An algebra for composing ontologies

Saket Kaushik[1], Csilla Farkas[2], Duminda Wijesekera[1], and Paul Ammann[1]

[1]Department of Information & Software Engineering, George Mason University, Fairfax, VA 22030, U.S.A, {skaushik|dwijesek|pammann}@gmu.edu
[2]Department of Computer Science and Engineering, University of South Carolina, Columbia, SC 29208 USA, farkas@cse.sc.edu

Abstract

Ontologies are used as a means of expressing agreements to a vocabulary shared by a community in a coherent and consistent manner. As it happens in the Internet, ontologies are created by community members in a decentralized manner, requiring that they be merged before being used by the community. We develop an algabra to do so in the Resource Discription Framework (RDF). To provide formal semantics of the proposed algebraic operators, we type a fragment of the RDF syntax.

1 Introduction

Ontologies, considered as specifications of conceptualizations are designed for the purpose of enabling knowledge sharing and reuse [15]. Rooted in Philosophy as a systematic theory of *existence* that specifies the entities that *exist* and relationships among existing objects, they are used in artificial intelligence and the world wide web as descriptions of domains of discourses that consists of entities and their relationships. Consequently, an *ontological commitment* is an agreement to use a vocabulary shared in a community in a coherent and consistent manner. Naturally, with the objective of the Semantic Web to have self describing, machine understandable and operable data, it relies on ontologies to specify domains of discourse.

Because data offered and consumed on the Semantic Web are considered *resources*, their semantics is expressed by specifying meta relations between web objects through ontologies, referred to as the *Resource Description Framework* (RDF), to be summarized shortly. Because the world wide web is a collection of independently maintained web sites without centralized control or agreement, their domain knowledge, captured by various domain experts, need to be combined, compared, contrasted and operated upon during web-based computations. Therefore, one of the core challenges identified for the Semantic Web involves decentralization and reuse of ontological conceptualization [18]

Several approaches exist that *combine* independently-developed ontologies. Klein's survey [18] presents an excellent overview of the problems in this domain. For instance, Klein discusses *mismatch* of conceptualization in multiple ontologies, where difference in syntax, expressivity, and semantics of the representation are problematic. As an example, consider the ontologies in figure 1. The three ontologies, named A, B and C, describe an auto-maker's products. Ontologies A and B are taken from Chalupsky's web page [6] with minor variations in property names, while ontology C has been composed by us for structural comparisons with his examples. Clearly, each differs from the other in its conceptualization. Several pertinent questions can be asked about the different conceptualizations, like – does any one ontology subsume another? Are the ontologies *structurally* consistent? Can they be combined in some way to yield a common single ontology? Many researchers have proposed ways to address such mismatches. In fact, there is an abundance of combinatorial techniques with seemingly similar terminology for the combination schemes, such as – alignment, merging, integrating, combining, mapping, articulation, translating, transforming, *etc.* However, they don't provide a comparison of their scheme with existing techniques.

In addition to mismatch, another important problem to be solved while combining ontologies is the automation of the process. Techniques that rely strongly on human input are able to score better on precision, but, are less scalable and labor intensive as compared to methods that rely strongly on automation. Automation usually requires approximations to resolve different types of mismatches. However, the problem of effectively measuring the difference between each approach still remains open. In this paper we propose an approach to solve this problem in the following way. First, we propose an algebra for composition of ontologies. Next, we compare existing combination techniques to elucidate how each of them differs from the ideal case.

Apart from mismatch of conceptualization, and from a practical viewpoint, a central problem in ontology combination is to translate multiple, independently-developed,

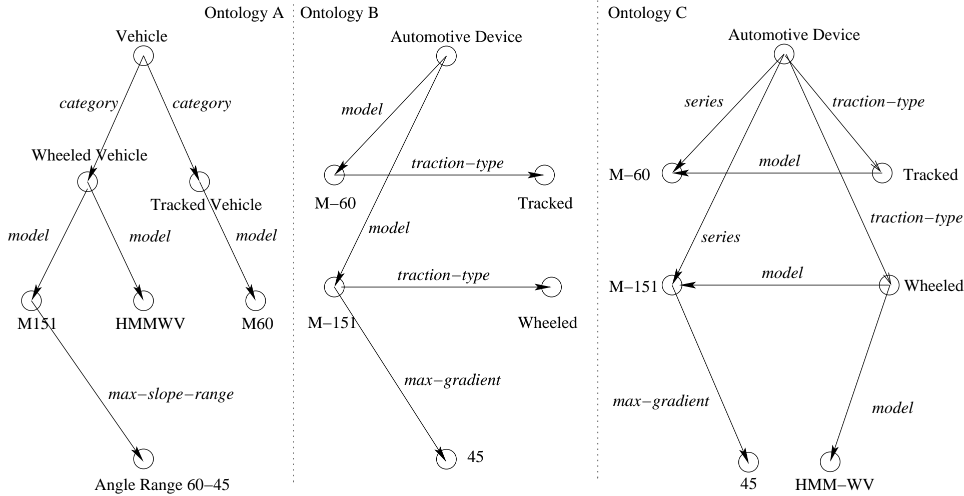

Figure 1: *Syntactic and conceptualization mismatch*

heterogenous sources of meta-information into a common syntax, or a *common data model*. Among other things, a correct translation should be semantically consistent and expressive enough to capture all the statements in the original ontology. The second important problem is to *merge* schema and instances that refer to the *same* real-world entity captured by multiple ontologies. Consider, for example, the city 'Arlington, TX' that is captured by an RDF ontology as '&r1' and by the CIA fact-book as '&r2'. These two instances must be identified in some way to represent the same real-world entity in order to combine information in the two ontologies that represent the domain of cities in the real-world.

Because RDF is the base language of most higher level Semantic Web languages, the common data model must at least support RDF syntactic elements. However, RDF allows considerable freedom of expression prompting many higher level languages to restrict features that it doesn't support. In our case, we model all features of RDF except the RDF collections, and, as will be shown shortly, our semantics is a higher order version of RDFS(FA) [27].

Several existing techniques solve the problem of merging ontologies. However, as will be shown, a majority of these techniques either involve an approximate merging (like ontomorph [7], Chimaera [20], FCA-merge [34], *etc.*) or are incomplete (like ONION [22]). Therefore, it is impossible to compare and contrast to what degree the ontologies have been merged. This is where our major contribution lies. We formally define a syntax and semantics for various combinations of ontologies that we are aware of in literature. This step involves introduction of several algebraic operators like union, intersection, merge, *etc.*

In order to formalize the algebra of combinators, we first present a recursive finite typing of RDF sans collections for expressing ontologies expressed in heterogenous languages. Second, we construct an well-founded algebraic scheme over a universe of ontologies that respects well known algebraic operations and identities. Third, we show how existing ontology combination schemes can be expressed in our algebraic language.

Because entity-relationship model of Peter Chen [9] has the objective of modeling entities and their relationships relevant to a data modeler, schematic descriptions of ontologies and database schema share some common properties. Therefore, in its formalization, merging independently-developed ontologies bears some similarity the the well studied database integration problem [2, 10, 29]. In addition, combination of databases in federations, also called federated database management systems (abbrev. FDBMS – for more details see [32]) have been well studied and can be leveraged upon. Finally, nested relations [24], originally designed for redundancy removal, bear stark similarities to *reified* RDF schema, and are relevant in our context. We develop our algebra with a view on solutions to well-known problems in the database literature.

The rest of the paper is organized as follows. In Section 2 and 3 we formalize RDF as a collection of higher typed objects and their properties. Section 4 defines homomorphisms between ontologies; in Section 5 we show how homomorphisms can be represented as ontologies in the same universe. In Section 6 we define basic algebraic operations and give their properties. Comparison of our algebra to existing algebras in the literature is given in Section 7. We conclude in Section 8.

2 The Resource Description Framework

The Semantic Web consists of a layered architecture [5] of web languages. The lowest layer of this stack is Unicode/URI layer on which the XML and `xmlschema` are built. The next layer has RDF(S), which defines both the

syntax and semantics of subsequent layers. Consequently, languages above the layer of RDF, such as OWL *etc.*, are built on top of RDF(S).

RDF specify meta-information about *resources*, *i.e.*, entities that can be uniquely identified, where the objectives of providing them are to enable machine processing and sharing meta-information about resources. Such meta information about resources are specified in RDF using binary properties between resources. RDF does so by using the syntax of *triples* where the subject (the first component of the triple) is related by the property (the second component of the triple) to the object (the third component). An RDF schema can be extended further through *reification,* that specify binary properties between triples. RDF(S) or RDF Schema is RDFs vocabulary description language. It has syntax to describe concepts and resources through meta-classes such as `rdfs:Class`, `rdf:type`, *etc.*, and relationships between resources through `rdf:property`. These meta classes are used to specify properties of user defined schema. RDF(S) vocabulary descriptions are written in RDF using a set of language primitives, described in [3].

3 Formalizing RDF

We formalize RDF as a collection of higher typed objects and binary relations among them, subjected to the restriction that containers and collections of RDF like `rdf:bag, rdf:alt`, *etc.* are not interpreted.

In our semantics, we interpret classes and properties as well founded sets in ZF set theory [19] with a name. Consequently, as in imperative programming languages, a class has a name, a location – given by its URI an universal address space, and elements. As a result, syntactic elements that are equal by name are also equal by location.

Definition 1 (Ontology [3]).

