Appendix A

GMR Implementation

In this appendix, we show one of all possible solutions to implement the idea behind each GMR. For each GMR, we explain: (i) how can we detect that a GMR is not fulfilled in the merged ontology, and (ii) how can we repair it.

We consider an ontology $\mathcal{O}$ contains a set of entities $\mathcal{E}$ including classes C, properties P and instances I. We called all by "entities".

The used notations through this documentation have been shown in Appendix A.1.

TABLE A.1: The used notations, symbols and nomenclature in this chapter.

Notation	Abbreviation for	Description
$\mathcal{O}_S$	Source Ontologies	-
$\mathcal{O}_M$	Merged Ontology	-
c_j	class	a sample class
c_j^T	-	one sample parent of a class
c_j^D	-	one sample child of a class
p_j	-	one sample property
c_j^p	-	a respective class (domain/range) of a property
I_j	-	a sample instance
c_j^I	-	the respective class of an instance
e_j	-	a sample entity
$\mathcal{E}$	Entity	all entities
X	Axioms	all axioms
α	-	a sample axiom

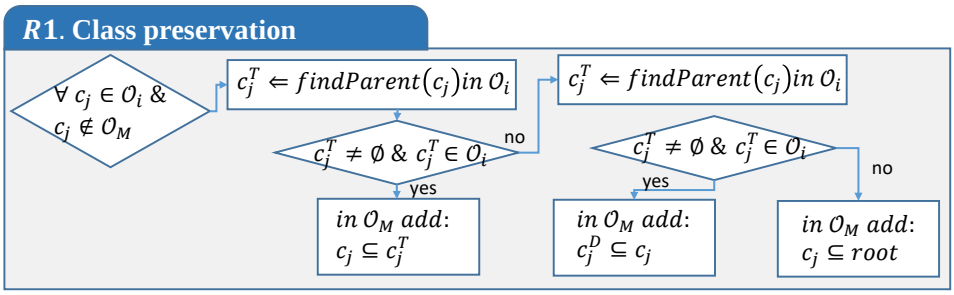

FIGURE A.1: R1- Repair solution.

***R1.* class preservation:**

- **Find**: We check whether all classes (or their mapped classes) from the source ontologies exist in the merged ontology. If no, we mark them as the missing classes.
- **Repair**: If there is any class (c_j) from source ontologies, which is missed in the merged ontology, i.e. $c_j \in \mathcal{O}_S$ but $c_j \notin \mathcal{O}_M$, this class (c_j) should add to the merged ontology. To do this:
  1. Find one parent (c_j^T) of this class in $\mathcal{O}_S$.
  2. If there exists a parent for it ($c_j^T \neq \emptyset$) and if this parent already exists in the merged ontology ($c_j^T \in \mathcal{O}_M$), add this class as a child of its found 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., find one child of the missing class (c_j^D), if it exists in $\mathcal{O}_M$, add c_j^D as a parent of c_j.
  4. Otherwise, add it to the root.

***R2.* Property preservation:**

- **Find:** We check whether all properties (or their mapped properties) from the source ontologies exist in the merged ontology. If no, we mark them as the missing properties.
- **Repair:** If there is any property (p_j) from source ontologies that are missed in the merged ontology, i.e., $p_j \in \mathcal{O}_S$ but $p_j \notin \mathcal{O}_M$, then p_j should add to the merged ontology. To do this:
  1. Find the respective class (c_j^p) for the missing property from $\mathcal{O}_S$. This can be a domain or range of that property. (Domains or ranges of the properties are the type of class).
  2. If c_j^p is not null and exist in the merged ontology, 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 process carries for all types of properties including objects, dataproperties, etc.

***R3.* Instance preservation:**

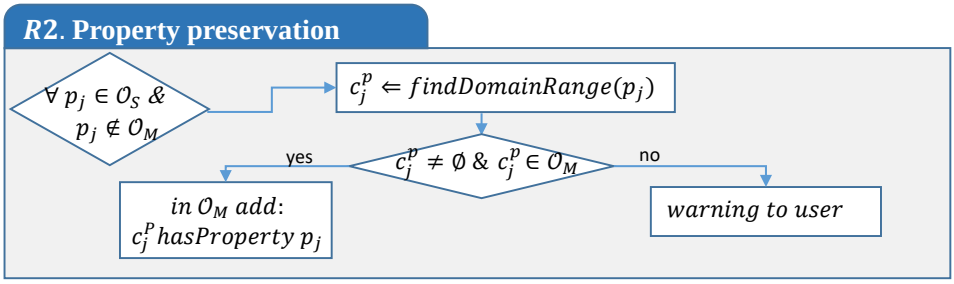

FIGURE A.2: R2- Repair solution.

