We then add this new class as domain/range of property p_j. Figure B.15 shows the repair process of *R15*.

***R16*. Acyclicity in the class hierarchy:**

- **Detecting *R16*:** There are two types of cycles:
  - Self-cycle: To detect the self-cycle, we check for any class c_j, this class should not appear to the list of its parents. If so, we mark it.
  - Recursive-cycle: During the visiting of all parents of a class, if we visit more than one time a parent, we mark it as a cycle.
- **Repairing *R16*:** We delete the respective axiom that caused a self-cycle in the merged ontology. To repair the recursive-cycle, we need user interaction. Figure B.16 shows the repair process of *R16*.

***R17*. Acyclicity in the property hierarchy:**

- **Detecting *R17*:** There are two types of cycles in the property hierarchy:

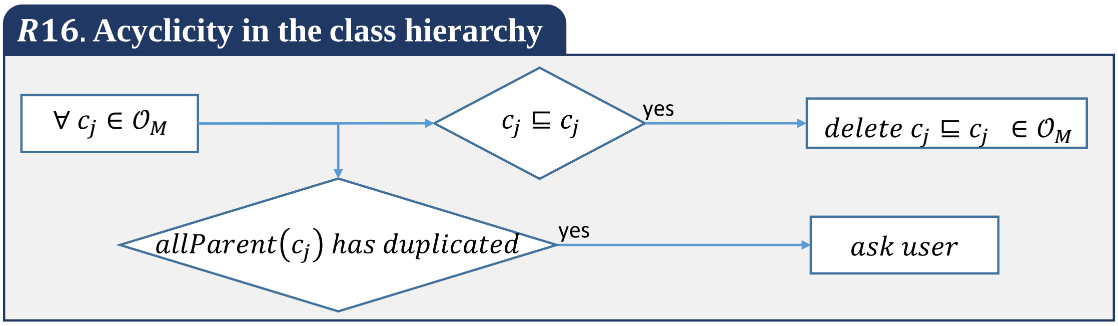
R16. Acyclicity in the class hierarchy
∀ $c_j \in \mathcal{O}_M$
$c_j \sqsubseteq c_j$
yes
delete $c_j \sqsubseteq c_j \in \mathcal{O}_M$
allParent(c_j) has duplicated
yes
ask user

FIGURE B.16: *R16*- Repair solution.

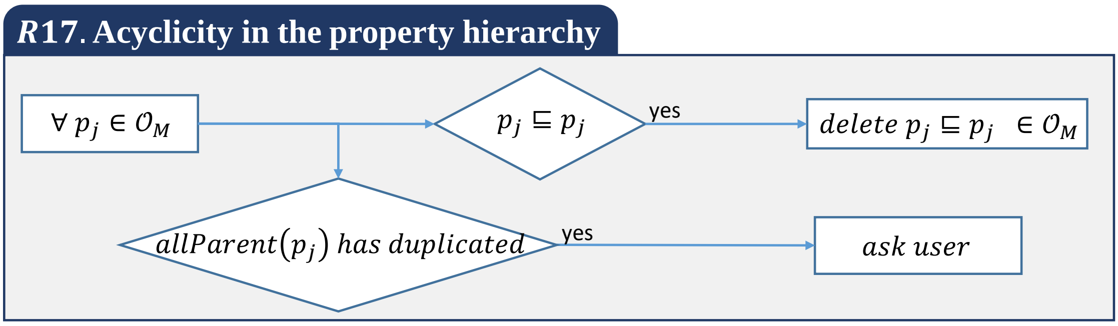
R17. Acyclicity in the property hierarchy
∀ $p_j \in \mathcal{O}_M$
$p_j \sqsubseteq p_j$
yes
delete $p_j \sqsubseteq p_j \in \mathcal{O}_M$
allParent(p_j) has duplicated
yes
ask user

FIGURE B.17: *R17*- Repair solution.

- Self-cycle: To detect this type of cycle, we check for any property p_j, this property should not appear as its subPropertyOf. If so, we mark it.
- Recursive-cycle: During the visiting subPropertyOf hierarchy for property p_j, if we visit more than one time a property, we mark it as a cycle.

- **Repairing *R17*:** We delete the respective axiom that caused a self-cycle on the property hierarchy in the merged ontology. To repair the recursive-cycle, we need user interaction. Figure B.17 shows the repair process of *R17*.

***R18*. Prohibition of properties being inverses of themselves:**

- **Detecting *R18*:** We check whether in the merged ontology, there is a property that is inverse of itself. If so, we mark it.
- **Repairing *R18*:** We delete the related inverseOf axiom of the marked property in the merged ontology. Figure B.18 shows the repair process of *R18*.