Primitive classes and properties: *Suppose U be a universe of objects. Let $C_0 \subseteq \mathcal{P}(U)$ and $P_0 \subseteq (C_0 \times C_0)$. Then any $c \in C_0$ is said to be a primitive class and any $p \in P_0$ said to be a primitive binary property over U.*

Classes and properties of finite type: *Suppose C_n and P_n are respectively classes and properties of type n. Then, classes and properties of type $(n+1)$ are those that satisfy $C_{n+1} \subseteq \mathcal{P}(C_n) \cup P_n$, and $P_{n+1} \subseteq (C_{n+1} \times C_{n+1})$. That is, in addition to classes of type n, classes of type $(n+1)$ may include properties of type n as elements. Collectively, the sequence of sets, i.e., $\bigcup_{i \geq 0} C_i$ is called a schema.*

Schema Naming: *We use a set of class names $\mathcal{CNAMES}$ for schema, and a function $\mathcal{CN} : \bigcup_{n \geq 0}(C_n \cup P_n) \mapsto \mathcal{CNAMES}$ - a naming function from $\bigcup_{n \geq 0}(C_n \cup P_n)$ to the name set $\mathcal{CNAMES}$. Also, $\mathcal{CNAMES} = \bigcup_{i \geq 0} CNAME_i$ where $CNAME_i$ is the set $C_i \cup P_i$ elements are mapped to.*

Instances: *A class c belonging to type C_n is represented $c :: C_n$ and a property p of type P_n is represented as $p :: P_n$. For any such class c and property p of type C_n, P_n (resp.), any member x being a member of $c :: C_n$ is represented as $x \in c :: C_n$ and pair of instances (y, z) related through the property $p :: P_n$ is represented as $(y, z) \in p :: P_n$. The set of instances of type n is named I_n and the set of all instances is named $\mathcal{I}$. Note that whenever the type is apparent, we drop '::' from the representation.*

Instance Naming: *We use a set of names $\mathcal{INAMES}$ for instances, and a function $\mathcal{IN} : \mathcal{I} \mapsto \mathcal{INAMES}$ said to be a naming function from $\mathcal{I}$ to $\mathcal{INAMES}$. Also, $\mathcal{INAMES} = \bigcup_{i \geq 0} INAME_i$ where elements in $I_i \cup P_i$ are mapped to the set $INAME_i$.*

Ontology *An ontology $\mathcal{O}$ is a tuple $\langle \mathcal{C}, \mathcal{P}, \mathcal{CN}, \mathcal{I}, \mathcal{IN} \rangle$ where $\mathcal{C} = \bigcup_{n \geq 0} C_n$ is a set of classes, $\mathcal{P} = \bigcup_{n \geq 0} P_n$ are the properties and $\mathcal{I}$ is the set of instances.*

Definition 1 defines Ontologies consisting of entities and relationships among them as recursively defined classes and relationships respectively. The base case of the recursion defines the *base* class and their *properties*. In that base case of the recursive definition, the collection of base classes C_0 are given axiomatically. That is they can be chosen without any other restrictions, and have to be given a name in some arbitrarily chosen, but fixed thereafter, collection of symbols referred to as $\mathcal{CNAME}$. In case of RDF, these are chosen to be the collection of URIs. Every class in C_0 has the semantics as some collection of named objects chosen out of a universe U with a name chosen from a class of instances $\mathcal{INAME}$. Relationships between classes in C_0 are defined as a collection of axiomatically chosen binary relationships given by P_0. That is, properties of P_0 are chosen binary relationships between objects of C_0 with names chosen from $\mathcal{CNAMES}$.

The inductive step of definition 1 chooses next class of objects - that is C_1 named subsets - to be all objects of C_0 and all named binary properties of P_0. That makes every property of P_0 an object of the next class C_1. The properties of second stage - that is P_1 - are defined as suitably chosen named binary relations among C_1. This application of the inductive step follows the same pattern for all stages.

This definition has several desirable properties. The first is that, because properties of a (lower) level become classes of the immediately higher class, properties among such properties - i.e. higher level properties - can be specified in the next level. Secondly, because RDF does not explicitly define classes and properties, and allows the user to define them, the type definition does not provide type constructors. Thirdly, when a class c of C_n becomes a class of C_{n+1}, the name that c is assigned is carried over to the next level. Similarly, a property p at level P_n is also on object in C_{n+1} and therefore has a name assigned at

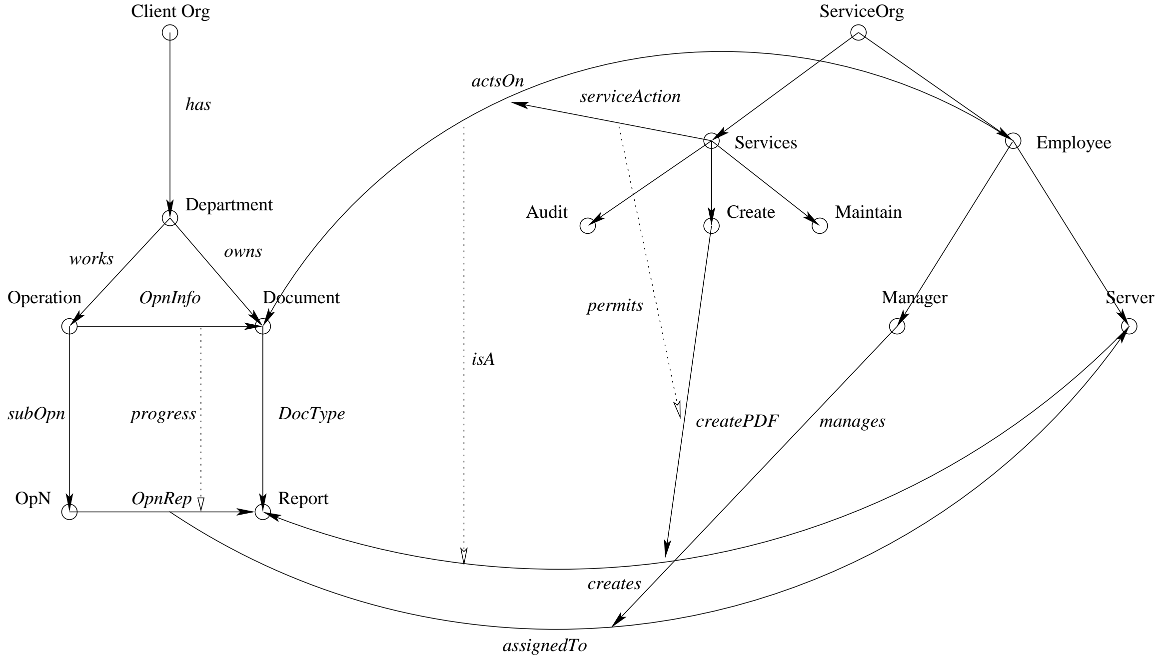

Figure 2: *Document service schema*

P_n that we take at the next level as well. Fourthly, due to the relationships $C_n \subset C_{n+1}$ and $P_n \subset C_{n+1}$, any class c of C_n and a property p of P_n are also classes of C_{n+1}. We now provide some examples of our definition.

Example 1. *Consider a document service schema in which client organizations buy document creation, handling and maintenance services from service organizations for the operations performed by departments in the client organization. Created documents are owned by the creating department. Document handling itself is organized into a host of services like report creation, maintenance and validation. A partial schema (without maintenance and audit relations) is pictured in figure 2. This schema amply demonstrates the ideas expressed in the definitions above, and is discussed next.*

The figure pictures following type 0 class names, i.e., the set $\mathcal{CN}(C_0)$ = {ClientOrg, Department, Operation, Document, OpN, Report, ServiceOrg, Services, Employee, Audit, Create, Maintain, Manager, Server}. These are identified by the nodes (small circles) in the figure, i.e., this set is constructed from names given to nodes in figure 2. The type C_0 nodes represent base ontological concepts. They are related to each other through type P_0 properties, i.e., binary relations that relate two base concepts. Type P_0 properties in the diagram are named from the set (i.e., $\mathcal{CN}(P_0)$) {has, works, owns, OpnInfo, subOpn, OpnRep, DocType, serviceSet, employees, creates, actsOn}.

Next is the beginning of the recursive step in the definition of recursive countable types. With types C_0 and P_0 defined, we begin constructing types C_1 and P_1. Type C_1 classes are the union $C_0 \cup P_0$. In other words, at level 1 all schema elements from the previous level are treated as concepts. Therefore, type P_1 properties can relate any subsets of elements over type C_0 or type P_0. Thus, all possible P_1 properties are a subset of $(C_1 \times C_1)$. The set P_1 in the shown schema includes the following property names: {progress, assignedTo, createPDF, serviceAction, isA}. Similarly, elements of higher types are defined recursively. We list the different (named) typed elements next:

Type C_0

C_0 = *{ClientOrg, Department, Operation, Document, OpN, Report, ServiceOrg, Services, Employee, Audit, Create, Maintain, Manager, Server}*

Type P_0

- *has:*$ClientOrg :: C_0 \mapsto Department :: C_0$
- *works:*$Department :: C_0 \mapsto Operation :: C_0$
- *owns:*$Department :: C_0 \mapsto Document :: C_0$
- *OpnInfo:*$Operation :: C_0 \mapsto Document :: C_0$
- *subOpn:*$Operation :: C_0 \mapsto OpN :: C_0$
- *OpnRep:*$Opn :: C_0 \mapsto Report :: C_0$
- *DocType:*$Document :: C_0 \mapsto Report :: C_0$
- *serviceSet:*$ServiceOrg :: C_0 \mapsto Services :: C_0$
- *employs:*$ServiceOrg :: C_0 \mapsto Employee :: C_0$
- *creates:*$Server :: C_0 \mapsto Report :: C_0$
- *actsOn:*$Employee :: C_0 \mapsto Document :: C_0$

Each property of type P_0 relates concepts of type C_0. Thus, 'has' identifies departments in a client organization; 'works' relates operations to departments; while 'employs' relates employees to the service organization.

Type C_1

$C_1 = \mathcal{P}(C_0) \cup P_0$.

Type P_1

- *progress:*$OpnInfo :: P_0 \mapsto OpnRep :: P_0$
- *assignedTo:*$OpnRep :: P_0 \mapsto Server :: C_0$
- *createPDF:*$Create :: C_0 \mapsto creates :: P_0$
- *serviceAction:*$Services :: C_0 \mapsto actsOn :: P_0$
- *isA:*$actsOn :: P_0 \mapsto creates :: P_0$

Type $P_i (i > 0)$ properties have a different utility, i.e., they relate 'constructed' concepts. For example, the property 'progress' relates the binary relation between 'operation' and 'document', i.e., 'OpnInfo', with the binary relation between a suboperation and its report. In other words, here it is specializing the property 'OpnInfo'. Similarly, the property 'assignedTo' relates the binary relation between operation and its report to the report creator.

Type C_2

$C_2 = \mathcal{P}(C_1) \cup P_1$.

Type P_2

- *manages:*$Manager :: C_0 \mapsto assignedTo :: P_1$
- *permits:*$serviceAction :: P_1 \mapsto createPDF :: P_1$

Type P_2 properties shown here are 'manages' and 'permits'. 'manages' relates a C_0 element (Manager) to a P_1 relation (assignedTo). Here, the intention is to capture the scenario where 'Servers' are not working for a 'Manager' exclusively, instead, they work with different managers on different document services. Similarly, 'permit' property relates 'serviceAction' property to a create action, i.e., 'createPDF', in the sense that a particular service action must first be allowed.

Definition 1 allows for expression of higher types, i.e., $C_3, P_3, C_4, P_4, \ldots$. However, the given schema is restricted to type 2 elements. Instances of the schema are constructed using the `rdf:type` *property but are not pictured in figure 2. Together the schema and its instance complete an ontology.*

Our definition of an ontology may appear to be different from RDF specification [16] of an ontology. However, by allowing the set of instances, *i.e.*, $\mathcal{I}$ to be empty, the two definitions can be made to converge. Thus, ontologies are named nested unary and binary schema [24] sans *keys*. Because ontologies are sometimes modeled as connected graphs with $c :: C_i, i \geq 0$ elements as nodes, $p :: P_i$ as edges and $\mathcal{I}$ as the set of instances, paths in ontologies have been defined. Our definition of ontologies can be used to define (typed) paths as follows.

Definition 2 (Path).