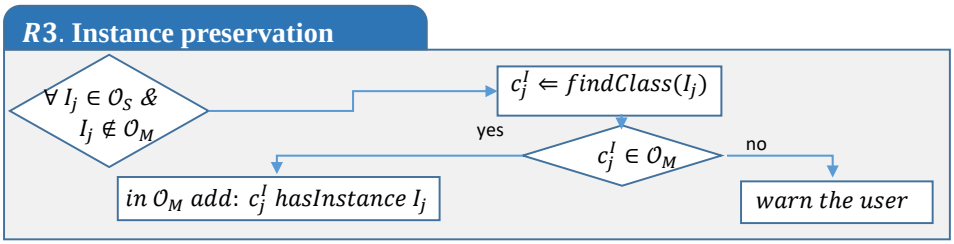

FIGURE A.3: R3- Repair solution.

- **Find:** We check whether all instances from the source ontologies exist in the merged ontology. If no, we mark them as the missing instances.
- **Repair:** If there is any instance (I_j) from the source ontologies which is missed in the merged ontology, i.e., $I_j \in \mathcal{O}_S$ but $I_j \notin \mathcal{O}_M$, then I_j should add to the merged ontology. To do this:
  1. Find the respective class c_j^I of the missing instance I_j.
  2. If c_j^I exists in the merged ontology, add the instance (I_j) to its found class (c_j^I).
  3. If c_j^I does not exist, we warn to the user that this instance could not add to the merged ontology.

***R4.* Correspondence preservation:**

- **Find:** We check whether all corresponding entities are integrated into one entity in the merged ontology or not. If no, we mark them.
- **Repair:** For those entities which have some correspondences, but they do not integrate together in one entity, we combined them in one entity. We add this new entity to the merged ontology, then delete those entities from the merged ontology.

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

- **Find:** We check for all corresponding classes that they merged to an integrated entity in $\mathcal{O}_M$, whether this integrated entity has the properties of its corresponding entities. If no, we mark them.
- **Repair:** For those marked entities, we added the properties of the corresponding entities to the integrated entity in $\mathcal{O}_M$.

***R6.* Value preservation:**

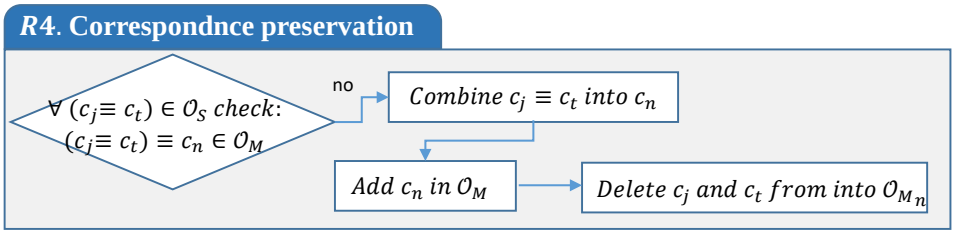

FIGURE A.4: R4- Repair solution.

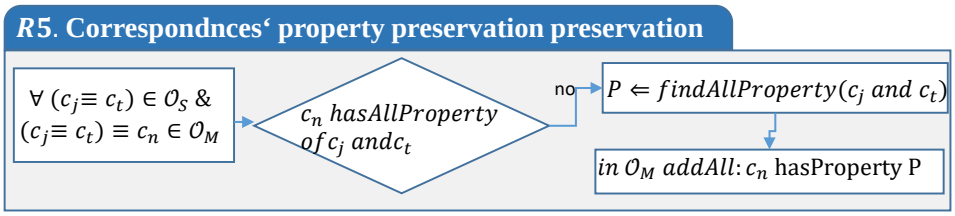

FIGURE A.5: R5- Repair solution.