***R19*. Unconnected class prohibition:**

- **Detecting *R19*:** If there is any class (c_j) which does not have any connections to the other classes in the is-a hierarchy, i.e. $SubClass(c_j) \& SuperClass(c_j) = \emptyset$, we mark it.
- **Repairing *R19*:** The repair process includes:

1. One of sub or superclass of c_j from $\mathcal{O}_S$, called c_j^T should be found.

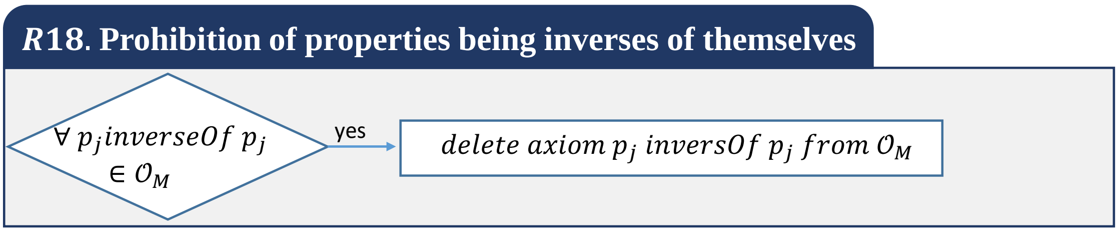
$R18$. Prohibition of properties being inverses of themselves
$\forall\, p_j inverseOf\, p_j \in O_M$
yes
$delete\ axiom\ p_j\ inversOf\ p_j\ from\ O_M$

FIGURE B.18: *R18*- Repair solution.

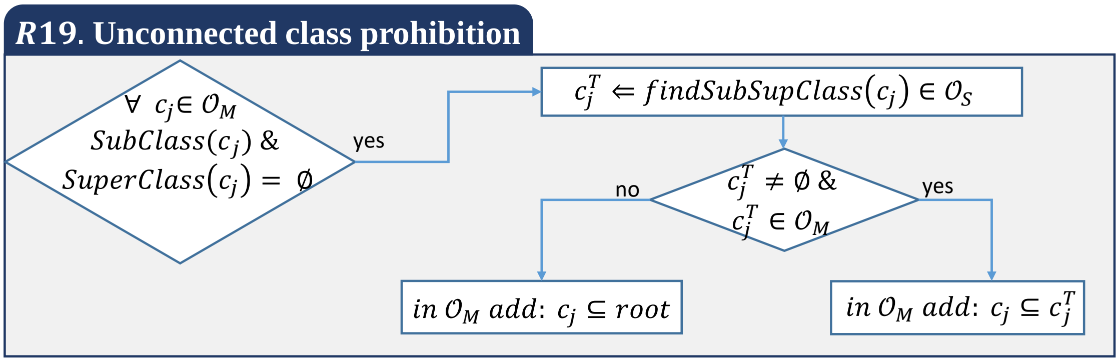
$R19$. Unconnected class prohibition
$\forall\ c_j \in O_M$ $SubClass(c_j)$ & $SuperClass(c_j) = \emptyset$
yes
$c_j^T \Leftarrow findSubSupClass(c_j) \in O_S$
$c_j^T \neq \emptyset$ & $c_j^T \in O_M$
no
yes
$in\ O_M\ add\!:\ c_j \subseteq root$
$in\ O_M\ add\!:\ c_j \subseteq c_j^T$

FIGURE B.19: *R19*- Repair solution.

2. If c_j^T is not null and exists in the merged ontology, we add c_j to c_j^T with an is-a relationship.
3. Otherwise, we add c_j to the root of the merged ontology.

Figure B.19 shows the repair process of *R19*.

***R20*. Unconnected property prohibition:**

- **Detecting *R20*:** If there is any property p_j which does not have any connections to the other properties in the subPropertyOf hierarchy, we mark it.
- **Repairing *R20*:** The repair process includes:
  1. One of sub or super property of p_j from $\mathcal{O}_S$, called p_j^T should be found.
  2. If p_j^T exists in the merged ontology, we add p_j to p_j^T with a subPropertyOf relationship.
  3. Otherwise, ask the user.

  Figure B.20 shows the repair process of *R20*.

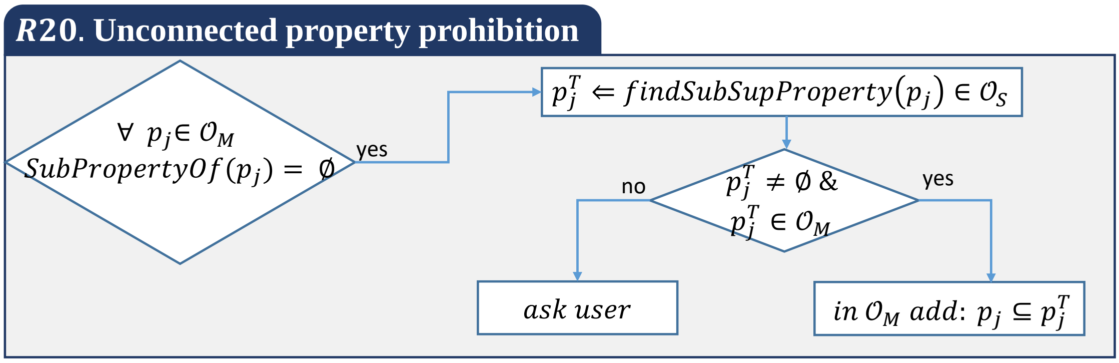
R20. Unconnected property prohibition
$\forall\ p_j \in \mathcal{O}_M$ $SubPropertyOf(p_j) = \emptyset$
yes
$p_j^T \Leftarrow findSubSupProperty(p_j) \in \mathcal{O}_S$
$p_j^T \neq \emptyset$ & $p_j^T \in \mathcal{O}_M$
no
yes
ask user
in $\mathcal{O}_M$ add: $p_j \subseteq p_j^T$

FIGURE B.20: *R20*- Repair solution.