Directed Typed Path *An n-edged directed typed path is a triple $\langle u_1, \pi_t, u_{n+1}\rangle$ where $u_i :: C_j$ (for some $j \geq 0$), $i \in [1, n+1]$ and π_t is a sequence of n edges given by $\langle \mathcal{CN}(e_1), \ldots, \mathcal{CN}(e_n)\rangle$, where $e_k :: P_j, k \in [1, n]$ and $e_1 = u_1 \mapsto u_2, \ldots, e_n = u_n \mapsto u_{n+1}$. Each property in a path is related through $\prec$ relation, i.e., $e_i \prec e_{i+1}$. The reflexive transitive closure of $\prec$ ($\prec^*$) arranges every pair of properties in π_t in a partial order.*

Directed path *An n-edged directed path is a triple $\langle u_1, \pi_d, u_{n+1}\rangle$ where $u_i \in \bigcup_j c :: C_j, i \in [1, n+1]$ and π_d is a sequence of n edges given by $\langle \mathcal{CN}(e_1), \ldots, \mathcal{CN}(e_n)\rangle$, where $e_k \in p :: P_j, k \in [1, n]$ and $e_1 = u_1 \mapsto u_2, \ldots, e_n = u_n \mapsto u_{n+1}$*

We abuse the syntax at times by relating paths as $\langle \mathcal{CN}(u_1), \pi, \mathcal{CN}(u_{n+1})\rangle$ instead of $\langle u_1, \pi, u_{n+1}\rangle$.

3.1 Limited RDF Semantics

In this section we briefly describe our semantics for named ontologies of finite types. Our semantics is closest to those of RDFS(FA) [27], where we build a hierarchy of named sets. In order to do so we start with a universe, *i.e.*, a set of URI's (denoted URI) and other constants, say B as *urelements*, *i.e.*, those atomic elements that do not have any further set theoretical structure to them)[1, 19]. Our hierarchical universe is built as follows:

$\mathbf{C_0}$ **and** $\mathbf{P_0}$**:**

$$C_0 = \{(u, b) : u \in URI \text{ and } b \in B\}$$
$$P_0 = \{(u, (B, B')) : u \in URI \text{ and } B, B' \subseteq Base\}$$

$\mathbf{C_{n+1}}$ **and** $\mathbf{P_{n+1}}$**:** Suppose $\mathbf{C_n}$ and $\mathbf{P_n}$ have been defined. Let $U_n = \{x : \exists u \in URI(u, x) \in C_n\}$. Then define

$$C_{n+1} = \{(u, B) : u \in URI \text{ and } B \subseteq \mathcal{P}(U_n)\}$$
$$P_{n+1} = \{(u, (B, B')) : u \in URI \text{ and } B, B' \subseteq \mathcal{P}(U_n)\}$$

Notice that every strata of the universe consists of ordered pairs, where the first component is a URI, denoting the name, and the second component is a set. Consequently, in the inductive step U_n recreates the base elements from the previous step. The level of the nestings in the construction of the set indicates the level of the universe. That is, a set with n nesting is at level n. Also notice that, stripped out of the name, a class at level n becomes an element at level $n + 1$. The main reason for introducing names (somewhat artificially, we agree) is to make the distinction between name (intensional) equality and extensional equality, because in ZF set theory, set equality is extensional. We use this stratified named universe to interpret our ontology and RDF syntax as follows:

Definition 3 (Semantic mapping $[\![\ ]\!]$).

Universe:

1. *Suppose U is the universe as defined in definition 1.*

2. *Let $[\![\ ]\!]_u : U \mapsto B$ be the surjective mapping that maps the constants in the syntax to a set B of urelements over which the syntax is interpreted.*

3. *Let $[\![\ ]\!]_{Iname} : \mathcal{INAMES} \mapsto URI$ be a mapping of RDF names to URIs.*

4. *Let $[\![\ ]\!]_{Iname} : \mathcal{CNAMES} \mapsto URI$ be a mapping of RDF names to URIs.*

Mapping instances: *Every instance i with name a is mapped as $[\![(a,i)]\!] = ([\![i]\!]_{INAMES}, [\![i]\!]_u)$.*

Mapping classes: *For every class $c :: C_n$ named a is mapped as $[\![(a,c)]\!] = ([\![a]\!]_{CNAMES}, [\![c]\!])$ where $[\![c]\!]$ is constructed by replacing every $u \in U$ in c with $[\![u]\!]$.*

Mapping properties: *For every property $p :: P_n$ named a is mapped as $[\![(a,p)]\!] = ([\![a]\!]_{CNAMES}, [\![p]\!])$ where $[\![p]\!]$ is constructed by substituting every $u \in U$ in p with $[\![u]\!]$.*

Interpreting `rdf:type`: *We say that $[\![$ rdf:type $x = y]\!]$ iff $x = (a,b)$, $y = (p,q)$ and $[\![b]\!] \in [\![q]\!]$.*

Interpreting `rdfs:subClassOf`: *We say that $[\![x$ rdfs:subClassOf $y]\!]$ iff $x = (a,b)$, $y = (p,q)$ and $[\![b]\!] \subseteq [\![q]\!]$.*

Interpreting names: *The name of an object or a property x is defined as the first coordinate of $[\![x]\!]$.*

Interpreting intensional equality: *x is said to be intensionally equal to y (written $x =_{int} y$) iff $[\![x]\!] = [\![y]\!]$*

Interpreting schema equality: *x is said to be structurally equal to y, written $x = y$ iff $(a,b) = [\![x]\!]$, $(p,q) = [\![y]\!]$ and $b = q$.*

The reason we say that our interpretation of RDF is limited because, as [12], stratified RDF semantics is more restrictive than RDF MT [27].

4 Homomorphisms

This section defines homomorphisms between ontologies. That is, structure preserving mapping between ontologies. We do so in order to analyze various existing definitions of *connections* in the literature in light of structure preserving mappings, and calibrate them against the latter with respect to preserving structure. For example $\mathcal{E}$-connections [14], that introduce subsumption relationships between ontologies do not preserve all structural properties, although commonly used functional mappings such as alignment, translation, *etc.*, preserve some structure. Our definition of ontological homomorphisms follows.

Definition 4 (Ontological homomorphism).

Schema homomorphism *An ontological schema homomorphism from ontology $A = (\mathcal{C}^A, \mathcal{P}^A, \mathcal{CN}^A, \mathcal{I}^A, \mathcal{IN}^A)$ to ontology $B = (\mathcal{C}^B, \mathcal{P}^B, \mathcal{CN}^B, \mathcal{I}^B, \mathcal{IN}^B)$ is a pair of mappings (f_C, f_P) that satisfy:*

1. *$f_C(C_n^A) \subseteq C_n^B$ and*
2. *$f_P(P_n^A) \subseteq P_n^B$ satisfy*

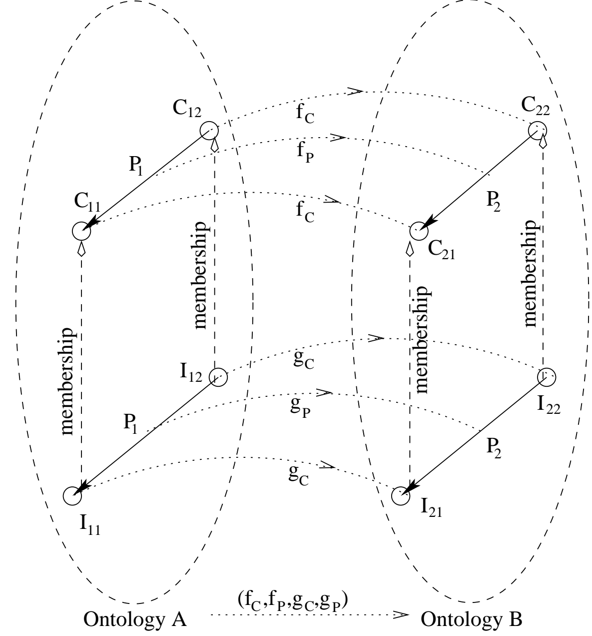

Figure 3: *Schema and instance homomorphisms*

3. *If p^A::(c^A, c'^A) then $f_P(p^A)$::$(f_C(c^A), f_C(c'^A))$.*

Furthermore, if $\mathcal{CN}(c^a) = \mathcal{CN}(f_C(c^a))$ for every $c^a \in C^A$ and $\mathcal{CN}(p^a) = \mathcal{CN}(f_P(p^a))$ for every $p^a \in P^A$, then we say that it is a name preserving homomorphism, or an intensional schema homomorphism.

Instance homomorphism: *An ontological instance homomorphism is a quadruple (f_C, f_P, g_C, g_P) where (f_C, f_P) is a schema homomorphism and g_C, g_P are functions that satisfy:*

1. *$g_C(x) \in f_C(c :: C_n^A)$ for every element $x \in c :: C^A)$ for some class c,*
2. *$(g_C(x), g_C(y)) \in f_P(p :: P_n^A)$ for every pair of elements $(x,y) \in p :: P_n^A$.*

Furthermore, if (f_C, f_P) is an intensional schema homomorphism that satisfy $\mathcal{IN}(x^a) = \mathcal{IN}(g_C(x^a))$ for every $x^a \in c^a :: C^A$, then we say that it is a name preserving homomorphism or an intensional instance homomorphism.

Using standard terminology, we say that a schema homomorphism is class-wise, property-wise or completely injective iff f_C, f_P or both f_C and f_P are injective mappings, respectively. Analogously, an instance homomorphism is said to be injective if (f_C, f_P) is injective as a schema homomorphism and the mappings g_C, g_P or both g_C and g_P are injective. An analogous notation is used for surjective homomorphisms. A homomorphism to itself is referred to as an endomorphism.

As stated in definition 4, an ontological schema homomorphism maps classes and properties of a source ontology to classes and properties of the target ontology with the same type. An instance homomorphism maps elements of the source class to elements of a target class and

a property instances of the source ontology is mapped to the same type of property instance in the target ontology so that if a pair of elements in a source class satisfy the property instance in the source ontology, then the image of the elements of the class instances satisfy the image of the property instance in the target ontology. Notice that intensional or name preserving homomorphisms are also homomorphisms that preserve the names in addition.

As an explanation of the definition of homomorphism, consider the ontology sketched in Figure 3. The homomorphism maps the classes and properties of the source ontology A to those of the the target ontology B using the function f_C, f_P respectively and their instances using functions g_C, g_P respectively as follows:

$\mathbf{f_C}$: $f_C(C_{11}) = C_{21}$, $f_C(C_{12}) = C_{22}$,

$\mathbf{f_P}$: $f_P(P_1) = P_2$,

$(\mathbf{g_C}, \mathbf{g_P})$: Suppose $I_{11} \in C_{11}, I_{12} \in C_{12}$, $I_{21} \in C_{21}$, and $I_{22} \in C_{22}$ and the only instances of these classes. Then (g_C, g_P) maps instances and their properties as $I_{21} = g_C(I_{11})$, $I_{22} = g_C(I_{12})$ and $g_P(I_{11}, I_{12} = (I_{21}, I_{21})$.