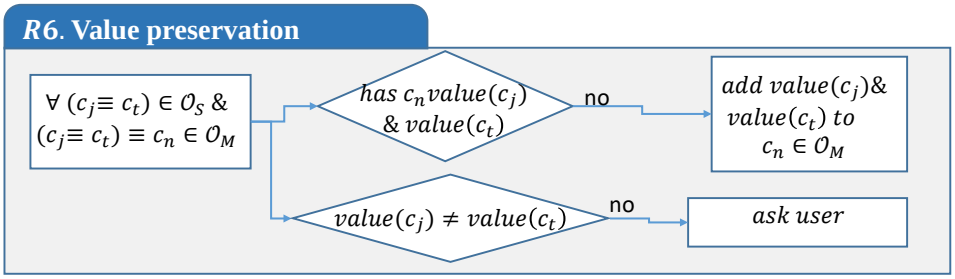

FIGURE A.6: R6- Repair solution.

- **Find:** For all corresponding entities with two different values, we check whether their integrated entity has both values or not, and if these both values have conflict, we mark them.
- **Repair:** If an integrated entity does not have both values of its corresponding entities, we set both values for the integrated one. But, if their values have a conflict with each other, to solve this, we need user interaction.

***R7.* Structure preservation:**

- **Find:** We check whether each class has the same ancestor as the source ontologies or not. If no, we mark them.
- **Repair:** For any classes c_j that does not have is-a relation to its respective parent in $\mathcal{O}_S$, 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 $\mathcal{O}_M$.

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

- **Find:** If there is any class c_j, which is redundant (duplicated) in the merged ontology, we mark it.
- **Repair:** For any marked class, keep one of them and delete the repeated one.

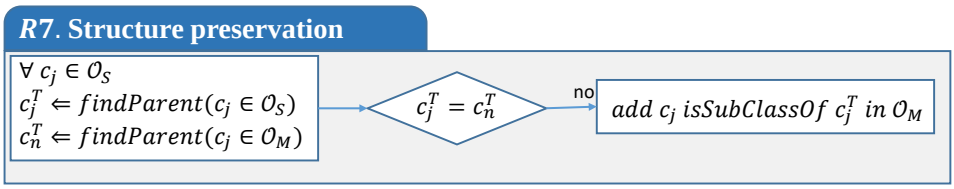

FIGURE A.7: R7- Repair solution.

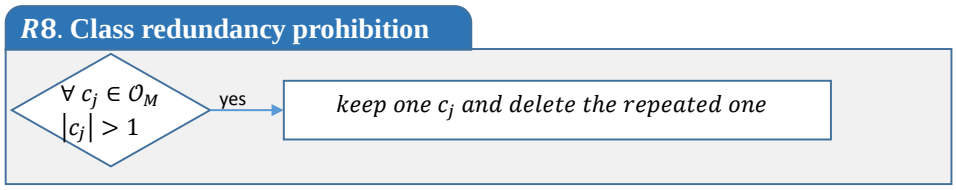

FIGURE A.8: R8- Repair solution.

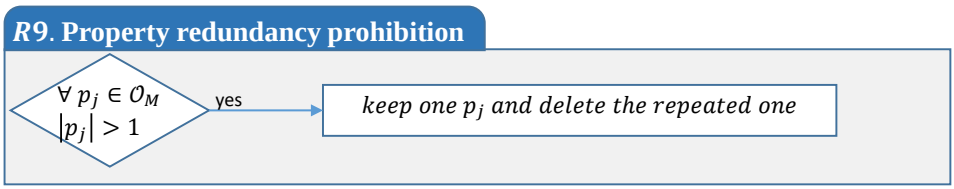

FIGURE A.9: R9- Repair solution.

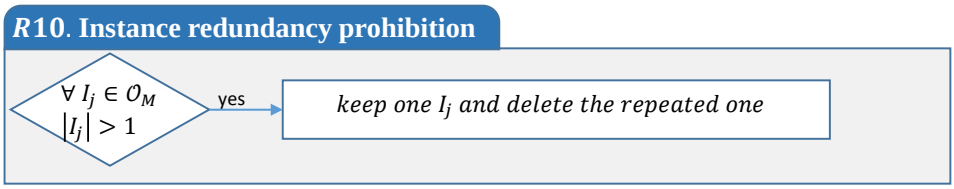

FIGURE A.10: R10- Repair solution.