We now give an example homomorphism form the ontologies in figure 1.

Example 2 (homomorphisms). *Consider the ontologies in figure 1 describing automotive devices. All the three ontologies A, B and C describe the same real-world entities (schema component of the three ontologies is missing in the figure shown, but is easy to construct or visualize. Therefore we ignore it in the rest of this example). Assuming that they are independently developed, location-wise, concepts, properties and instances in A, B and C are disjoint. We attempt to reconcile the difference in conceptualization (and syntax if need be) with a homomorphism between A and B.*

First, we note that C_0^A members 'Vehicle', 'Tracked Vehicle' and 'M 60' are corresponding to 'Automotive Device', 'Tracked' and 'M-60' classes in C_0^B. However, in A 'model' property is expressed as $\langle Tracked\ Vehicle, M60 \rangle$, whereas in B corresponding elements are related through property 'traction-type' as $\langle M-60, Tracked \rangle$. Clearly, a morphism between the two ontologies will not preserve structure between the two ontologies since the direction of the property is reversed. However, it is possible to construct a homomorphism from A to C as follows. (We only provide (g_C, g_P) morphisms here).

$$
\begin{aligned}
g_C(\textit{Vehicle}) &= \textit{Automotive Device}\\
g_C(\textit{Wheeled Vehicle}) &= \textit{Wheeled}\\
g_C(\textit{Tracked Vehicle}) &= \textit{Tracked}\\
g_C(\textit{M151}) &= \textit{M-151}\\
g_C(\textit{M60}) &= \textit{M-60}\\
g_C(\textit{HMMWV}) &= \textit{HMM-WV}\\
g_C(\textit{Angle Range 60-45}) &= 45\\
g_P(\textit{category}) &= \textit{traction-type}\\
g_P(\textit{model}) &= \textit{model}\\
g_P(\textit{max-slope-range}) &= \textit{max-gradient}
\end{aligned}
$$

We now consider equality between ontologies.

Definition 5 (Equality).

Schema equality *Two ontologies $A = (\mathcal{C}^A, \mathcal{P}^A, \mathcal{CN}^A, \mathcal{I}^A, \mathcal{IN}^A)$ and $B = (\mathcal{C}^B, \mathcal{P}^B, \mathcal{CN}^B, \mathcal{I}^B, \mathcal{IN}^B)$ are said to be structurally equal - expressed $A \equiv_s B$ - iff there is a schema isomorphism (f_C, f_P) from A to B.*

Instance equality *Two ontologies $A = (\mathcal{C}^A, \mathcal{P}^A, \mathcal{CN}^A, \mathcal{I}^A, \mathcal{IN}^A)$ and $B = (\mathcal{C}^B, \mathcal{P}^B, \mathcal{CN}^B, \mathcal{I}^B, \mathcal{IN}^B)$ are said to be instance-wise equal - expressed $A \equiv_I B$ - iff there is an instance isomorphism (f_C, f_P, g_C, g_P) from A to B.*

Name equality *Two structurally equal ontologies $A = (\mathcal{C}^A, \mathcal{P}^A, \mathcal{CN}^A, \mathcal{I}^A, \mathcal{IN}^A)$ and $B = (\mathcal{C}^B, \mathcal{P}^B, \mathcal{CN}^B, \mathcal{I}^B, \mathcal{IN}^B)$ through the isomorphism (f_C, f_P) are said to be equal by name - expressed $A \equiv_N B$ - iff for every class $c \in \mathcal{C}^A$, $\mathcal{CN}(c) = \mathcal{CN}(f_C(c))$.*

As stated, two ontologies are schema equal if classes and properties between them are isomorphic. In addition when the schemas and relations are populated by instances are bijective, then the two ontologies are said to be instance equal. If a schema isomorphism between two ontologies preserve names, the they are said to be name equal.

5 Homomorphisms as ontologies

Ontological homomorphisms capture knowledge about similarities between concepts, properties and instances across ontologies within a universe. Consequently, they specify or express relationships between existing ontologies or concepts and properties between them, and therefore can be represented in ontologies [15]. In this section we show how this can be achieved by constructing homomorphisms as ontologies.

Definition 6 (Homomorphic ontology). *Given ontologies $A = (\mathcal{C}^A, \mathcal{P}^A, \mathcal{CN}^A, \mathcal{I}^A, \mathcal{IN}^A)$, $B = (\mathcal{C}^A, \mathcal{P}^B, \mathcal{CN}^B, \mathcal{I}^B, \mathcal{IN}^B)$, and an instance homomorphism (f_C, f_P, g_C, g_P) between them, their homomorphic ontology $H^{AB} = (C, P, CN, I, IN)$ is defined as:*

C$_0$: *The set of primitive concepts is given by* $(domain(f_C \cup f_P) \cap (C_{0,A} \cup P_{0,A})) \cup (range(f_C \cup f_P) \cap (C_{0,B} \cup P_{0,B}))$

P$_0$: *The set of primitive properties* $P_0 \subseteq (C_0 \times C_0)$ *where*

$$P_0 = \begin{cases} \{p | p \in (f_C \cup f_P) \wedge p \subset C_0 \times C_0\} \\ \cup \{\pi_A | \exists \langle a, \pi_A, b\rangle \in A \wedge a, b \in C_0\} \\ \cup \{\pi_B | \exists \langle a, \pi_B, b\rangle \in B \wedge a, b \in C_0\} \end{cases}$$

C$_{n+1}$: $C_{n+1} \subseteq \mathcal{P}(C_n) \cup P_n$

P$_{n+1}$: $P_{n+1} \subseteq (C_{n+1} \times C_{n+1})$

CN *Schema naming function is defined as CN:* $\mathcal{CN}^A \cup \mathcal{CN}^B \cup \{(f_C \cup f_P) \mapsto sameAs, \pi \mapsto related, p :: P_n\ (n > 0) \mapsto samePath\}$

I: *Instances of classes and properties are provided by the original ontologies, i.e.,* $x \in c :: C_{0,A} \wedge c :: C_0 \rightarrow x \in c :: C_0$. *Similarly, B contributes instances for* C_0. *Property instances are then defined in the standard manner.*

IN: *Instance naming is the union of the respective instance naming function and new property name mappings, i.e., IN:* $\mathcal{IN}^A \cup \mathcal{IN}^B \cup \{(g_C \cup g_P) \mapsto sameAs, \pi \mapsto related, p :: P_n\ (n > 0) \mapsto samePath\}$

Definition 6 precisely captures the knowledge regarding *alignment* of ontologies, as discussed in the literature [18, 20, 22, 26]. Classes and properties in ontology A that are mapped to, respectively, classes and properties in ontology B are cast as type C_0 classes in the homomorphic ontology H^{AB}. Mappings (f_C, f_P) (resp. (g_C, g_P) for instances) constitute the level 0 binary properties between these level 0 concepts. Note that the information that pairs of type C$_0$ classes in ontology A (resp. B) that are connected through a path π_A (resp. π_B) are connected in H^{AB}. This captures structure from the original ontology into the homomorphic ontology, allowing us to mark the classes as 'related'. Paths need not traverse only through classes that are mapped. Further, similar paths, *i.e.*, π_A, π_B may be related through a binary property, named 'samePath' – a P$_1$ level property, and so on.

6 Algebraic operators and their properties

Sometimes independently developed and maintained ontologies need to be merged, suitably composed, compared and contrasted in many ways in order do web-based computations. In this section, we define these operations and show their utility.

As ontologies consist of classes with relationships among them and elements that populate the aforementioned, the first kind of operations we explore are the set theoretical ones such as union, intersection, complement, *etc.* However, due to having instances of classes and properties, all these operations exists at the schema and instance levels. Another difference is the distinction between extensional and intensional equality, because RDF uses the latter, and set theory uses the former, any set theoretical interpretation of algebraic operations need to find suitable encodings to account for both.

Definition 7 (Set-based intensional operations). *Suppose* $A = (\mathcal{C}^A, \mathcal{P}^A, \mathcal{CN}^A, \mathcal{I}^A, \mathcal{IN}^A)$ *and* $B = (\mathcal{C}^B, \mathcal{P}^B, \mathcal{CN}^B, \mathcal{I}^B, \mathcal{IN}^B)$ *are two ontologies. Then define the*

Intensional union of (A ∪ B) as:

1. $\mathcal{C}_i^{A\cup B} = \mathcal{C}_i^A \cup \mathcal{C}_i^B$ *for every type i. That is, for every type i, all classes in* $\mathcal{C}_i^A$ *and* $\mathcal{C}_i^B$ *are taken together for* $\mathcal{C}_i^{A\cup B}$
2. $\mathcal{P}_i^{A\cup B} = \mathcal{P}_i^A \cup \mathcal{P}_i^B$ *for every type i. That is, for every type i, all properties in* $\mathcal{P}_i^A$ *or* $\mathcal{P}_i^B$ *are taken together for* $\mathcal{P}_i^{A\cup B}$
3. $\mathcal{CN}^{A\cup B} = \mathcal{CN}^A \cup \mathcal{CN}^B$. *That is, the names of the classes and properties given from the components A and B are the same taken to be the names assigned in the ontology* $A \cup B$.
4. $x \in c :: C_n^{A\cup B}$ *iff* $x \in c :: C_n^A$ *or* $x \in c :: C_n^B$. *That is, an element belongs to a class c in* $\mathcal{C}_n^{A\cup B}$ *iff it belongs to a constituent class of c in* $\mathcal{C}_n^A$ *or* $\mathcal{C}_n^B$.
5. $(x, y) \in p :: P_n^{A\cup B}$ *iff* $(x, y) \in p :: P_n^A$ *or* $(x, y) \in p :: P_n^B$ *That is, a subject object pair* (x, y) *satisfy a property p in* $\mathcal{P}_n^{A\cup B}$ *iff* (x, y) *satisfy the same property in either* $\mathcal{P}_n^A$ *or* $\mathcal{P}_n^B$.

Intensional intersection (A ∩ B) as:

1. $\mathcal{C}_i^{A\cap B} = \mathcal{C}_i^A \cap \mathcal{C}_i^B$ *for every type i. That is, for every type i, all classes common to* $\mathcal{C}_i^A$ *and* $\mathcal{C}_i^B$ *are in* $\mathcal{C}_i^{A\cup B}$
2. $\mathcal{P}_i^{A\cap B} = \mathcal{P}_i^A \cap \mathcal{P}_i^B$ *for every type i. That is, for every type i, properties common to* $\mathcal{P}_i^A$ *and* $\mathcal{P}_i^B$ *are in* $\mathcal{P}_i^{A\cup B}$.
3. $\mathcal{CN}^{A\cup B} = \mathcal{CN}^A \cap \mathcal{CN}^B$. *That is, the names of the classes and properties given from the components A and B are taken as the names in* $A \cap B$.
4. $x \in c :: C_n^{A\cap B}$ *iff* $x \in c :: C_n^A$ *and* $x \in c :: C_n^B$. *That is, an element belongs to a class c in* $\mathcal{C}_n^{A\cap B}$ *iff it belongs to the same class c in* $\mathcal{C}_n^A$ *and* $\mathcal{C}_n^B$.
5. $(x, y) \in p :: P_n^{A\cup B}$ *iff* $(x, y) \in p :: P_n^A$ *and* $(x, y) \in p :: P_n^B$ *That is, a subject object pair* (x, y) *satisfy a property p in* $\mathcal{P}_n^{A\cap B}$ *iff* (x, y) *satisfy the same property in both ontologies* $\mathcal{P}_n^A$ *and* $\mathcal{P}_n^B$.

Definition 7 defines intensional union and intersection. They mimick the basic set theoretical operations of union and intersection, interpreted as a class or a property belong to the union or the intersection iff they belong to either or both constituents, but with a difference. The difference is that two classes in the constituents are considered the same iff they are name-wise equal according to definition 5. The same holds for properties and instances of classes and properties. This is the reason that in step (3) of definition 7 does not create a name clash in using the same name of classes/ properties that exists in both constituent classes/properties. A similar definition can be given for the difference of two ontologies that we omit here. Now we give an example of ontological unions and intersections.

Example 3 (Ontology union/difference). *Consider two ontologies $\mathcal{O}^a, \mathcal{O}^b \in U$ with some instances $\mathcal{I}^a$ and $\mathcal{I}^b$. For the purposes of this example, we don't distinguish between an element x and its name CN(x) or IN(x). Let $\mathcal{O}^a$ be given by following classes and properties:*

$C_0^a = \{$*university, school, employeeID* $\}$,
$P_0^a = \{$*academics, employed* $\}$.

Here academics is of type (university, school) and employed::(school, employeeID). Let $\mathcal{O}^b$ be given by following schema elements:

$C_0^b = \{$*organization, department, eid, site*$\}$,
$P_0^b = \{$*subdivision, payroll, location*$\}$ *and*
$P_1^b = \{$*reportsAt*$\}$.

Here 'subdivision' relates 'organization' to 'department', 'payroll' relates 'department' to 'eid', 'location' relates 'department' to 'site' while 'reportsAt' relates 'eid' to 'location'. The union $\mathcal{O}^{a \cup b} = \mathcal{O}^a \cup \mathcal{O}^b$ is given by following elements:

- $C_0^{a \cup b} = C_0^a \cup C_0^b = \{$*university, school, employeeID, organization, department, eid, site*$\}$
- $P_0^{a \cup b} = P_0^a \cup P_0^b = \{$*academics, employed, subdivision, payroll, location*$\}$
- $P_1^{a \cup b} = P_1^a \cup P_1^b = \{$*reportsAt*$\}$
- $\mathcal{I}^{a \cup b} = \mathcal{I}^a \cup \mathcal{I}^b$

Next we show the intersection operation by calculating $\mathcal{O}^{a \cup b} \cap \mathcal{O}^a$:

- $C_0^{(a \cup b) \cap a} = C_0^{a \cup b} \cap C_0^a = \{$*organization, department, eid, site*$\}$
- $P_0^{(a \cup b) \cap a} = P_0^{a \cup b} \cap P_0^a = \{$ *subdivision, payroll, location*$\}$
- $P_1^{(a \cup b) \cap a} = P_1^{a \cup b} \cap P_1^a = \{$*reportsAt*$\}$
- $\mathcal{I}^{(a \cup b) \cap a} = \mathcal{I}^{a \cup b} \cap \mathcal{I}^a$ □

Next we define the notion of substructure among ontologies, that we refer to as a *sub-ontology* as follows.

Definition 8 (Sub-ontologies).

Substructures: *An ontology $A = (\mathcal{C}^A, \mathcal{P}^A, \mathcal{CN}^A, \mathcal{I}^A, \mathcal{IN}^A)$ is said to be a schema subontology of an ontology $B = (\mathcal{C}^B, \mathcal{P}^B, \mathcal{CN}^B, \mathcal{I}^B, \mathcal{IN}^B)$, denoted by $A \sqsubseteq_S B$ iff $\mathcal{C}^A \subseteq \mathcal{C}^B$, $\mathcal{P}^A \subseteq \mathcal{P}^B$, $\mathcal{CN}^A = \mathcal{CN}^B \restriction \mathcal{C}^A$.*

Intensional substructures: *In addition, if $x \in c :: \mathcal{C}^A$ implies $x \in c :: \mathcal{C}^B$ and $(x, y) \in p :: \mathcal{P}^A$ implies $(x, y) \in p :: \mathcal{P}^B$, we say that B is an intensional sub ontology of A, and is denoted $A \sqsubseteq_I B$.*

Clearly, the relations $\sqsubseteq_S$ and $\sqsubseteq_I$ are partial orders. Consequently, any sub ontology of an ontology is itself an ontology, implying that all its paths originating in the sub ontology only reach entities within the sub ontology.

Example 4 (Subontologies). *Consider again the ontologies in example 3. We compare $\mathcal{O}^{a \cup b}$ with $\mathcal{O}^a$ to show that $\mathcal{O}^a \sqsubseteq_S \mathcal{O}^{a \cup b}$. We first note that $\mathcal{C}^a \subseteq \mathcal{C}^{a \cup b}$, i.e., $C_0^a \subset C_0^{a \cup b}$ and $C_1^a \subset C_1^{a \cup b}$ (and since there are no P_1^a properties, therefore, the relation holds for all subsequent types). Similarly, on inspection $\mathcal{P}^a \subseteq \mathcal{P}^{a \cup b}$ and $\mathcal{CN}^a = \mathcal{CN}^{a \cup b} \restriction \mathcal{C}^a$* □

We now use the sub ontologies to define *quotient* ontologies that are obtained by abstracting an entire sub ontology of a a super ontology to a single entity. This concept is the same as the contraction of a graph [11], and has the same meaning in the graph semantics of RDF.

Definition 9 (Quotient ontology). *Suppose $B = (\mathcal{C}^B, \mathcal{P}^B, \mathcal{CN}^B, \mathcal{I}^B, \mathcal{IN}^B)$ is a schema sub ontology of $A = (\mathcal{C}^A, \mathcal{P}^A, \mathcal{CN}^A, \mathcal{I}^A, \mathcal{IN}^A)$. Then we define the quotient ontology A/B, say (C, P, CN, I, IN) as follows.*

$$C = \mathcal{C}^A \setminus \mathcal{C}^B \cup \{c^*\} \text{ where } c^* \notin \mathcal{C}^A$$

$$P = \begin{cases} \mathcal{P}^A \setminus \mathcal{P}^B \cup \{p^* :: c_1 \times c^* \mid p :: c_1 \times c_2 \text{ and } c_2 \in \mathcal{C}^B\} \\ \cup \{p^* :: c^* \times c_2 \mid p :: c_1 \times c_2 \text{ and } c_1 \in \mathcal{C}^B\} \end{cases}$$

$$CN(x) = \begin{cases} \mathcal{CN}^A(x) & \text{if } x \in \mathcal{C}^A \setminus \mathcal{C}^B, \\ n \text{ if } x{=}c^* & \text{where } n \notin \mathcal{CN}(\mathcal{C}^B) \text{ is a new name} \end{cases}$$

The intuition captured in definition 9 is that the entire sub ontology B is shrunk to a single entity in the quotient ontology, and therefore the properties that relates any entity in $\mathcal{C}^A \setminus \mathcal{C}^B$ relates to this single (shrunk) entity in the quotient ontology A/B. This construction directly mirrors the contraction operation in graphs, and captures the intent of quotients they are defined in algebra or topology.

We now follow this trend and define *products* of ontologies, where the intent invert the contraction induced by taking a quotient, and therefore *expand* the original ontology. Towards that end taking a quotient resulted in two things: first, all entities in the sub ontology were identified as one *new* entity, and all properties that involved those entities were now associated with this *new* entity. Thus to reverse the effect of a quotient, we select a point, that we call a pivot point that we replace with another ontology and re-direct all properties associated with the previous pivot point to all other entities of the second ontology. Our formal definition follows.

Definition 10 (Pivoted product of ontologies). *Suppose $A = (C^A, P^A, CN^A, I^A, IN^A)$ and $B = (C^B, P^B, CN^B, I^B, IN^B)$ are schema ontologies and $c^* \in C^A$ a class in A. Then we define the product of A and B pivoted around c^*, say*

(C, P, CN, I, IN), *denoted by* $A\lfloor c^* \rfloor B$ *as follows.*

$$C = C^A \setminus \{c^*\} \cup C^B$$

$$P = \begin{cases} P^A \setminus \{p \in P^A \mid \textit{domain}(p) = c^* \vee \\ \quad \textit{range}(p) = c^*\} \cup \\ \bigcup\{p_{c'} \mid \exists p \in P^A, \textit{domain}(p) = c^*, \\ \quad \textit{domain}(p_{c'}) = c' \text{ and } c' \in C^B\} \cup \\ \bigcup\{p_{c'} \mid \exists p \in P^A, \textit{range}(p) = c^*, \\ \quad \textit{range}(p_{c'}) = c' \text{ and } c' \in C^B\} \end{cases}$$

$$CN(x) = \begin{cases} CN^B(x) \text{ if } x \in P^B \\ CN^A(x) \text{ if } x \in P^A, \textit{dom}(x) \neq c^*, \\ \quad \textit{range}(x) \neq c^* \\ n_{c'} \text{ where } n_{c'} \text{ is a new name for } c' \in C^B \end{cases}$$

Thus, the pivoted cartesian product of two ontologies collects the entities sans the pivot point of its two constituents as the entities of the pivot product. Then it replaces every property that has the pivot point with a set of *new* relations that relates the entity that is not the pivot point to every entity in the second ontology. The objective here is to ensure that the pivoted product produces the maximal possible properties between the entities that are not the pivot and the elements replacing the pivot, stated and proved in the next lemma.

Lemma 6.1 (Quotient-Product theorem). *Suppose* $B = (C^B, P^B, CN^B, I^B, IN^B)$ *is a schema sub ontology of the ontology* $A = (C^A, P^A, CN^A, I^A, IN^A)$, $A/B = (C, P, CN, I, IN)$ *is their quotient and* $c \in C$ *is some entity in C,* $c \notin C^A$. *Then the following holds.*

$$A \sqsubseteq_S (A/B)\lfloor c \rfloor B$$

Proof Sketch: Let O = $A/B\lfloor c \rfloor B$. We give the proof by showing that A $\sqsubseteq_S$ O, *i.e.*, $C^A \subseteq C^O$, $P^A \subseteq P^O$, $CN^A = CN^O \upharpoonright C^A$.