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

- **Find:** If there is any property p_j, which is redundant (duplicated) in the merged ontology, we mark it.
- **Repair:** For any marked property, keep one of them and delete the repeated one.

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

- **Find:** If there is any instance I_j, which is redundant (duplicated) in the merged ontology, we mark it.
- **Repair:** For any marked instance, keep one of them and delete the repeated one.

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

- **Find:** For all entities belong to $\mathcal{O}_M$, we check whether they exist in $\mathcal{O}_S$. If no, we mark them.
- **Repair:** Any extra marked entity is deleted from $\mathcal{O}_M$.

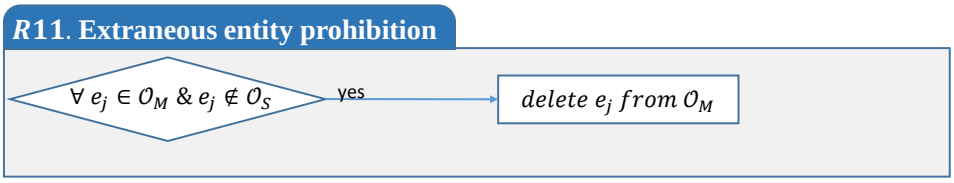

FIGURE A.11: R11- Repair solution.

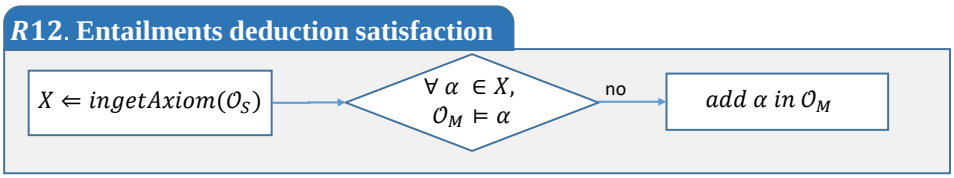

FIGURE A.12: R12- Repair solution.

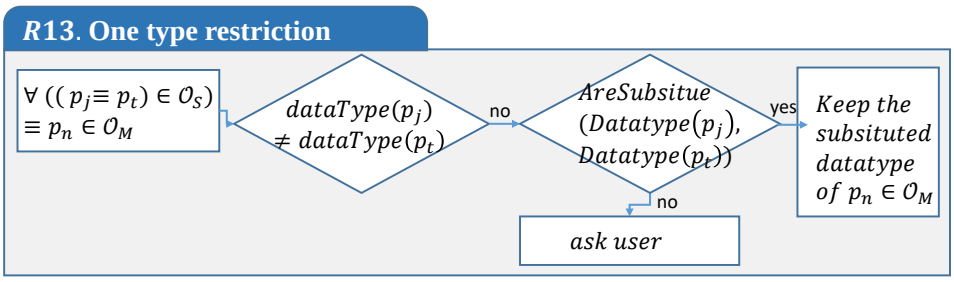

FIGURE A.13: R13- Repair solution.

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

- **Find:** 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 ontology. Then, we ask a reasoner to check whether the merged ontology can entail those axioms? If no, we mark it.
- **Repair:** We add those not-entailed axioms in $\mathcal{O}_M$.

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

- **Find:** 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.
- **Repair:** 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 overall one, i.e. type STRING for p_n. If no, we ask the user.

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

- **Find:** We check all constraint type: ObjectMaxCardinality, ObjectMinCardinality, ObjectExactCardinality, DataMaxCardinality, DataMinCardinality, DataExactCardinality, ObjectSomeValuesFrom,

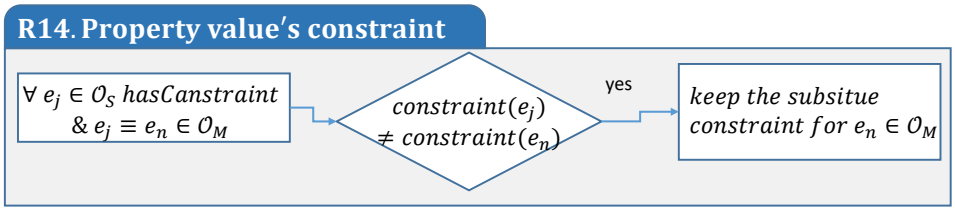

FIGURE A.14: R14- Repair solution.