By definition, $C^{X/Y} = C^X \backslash C^Y \cup \{c^*\}$ $(c^* \notin C^X)$ and $C^{X\lfloor c \rfloor Y} = C^X \backslash c \cup C^Y$. Therefore $C^O = C^A \backslash C^B \cup \{c^*\} \backslash \{c\} \cup C^B = C^A \cup \{c^*\} \backslash \{c\}$. Now, since $c \notin C^A$, therefore $C^A \subseteq C^O$. Similarly, computing P^O, CN^O we can show in a straightforward manner that $P^A \subseteq P^O$, $CN^A = CN^O \upharpoonright C^A$. ■

Example 5 (Quotient/Product). *Quotient and pivot product operations can contract and expand RDF graphs, therefore, ideal for scenarios where domain information from several sources need to combined into a single ontology [13, 35]. For instance, consider the case where information about different types of vehicles, like, cars (makers: Ford, GM); trucks (makers: Ford, Toyota) and buses (maker: Mitsubishi) need to be combined or distributed. Using a modular approach, a design decision is made to incorporate individual ontologies into a* `top` *ontology. In the most simplistic case, we define a* `core` *vehicle ontology as:*
C_0 = {*Auto, Car, Truck, Bus* } *and*
P_0 = {*autoType*}
The property 'autoType' relates 'Auto' to 'Car', 'Truck' and 'Bus'. Next assume each `domain` *ontology expresses the domain knowledge as a separate ontology. Now, using pivot product on the elements 'Car', 'Truck' and 'Bus' the domain ontologies can be combined with the core ontology. For example, 'Car' ontology can capture details on gasoline based internal combustion engines, say a multi-point injection or number of cylinders for different makers or their models, whereas, 'Bus' ontology captures details on compression based internal combustion engines, while a truck ontology can accommodate both. As an illustration, consider following types with members and the resulting schema with usual meanings.*

C C_0^C = {*Cname, gasEngine, cylinders, injctMech*}; P_0^C = {*eType, hasCyl, hasInjctn* }

B C_0^B = {*Bname, cmpEngine, compRatio, displacement*}; P_0^B = {*detail, hasRatio, hasdispt* }

T C_0^T = {*Tname, Engine, horsePower*}; P_0^T = {*EngDetail, power*}

A product of core and domain ontologies can be computed, and would involve 'car', 'truck' and 'bus' replaced by C, T and B respectively. Similarly, in a reversal of steps, taking a quotient of the combined ontology with respect any domain ontology would give the combined ontology sans the particular domain ontology. Quotients can also be computed with respect to any other substructure of the combined ontology.

We now use morphisms to define merge operation that constructs a new ontology from the ontologies that contain the image and co-image of a homomorphism. As usual, we differentiate between schema merging and merging of the the complete ontology. Merge operations have two other flavors: the intensional merge, *i.e.*, equating classes, instances and properties related through a homomorphism, and the extensional merge, *i.e.*, equating ontological elements based on their structures. Both types of merge involve fusing the equated elements, while maintaining the graph structures in both the component ontologies. We first define a *fused* class, property and instance before defining the operations.

Definition 11 (Fused Elements). *Suppose* $A = (C^A, P^A, CN^A, I^A, IN^A)$ *and* $B = (C^B, P^B, CN^B, I^B, IN^B)$ *are schema ontologies and* (f_C, f_P, g_C, g_P) *be a morphism from A to B. Then for every pair of classes, properties or instances related through the homomorphism we say that they are intensionally equal up to the homomorphism, i.e., c::*$C^A =_{mrph} f_C(c)$, *p::*$P^A =_{mrph} f_P(p)$, *etc. (In the following, we omit subscript of = symbol to reduce clutter). Each component of the pair can be mapped to a fused set as follows.*

Fused Class $\digamma_C$**:** *If c::*C^A = $f_C(c)$ = *d::*C^B, *then* $\digamma_C(c)$ = $\digamma_C(d)$ = {*c,d*} *and CN(*$\digamma_C(c)$*) = CN(*$\digamma_C(d)$*) =* {CN^A*(c)*,CN^B*(d)*}

Fused Property $\digamma_P$: *If p::P^A = $f_C(p)$ = q::P^B, then* $\digamma_P(c)$ = $\digamma_P(d)$ = {*p,q*} *and CN(*$\digamma_P(p)$*) = CN(*$\digamma_P(q)$*) =* {*CN*^{A}*(p),CN*^{B}*(q)*}

Fused Instance $\digamma_I$: *If* $x \in c :: C^A$ *(resp* $p :: P^A$*) =* $g_C(c)$ *(resp* $g_P(p)$*)=* $y \in d :: C^B$ *(resp* $q :: P^B$*), then* $\digamma_I(x) = \digamma_I(y)$ = {*x,y*} *and CN(*$\digamma_I(x)$*) = CN(*$\digamma_I(y)$*)* ={*CN*^{A}*(x),CN*^{B}*(y)*}

Definition 12 (Intensional merge).

Schema Merge *Suppose* $A = (C^A, P^A, CN^A, I^A, IN^A)$ *and* $B = (C^B, P^B, CN^B, I^B, IN^B)$ *are schema ontologies and* (f_C, f_P) *be a schema morphism from A to B. Then we define the intensional schema merge of A and B, say (C, P, CN, I, IN), denoted A* $\oplus$ *B, as follows:*

$$
\begin{aligned}
C_1 &= \{c :: C^A | \exists d :: C^B, d = f_C(c)\} \\
C &= (C^A \cup C^B \setminus C_1)) \cup \digamma_C(C_1) \\
P_1 &= \{p | p = c_1 \times c_2, (c_1 \vee c_2 \in dom(f_C) \cup ran(f_C))\} \\
P_2 &= \{p| \begin{cases} p = c_1 \times c_2^* \wedge \exists p = c_1 \times c_2 \wedge c_2 \in \\ \qquad dom(f_C) \cup ran(f_C) \text{ and } c_2^* = \digamma_C(c_2) \\ p = c_1^* \times c_2 \wedge \exists p = c_1 \times c_2 \wedge c_1 \in \\ \qquad dom(f_C) \cup ran(f_C) \text{ and } c_1^* = \digamma_C(c_1) \\ p = c_1^* \times c_2^* \wedge \exists p = c_1 \times c_2 \wedge c_1, c_2 \in \\ \qquad dom(f_C) \cup ran(f_C) \text{ and } c_1^* = \digamma_C(c_1) \\ \qquad c_2^* = \digamma_C(c_2) \end{cases} \\
P_3 &= \{p | p \in (dom(f_P) \cup ran(f_P))\} \\
P_4 &= \{\digamma_P(p) | p \in (dom(f_P) \cup ran(f_P))\} \\
P &= (P^A \cup P^B \cup P_2 \cup P_4) \setminus (P_1 \cup P_3) \\
CN(x) &= \begin{cases} CN^A(x) \text{ if } x \in C^A \wedge x \notin dom(f_C) \\ CN^B(x) \text{ if } x \in C^B \wedge x \notin ran(f_C) \\ CN^A(x) \text{ if } x \in P^A \wedge x \notin dom(f_P) \\ CN^B(x) \text{ if } x \in P^B \wedge x \notin ran(f_P) \\ CN(\digamma_C(x)) \text{ if } x \in (dom(f_C) \cup ran(f_C)) \\ CN(\digamma_P(x)) \text{ if } x \in (dom(f_P) \cup ran(f_P)) \end{cases} \\
I &= \begin{cases} x \in c :: C^A \text{ if } x \in c :: C^A \wedge x \notin dom(f_C) \\ x \in c :: C^B \text{ if } x \in c :: C^B \wedge x \notin ran(f_C) \\ x \in \digamma(c) \text{ if } x \in c :: C^A \wedge x \in dom(f_C) \\ x \in \digamma(c) \text{ if } x \in c :: C^B \wedge x \in ran(f_C) \end{cases} \\
IN &= IN
\end{aligned}
$$

Instance Merge *In this case, the morphism is an instance homomorphism, therefore, I*^{A}*, I*^{B}*, and IN*^{A}*, IN*^{B} *are fused in a similar manner using* g_C *and* g_P *instead. The instance merge is represented as A* $\otimes$ *B.*

Definition 12 captures the cases where homomorphisms or mappings are used to identify 'similar' elements across ontologies and then fused together to yield a single ontology. Essentially, this involves replacing a class in either ontology by its fused class, a property by its fused property and an instance by its fused instance. Also, all the properties that were earlier connecting classes that are now fused have to be reconnected to the fused class in the merged ontology. The above definition achieves this by identifying those properties and replacing them.

Next we define the extensional flavor of merge operation. Here, elements of an ontology, *i.e.*, classes, and properties are *equated* based on their structure. In other words, classes and properties are equal *up to their extensions*. Clearly, this means that the two ontologies either share the same instances or their instances have already been fused using some other means. We define extensionally fused elements using (Γ_C, Γ_P) mappings. In other words, if c::C^A = d::C^B, *i.e.*, c and d are structurally equal, then they are fused together. Similarly, structural equality can be used to define fused elements for properties. As expected, $\Gamma_C(c) = \Gamma_C(d)$ = {c, d} with CN($\Gamma_C(c)$) = CN($\Gamma_C(d)$) = {CN^A(c), CN^B(d)}. Γ_P achieves fusion for properties.

Definition 13 (Extensional merge). *Suppose* $A = (C^A, P^A, CN^A, I, IN)$ *and* $B = (C^B, P^B, CN^B, I, IN)$ *are ontologies. We define the extensional merge of A and B, say (C, P, CN, I, IN), denoted A* $\uplus$ *B, as follows:*

$$
\begin{aligned}
C_1 &= \{c| \begin{cases} c :: C^A, \exists d :: C^B, c = d\} \\ c :: C^B, \exists d :: C^A, c = d\} \end{cases} \\
C &= (C^A \cup C^B \cup \Gamma_C(C_1)) \setminus C_1 \\
P_1 &= \{p | p = c_1 \times c_2, (c_1 \in C_1 \vee c_2 \in C_1)\} \\
P_2 &= \{p| \begin{cases} p = c_1 \times c_2^* \wedge \exists p = c_1 \times c_2 \wedge c_2 \in C_1 \\ \qquad \text{and } c_2^* = \Gamma_C(c_2) \\ p = c_1^* \times c_2 \wedge \exists p = c_1 \times c_2 \wedge c_1 \in C_1 \\ \qquad \text{and } c_1^* = \Gamma_C(c_1) \\ p = c_1^* \times c_2^* \wedge \exists p = c_1 \times c_2 \wedge c_1, c_2 \in C_1 \\ \qquad \text{and } c_1^* = \Gamma_C(c_1), c_2^* = \Gamma_C(c_2) \end{cases} \\
P_3 &= \{p| \begin{cases} p :: P^A, \exists p' :: P^B \text{ s.t. } p = p'\} \\ p :: P^B, \exists p' :: P^A \text{ s.t. } p = p'\} \end{cases} \\
P_4 &= \{\Gamma_P(p) | p \in P_3\} \\
P &= (P^A \cup P^B \cup P_2 \cup P_4) \setminus (P_1 \cup P_3) \\
CN(x) &= \begin{cases} CN^A(x) \text{ if } x \in C^A \wedge x \notin C_1 \\ CN^B(x) \text{ if } x \in C^B \wedge x \notin C_1 \\ CN^A(x) \text{ if } x \in P^A \wedge x \notin P_3 \\ CN^B(x) \text{ if } x \in P^B \wedge x \notin P_3 \\ CN(\digamma_C(x)) \text{ if } x \in C_1 \\ CN(\digamma_P(x)) \text{ if } x \in P_3 \end{cases}
\end{aligned}
$$

In addition, if x $\in$ *c::C*^{A} *then x* $\in \Gamma_C(c)$ *or if x* $\in$ *d::C*^{B} *then x* $\in \Gamma_C(d)$*. Finally, IN remains unchanged.*

Example 6 (Schema Merge). *Consider again the ontologies* $\mathcal{O}_a$ *and* $\mathcal{O}_b$ *introduced in example 3. Following list recaps the schema elements in each ontology:*

- C_0^a = {*university, school, employeeID* }
- P_0^a = {*academics, employed* }

– $C_0^b = \{$*organization, department, eid, site*$\}$
– $P_0^b = \{$*subdivision, payroll, location*$\}$
– $P_1^b = \{$*reportsAt*$\}$

Assume next a scenario where an organization wishes to share university data on employees in different schools. The schema need to be merged according to the following understanding:

- *employeeID in $\mathcal{O}^a$ is the same as eid in $\mathcal{O}^b$*
- *employed in $\mathcal{O}^a$ is the same as payroll in $\mathcal{O}^b$*
- *school in $\mathcal{O}^a$ is the same as department in $\mathcal{O}^b$*

Using these rules a homomorphism is constructed with following mappings:

$$\begin{aligned} f_C(school) &= department \\ f_C(employeeID) &= eid \\ f_P(employed) &= payroll \end{aligned}$$

Fused classes and properties are given as

$$\begin{aligned} F_C(school) &= \{school, department\} \\ &= F_C(department) \\ F_C(employeeID) &= \{employeeID, eid\} \\ &= F_C(eid) \\ F_P(employed) &= \{employed, payroll\} \\ &= F_P(payroll) \end{aligned}$$

Based on the equivalence classes, it is easy to see that the merge $\mathcal{O}^a \oplus \mathcal{O}^b$ is given as:

– $C_0^{a\oplus b} = \{$*university*, $\{$*school, department*$\}$, $\{$*employeeID, eid*$\}$, *organization, site*$\}$
– $P_0^{a\oplus b} = \{$*academics*, $\{$*employed, payroll*$\}$, *subdivision, location*$\}$
– $P_1^{a\oplus b} = \{$*reportsAt*$\}$ □

6.1 Algebraic properties

In this section we explore simple algebraic properties of the operations introduced in the previous section. We begin with the usual laws of union, complement and intersection, *i.e.*, commutative union, commutative intersection, distributive union, distributive intersection, *etc.*

Lemma 6.2. *For all ontologies A, B and C $\in$ U, the following statements hold:*

1. *commutative union: $A \cup B = B \cup A$*
2. *associative union: $A \cup (B \cup C) = (A \cup B) \cup C$*
3. *commutative intersection: $A \cap B = B \cap A$*
4. *associative intersection: $A \cap (B \cap C) = (A \cap B) \cap C$*
5. *distributive union: $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$*
6. *distributive intersection: $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$*
7. *Absorption: $A \cap (A \cup B) = A$*
8. *Absorption: $A \cup (A \cap B) = A$*

Proof Sketch:

1. By definition, $A \cup B = \langle \mathcal{C}^{A\cup B}, \mathcal{P}^{A\cup B}, \mathcal{CN}^{A\cup B}, \mathcal{I}^{A\cup B}, \mathcal{IN}^{A\cup B} \rangle$, with $\mathcal{CN}^{A\cup B} = \mathcal{CN}^A \cup \mathcal{CN}^B$ and $\mathcal{IN}^{A\cup B} = \mathcal{IN}^A \cup \mathcal{IN}^B$. Also by definition, $\mathcal{C}^{A\cup B}$ is given (inductively) by $C_i^{A\cup B} = C_i^A \cup C_i^B$, and $\mathcal{P}^{A\cup B}$ by $P_i^{A\cup B} = P_i^A \cup P_i^B$. Similarly, $\mathcal{I}^{A\cup B} = \mathcal{I}^A \cup \mathcal{I}^B$. But, by the commutativity of set union, each of the sets obtained are the equal to those obtained by $B \cup A$.

Similarly, rest of the proofs are shown by the analogous properties of sets.

Lemma 6.3 (Merge). *For A,B and C $\in$ U, the following identities hold:*

1. *If homomorphisms are injective then following operations are commutative:*
   - *$A \oplus B = B \oplus A$*
   - *$A \otimes B = B \otimes A$*
   - *$A \uplus B = B \uplus A$*
2. *Idempotence:*
   - *$A \oplus A = A$*
   - *$A \otimes A = A$*
   - *$A \uplus A = A$*

Proof Sketch:

1. *Commutativity:* Since homomorphisms are injective, and surjective on the set of elements that is a restriction to the image of the homomorphism. More precisely

$$c \xrightarrow{f_C} d$$

then

$$d \xrightarrow{f_C^{-1}} c$$

Hence $F_C(c) = F_C(d) = \{$c,d$\}$ for both f_C and f_C^{-1}. Similarly, fused properties and instances are the same for injective homomorphisms. Secondly, domain(f_C) = range(f_C^{-1}); range(f_C) = domain(f_C^{-1}). Similar equality holds for f_P, g_C and g_P. Therefore, the set of fused classes, properties and instances is identical for both L.H.S and R.H.S. Next, by definition, and set commutativity, the set of properties of merged ontology in the L.H.S. is identical to the set on the R.H.S. Same holds for the schema and instance naming mappings. Therefore in each case, *i.e.*, $\oplus, \otimes, \uplus$, the merged ontology on the L.H.S is identical to the one on the R.H.S.

2. *Idempotent:* The homomorphism under consideration is an endomorphism. Due to absorption in sets, fused elements are identical to their counter images under F or Γ. Also, an endomorphism is an isomorphism, therefore, by arguments given above the result holds for all three cases – $\oplus, \otimes, \uplus$. ■

Lemma 6.4 (Subontology). *For ontologies A, B and C $\in$ U, following identities hold:*

- *If $A \sqsubseteq B$ then $A \oplus C \sqsubseteq B \oplus C$*
- *If $A \sqsubseteq B$ then $A \otimes C \sqsubseteq B \otimes C$*
- *If $A \sqsubseteq B$ then $A \uplus C \sqsubseteq B \uplus C$*

Proof Sketch: By definition, A $\sqsubseteq$ B implies $C^A \subseteq C^B$, $P^A \subseteq P^B$ and $CN^A C^B \restriction C^A$. Therefore for each of the three cases – $\oplus, \otimes, \uplus$, we must show that

$$C^{A\ op\ C} \subseteq C^{B\ op\ C}, P^{A\ op\ C} \subseteq P^{B\ op\ C} and$$

$$CN^{A\ op\ C} = C^{B\ op\ C} \restriction C^{A\ op\ C}$$

where *op* substitutes for either of $\oplus, \otimes$ and $\uplus$. Informally, by definition for each operator *op*, the set of classes in merged ontology A *op* C is $C^A \cup C^C \setminus (range(f_C) \cup domain(f_C)) \cup \digamma_C(A)$. By a similar token, and $C^A \subseteq C^B$, it is easily shown that $C^{A\ op\ C} \subseteq C^{B\ op\ C}$. Similarly, other parts are shown in a straightforward manner. ■

7 Comparison with existing algebras

7.1 Query Algebras

Several proposals for RDF Query algebra exist and here we compare our algebra to operators defined in earlier works. Prominent ones include RQL [17], SquishQL [21], TRIPLE [33], RDQL [31], SeRQL [4], SPARQL [30], LAGAR [8], *etc.* We give an overview of operators in some of these proposals and evaluate them with respect to algebra presented here.

LAGAR supersedes most of the earlier proposals, so next, we make detailed comparisons with this work. From a similarities standpoint, both algebras consider only closed operators that can be composed (we give additional proofs of this statement). A basic difference between the two algebras is in modeling of RDF. LAGAR employs 'flat' graphs, whereas as our graphs are layered to accommodate reified schemas. So, we support a larger class of RDF graphs. Mappings between two LAGAR graphs are sub-group isomorphisms, *i.e.*, they preserve the structure of mapped triples. Our mappings are relaxed to be non-injective, but have to be properly typed. Thus, they can themselves be captured as ontologies in the same universe of ontologies. Next major difference is the absence of a predefined universe of elements, therefore, interpretations of admitted set-based operations are not well founded. Specific comparisons follow.

(σ, π) Since LAGAR is a query algebra, it includes the usual *selection* and *projection* operators, which we don't currently provide. These operators are based on *schema-assisted* pattern graph matching on the knowledge base and restricting the resultant graphs to a set of output nodes. Since we interpret primitive `rdf` properties, like, `rdf:type`, `rdfs:subClassOf`, we can compute semantic-aware paths.

$(\times, join)$ For *product* two graphs are connected at their *roots* through a common *super node*. In comparison, our product is graph expansion operation, which can be applied to any node of a schema. LAGAR uses intensional equality for *join* operations, whereas, we support multiple types of equality – name, structural and location equality.

$(union, intersection, difference)$ Union, intersection and difference are defined as component-wise set union of graphs with, presumably, intensional equality. However, the semantics of these operations can possibly be non-well founded. In comparison, our intensional union and intersection operators have well-founded semantics.

$(merge)$ Merge operation is defined as a union after removal of 'identical' blank nodes from one of the graphs. This amounts to deleting some properties in one of the graphs before a (intensional) set union is performed. In contrast, our merge operation relates two ontologies through a homomorphism, and structurally aligns or fuses them.

$(Construction\ and\ functional\ operators)$ Inserting nodes, edges, deleting them and changing their values are termed as construction operators in LAGAR. We model these through quotient and product operations that add sets of nodes and edges while preserving or recovering original structure. Functional operators are not modeled.

Other related query algebra exist and we revisit only specific portions that are different from LAGAR. For instance, algebra proposed in SquishQL [21] is a subset of RQL's query algebra. Similarly, SeRQL [4] provides *select* and *construct* operators, which, essentially, provide similar functionality as SquishQL. However, construct can reverse the structure of some triples, and cause structural mismatches between the input and output graphs, therefore, not modeled here. SeRQL supports reification – same as us; and some schema-awareness in matching path patterns, which we can support.