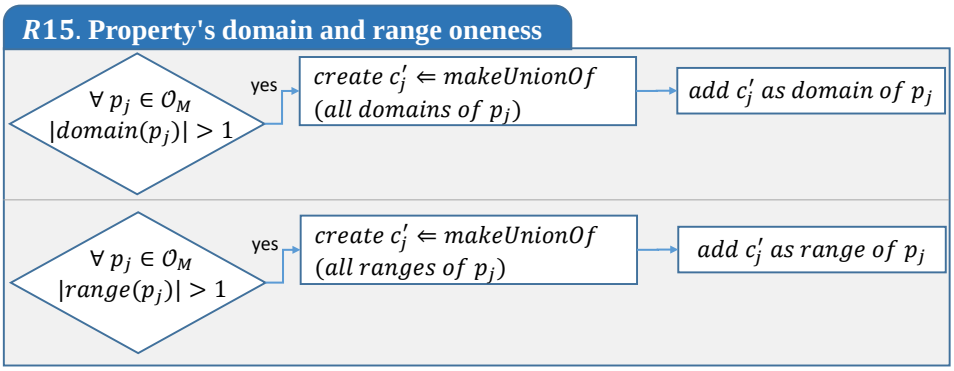

FIGURE A.15: R15- Repair solution.

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

- **Repair:** For any marked entity e_j, we keep the substitute value in $\mathcal{O}_M$, only.

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

- **Find:** If for a class c_j in the merged ontology, there is multiple domains or ranges ($|domainRange(c_j)| > 1$), we mark it.
- **Repair:** For each property p_j, which has multiple domains or ranges, 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.

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

- **Find:** There are two types of cycles: (i) 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. (ii) recursive-cycle: If during the visiting of all parents of a class, we visit more than one time a parent, we mark it as a cycle.
- **Repair:** We delete the related axiom to the self-cycle in the merged ontology. To repair the recursive-cycle, we need user interaction.

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

- **Find:** There are two types of cycles: (i) self-cycle: to detect this type of cycle, we check for any property p_j, this property should not appear

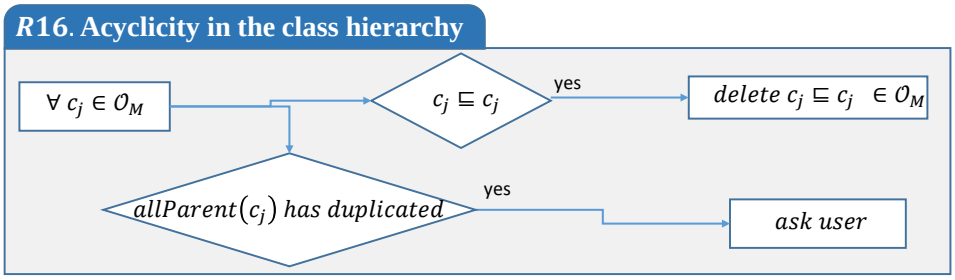

FIGURE A.16: R16- Repair solution.

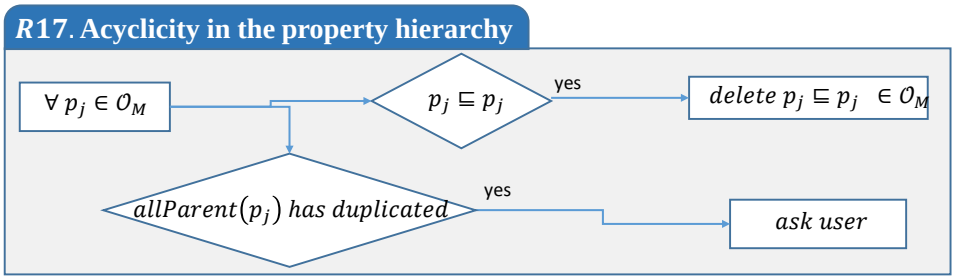

FIGURE A.17: R17- Repair solution.

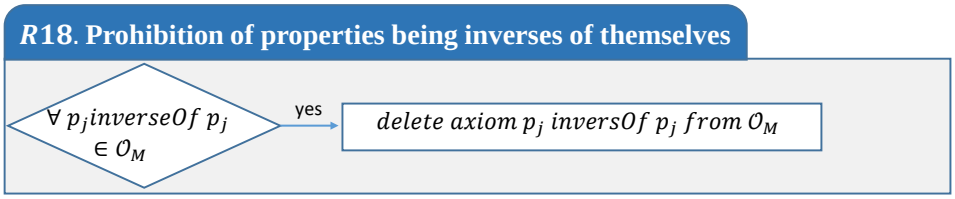

FIGURE A.18: R18- Repair solution.

as its subPropertyOf. If so, we mark it. (ii) recursive-cycle: if during the visiting subPropertyOf hierarchy for property p_j, we visit more than one time a property, we mark it as a cycle.

- **Repair:** We delete the related axiom to the self-cycle in the merged ontology. To repair the recursive-cycle, we need user interaction.

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

- **Find:** We check whether in the merged ontology there is a property that it is inverse of itself. If so, we mark it.
- **Repair:** we delete the related inverseOf axiom of the marked property in the merged ontology.

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

- **Find:** 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.
- **Repair:**

1. Find one of sub or superclass of c_j from $\mathcal{O}_S$, call it c_j^T

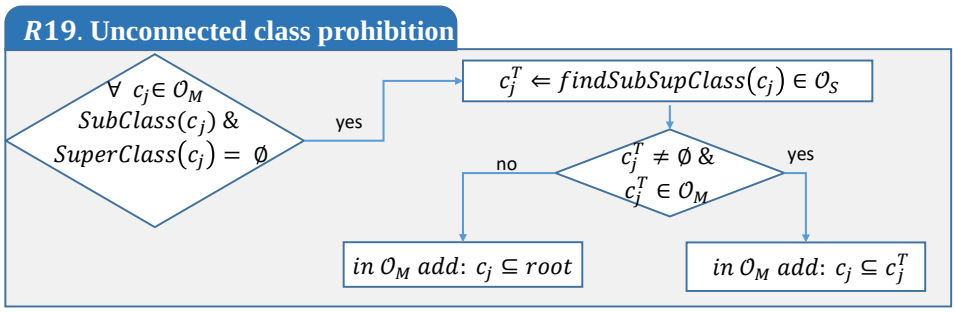

FIGURE A.19: R19- Repair solution.

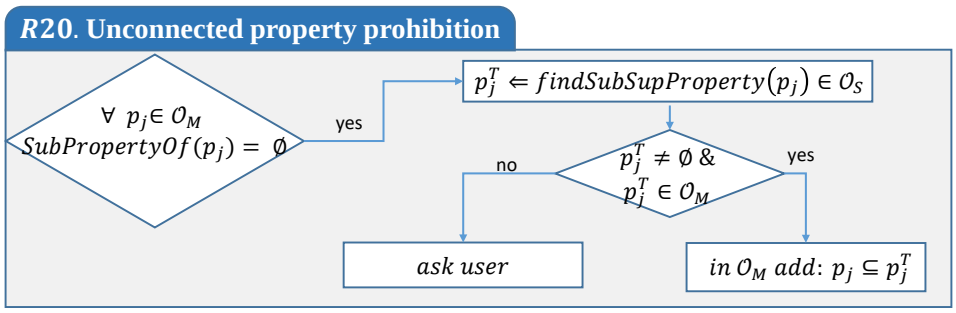

FIGURE A.20: R20- Repair solution.

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. Else, we add c_j to the root of the merged ontology.

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

- **Find:** If there is any property p_j which does not have any connections to the other properties in the subPropertyOf hierarchy, we mark it.
- **Repair:**
  1. Find one of sub or super property of p_j from $\mathcal{O}_S$, call it p_j^T
  2. If p_j^T is not null and exists in the merged ontology, we add p_j to p_j^T with a subPropertyOf relationship.
  3. Else, ask the user.

For more information, please refer to: Babalou, Samira, and Birgitta König-Ries. "GMRs: Reconciliation of Generic Merge Requirements in Ontology Integration." In SEMANTICS 2019 Poster and Demo Track.