Finally, RQL's query algebra provides a type system for RDF, just like we do here. However, their type system supports a very limited form of *reification*, which they call *property refinement*, whereas through our recursive countable types we support it completely. Their algebra consists of RDF counterparts of relational operators like *select*, *join*, *etc.*, most of which are subsumed in LAGAR.

7.2 Ontology Integration

In this section we compare the algebra we present to existing ontology-combination implementations like Chimaera [20], Ontomorph [7], onion [22], Prompt [26], *etc.*

Each of the discussed techniques are semi-automatic approaches towards ontology merging or alignment and are focussed at constructing software tools that can be used to *combine* independently developed ontologies. In contrast, the approach taken here is to develop a systematic mathematical theory for ontology combination. Hence, we ignore the engineering aspects and focus on the mathematical aspects only.

Smart [25] describes combination operations, without formally defining them, as follows. *Merge* is stated as an operation that takes as input two schemas and computes a single schema with similar classes and properties fused together. *Alignment*, on the other hand, combines the two input ontologies without fusing them together, *i.e.*, the two ontologies maintain their separate identities after alignment, with the addition of *bridges* or *links* built across the two, identifying similar elements. Finally, they also describe a *translation* between ontologies, *i.e.*, a schema isomorphism. But, the authors discount the feasibility of isomorphisms between ontologies. Each of these operations are formally defined and modeled by us. Smart (and its descendant Prompt [26]) discovers 'equal' or similar items algorithmically, through name matching, *i.e.*, they only use intensional equality theory. Operations on ontologies are restricted to primitive element copies, like merge, shallow-copy, deep-copy, *etc.* In contrast, Chimaera proposes both intensional and extensional equality criteria for discovering 'equal' elements, which we formally define here. Again, they ignore structural details and formalisms and focus mainly on strengthening their matching algorithm.

The ONION project [22, 23] provides another algebra for ontology composition. However, it is limited in comparison to our algebra. These limitations include the absence of a predefined universe that prevents well founded interpretations of objects. ONION skirts this issue by restricting to the syntax alone, without formally going into the semantics of their proposed syntax. Also, ONION does not accommodate reified schemas. A fundamental component of ONION is the *articulation ontology* that represents (subsumption) relationships between the source ontologies. In essence, articulation ontology is similar to our homomorphic ontologies, though:

- ONION allows subsumption relationships between classes in the source ontologies, as well as their collections. We do not model subsumption between classes in different ontologies.
- ONION allows *Horn Clause* rules for expressing relationships between ontologies, without interpreting them. We restrict to mappings only.
- We distinguish between schema and instance level homomorphism and, consequently, between schema, instance and name equality of source ontologies. ONION does not address these issues.

ONION's graphs can be transformed using primitive node addition, node deletion, edge addition, and edge deletion operations on graphs. In our approach, we define more standard product and quotient operators to fuse/extricate graphs to/from existing graphs. Next we analyze their algebraic operations on graphs:

- Unary operators *filter* and *extract* that are similar to relational algebra select, project operations. Currently we do not provide these operators.
- Binary operators *intersect, union, difference* use articulation rules AR to construct new ontologies. More specifically,
  - The intersection operation generates an articulation ontology graph that contains nodes that appear in articulation rules; edges in the source ontologies that connect nodes appear in the articulation rules; and the new edges generated by the articulation rules. Union combines the source ontologies while using an articulation to 'equate' nodes. Difference includes nodes and edges of the first ontology that are not in the second one.
  - We define these operations as type-based intersection of classes, properties, and instances, where elements are equal (for finding duplicates in a set) if they are name-wise equal. We also support flavors of merge operation.

In our work, we can capture most of the expressed operations and requirements algebraically. For instance, we provide intensional and extensional schema (and instance) merge operations for capturing merge operations discussed above. Similarly, alignments, or articulations are captured in homomorphic ontologies. In fact, homomorphic ontologies capture much more information that the earlier proposals miss. In addition, we describe the standard intensional and extensional set union and intersection operations, homomorphisms, substructures, product and quotients. Thus, we supply a fairly well-developed algebra for ontologies and show how these operations are useful in real world applications.

8 Conclusion

In this paper we develop an algebra for composing ontologies. We review related work in detail in section 7, but most of earlier proposals are focussed on the practical aspects of ontology combination, with limited theoretical or conceptual understanding of issues involved. As a result, they fail to precisely state the details of the combination operations they describe, even though considerable work has undergone to formalize the underlying objects they work with. Consequently, it is difficult for a reader to realize the subtle differences between the

various combinatorial operations described in the literature. Hence, there is a need to develop a theoretical basis to precisely define combinatorial operations, allowing consumers of knowledge representation to realize benefits and drawbacks of each technique. Thus, with a set of basic operations, like, intensional union/intersection, quotients, intensional/extensional merge, pivot products, *etc.*, readers can reduce any operation in the literature to the combination of these basic operations.

In addition to filing this gap in knowledge representation, we also formally define countably reified RDF statements. We achieve this task through a recursive countable type representation for reified statements that assigns them layered semantics more general than the stratified semantics presented by Pan and Horrocks [28]. Thus we enhance the state of the art in formalizing RDF. It is our hope that these contributions will help remove ambiguities that currently exist in the field.

References

[1] P. Aczel. *Non-well-founded sets*, volume 14 of *Lecture Notes*. CSLI, 1988. (See page 11).

[2] Y. Breitbart, P. L. Olson, and G. R. Thompson. Database integration in a distributed heterogeneous database system. In *Proceedings of the Second International Conference on Data Engineering, February 5-7, 1986, Los Angeles, California, USA*, pages 301–310. IEEE Computer Society, 1986.

[3] D. Brickley and R. Guha. Resource Description Framework (RDF) Schema Specification 1.0: RDF schema. W3C workding Draft, 2003.

[4] J. Broekstra and A. Kampman. Serql, a second generation RDF query language. In *SWAD-Europe Workshop on Semantic Web Storage and Retrieval*, Amsterdam, Nov 2004.

[5] T. Burners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, May 2001.

[6] H. Chalupsky. Ontomorph: A translation system for symbolic knowledge. http://www.isi.edu/ hans/ontomorph/presentation/ ontomorph.html.

[7] H. Chalupsky. Ontomorph: A translation system of symbolic logic. In *KR2000: Principles of Knowledge representation and reasoning*, pages 471–482, 2000.

[8] L. Chen, A. Gupta, and M. E. Kurul. A semantic-aware RDF query algebra. In *12th International Conference on Management of Data (COMAD)*, Hyderabad, Dec 2005.

[9] P. P. Chen. The Entity-Relationship Model - Toward a Unified View of Data. *ACM Transactions on Database Systems (TODS)*, 1(1):9–36, 1976.

[10] U. Dayal and H.-Y. Hwang. View definition and generalization for database integration in a multidatabase system. *IEEE Transactions on Software Engineering*, SE(10):628–645, Nov 1984.

[11] R. Diestel. *Graph Theory*. Springer-Varlag, 1997. Graduate texts in Mathamatics 173.

[12] B. C. Grau. A possible simplification of the semantic web architecture. In *WWW 2004*, pages 17–22. ACM, May 2004.

[13] B. C. Grau, I. Horrocks, O. Kutz, and U. Sattler. Will my ontologies fit together? In *2006 International Workshop on Description Logics - DL2006*, 2006.

[14] B. C. Grau, B. Parsia, and E. Sirin. Working with multiple ontologies on the semantic web. In *Proceedings of the Third Internatonal Semantic Web Conference (ISWC2004). Volume 3298 Lecture Notes in Computer Science.*, 2004.

[15] T. Gruber. What is an ontology. available at http://www-ksl.stanford.edu/kst/what-is-an-ontology.html.

[16] P. Hayes. RDF semantics. W3C workding Draft, February 2003.

[17] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl. RQL: a declarative query language for RDF. In *11th international world wide web conference (WWW02)*, pages 592–603, May 2002.

[18] M. Klein. Combining and relating ontologies: an analysis of problems and solutions. In A. Gomez-Perez, M. Gruninger, H. Stuckenschmidt, and M. Uschold, editors, *Workshop on Ontologies and Information Sharing, IJCAI'01*, Seattle, USA, Aug. 4–5, 2001.

[19] K. J. Kunen. *Set Theory*. North Holland, Reprint edition, December 1983.

[20] D. L. McGuinness, R. Fikes, J. P. Rice, and S. Wilder. An environment for merging and testing large ontologies. In *KR2000: Principles of Knowledge representation and reasoning*, pages 483–493, 2000.

[21] L. Miller, A. Seaborne, and A. Reggiori. Three implementations of SquishQL, a simple RDF query language. In *International Semantic Web Conference (ISWC)*, pages 399–403, 2002.

[22] P. Mitra and G. Wiederhold. *An Ontology-Composition Algebra*, pages 93–117. International Handbooks on Information Systems. Springer-Verlag, handbook on ontologies edition, 2004.

[23] P. Mitra, G. Wiederhold, and M. Kersten. A graph oriented model for articulation of ontology interdependencies. In *Conference on extending database technology (EDBT 2000)*, March 2000.

[24] W. Y. Mok. A comparative study of various nested normal forms. *Knowledge and Data Engineering*, 14(2):369–385, 2002.

[25] N. Noy and M. Musen. Smart: Automated support for ontology merging and alignment. In *Twelth Workshop on Knowledge Acquisition, Modeling, and Management*, Banff, Canada, 1999.

[26] N. F. Noy, M. A. Musen, and E. Shortliffe. PROMPT: algorithm and tool for automated ontology merging and alignment. In *17th National conference on artificial intelligence (AAAI-2000)*, 2000.

[27] J. Pan and I. Horrocks. RDFS(FA) and RDF MT: Two semantics for RDFS. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *Proc. of the 2003 International Semantic Web Conference (ISWC 2003)*, number 2870 in Lecture Notes in Computer Science, pages 30–46. Springer, 2003.

[28] J. Pan and I. Horrocks. RDFS(FA) and RDF MT: Two semantics for RDFS. In *2003 International Semantic Web Conference (ISWC 2003)*, pages 30–46, 2003.

[29] C. Parent and S. Spaccapietra. Issues and approaches

of database integration. *Communications of the ACM*, 41(5es):166–178, May 1998.

[30] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. http:/www.w3.org/TR/rdf-sparql-query, Apr, 2005.

[31] A. Seaborne. A query language for RDF. http:/www.w3.org/Submission/2004/SUBM-RDQL-20040109, 2004.

[32] A. Sheth and J. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[33] M. Sintek and S. Decker. Triple, an RDF query, inference and transformation language. In *Deductive databases and knowledge management (DDLP)*, 2001.

[34] G. Stumme and A. Maedche. FCA-MERGE: Bottom-up merging of ontologies. In *Workshop on Ontologies and Information Sharing, IJCAI'01*, pages 225–234, 2001.

[35] R. Volz, D. Oberle, and A. Maedche. Towards a Modularized Semantic Web. In *Workshop on Ontologies and Semantic Interoperability*, July 2002.