A Graph Theoretical Foundation for Integrating RDF Ontologies[*]

Octavian Udrea[1] **Yu Deng**[1] **Edna Ruckhaus**[2,3] **V.S. Subrahmanian**[1]

[1] University of Maryland, College Park MD 20742, USA, {udrea, yuzi, vs}@cs.umd.edu

[2] Universidad Simón Bolívar, Caracas, Venezuela, ruckhaus@ldc.usb.ve

[3] Maryland Information and Network Dynamics Laboratory, College Park MD 20742, USA

Abstract

RDF ontologies are rapidly increasing in number. We study the problem of integrating two RDF ontologies under a given set H of Horn clauses that specify semantic relationships between terms in the ontology, as well as under a given set of negative constraints. We formally define the notion of a "witness" to the integrability of two RDF ontologies under such constraints. A witness represents a way of integrating the ontologies together. We define a "minimal" witnesses and provide the polynomial CROW (Computing RDF Ontology Witness) algorithm to find a witness. We report on the performance of CROW both on DAML, SchemaWeb and OntoBroker ontologies as well as on synthetically generated data. The experiments show that CROW works very well on real-life ontologies and scales to massive ontologies.

Introduction

Since the adoption of "Resource Description Framework" (RDF) as a web recommendation by the World Wide Web Consortium, there has been growing interest in using RDF for expressing ontologies about a diverse variety of topics. As more and more ontologies emerge about the same topics, there is a growing need to integrate these ontologies.

There are some initial approaches to merging ontologies in the literature. The initial pioneering work of (Mitra, Wiederhold, & Jannink 1999) showed that ontology merging is an important problem. In another important paper, (Calvanese, Giacomo, & Lenzerini 2001) develop a model theoretic basis for merging ontologies assuming they are in description logic. (Bouquet *et al.* 2003) extends the syntax of OWL by adding constructs to express relationships between multiple OWL ontologies, but don't actually tell us how to merge ontologies together using these relationships. (McGuinness *et al.* 2000) describe a tool called Chimaera that finds taxonomic "areas" for merging as well a a list of similar terms. (Stumme & Maedche 2001) use natural language and formal concept analysis methods to merge ontologies using a concept lattice which is explored and transformed by user interactions.

Our work is different from the above papers in the following respects: (i) First, we focus on RDF ontologies, (ii) Unlike the work of (Bouquet *et al.* 2003; McGuinness *et al.* 2000), we do not focus on finding relationships between terms – but we can use their work as an input to ours, (iii) our algorithms need human intervention only in specifying term relationships [1] - once these are known, we can merge ontologies without human input, (iv) our framework allows relationships to be not only between terms, but also allows complex Horn Constraints, (v) Our approach is rooted in the novel concept of an integration witness and in graph theoretic methods and includes correctness and complexity proofs, (vi) we have a prototype implementation that was tested both on large automatically generated synthetic ontologies as well as on real ontologies listed at the DAML, SchemaWeb and OntoBroker sites — the implementation shows that our algorithms scale well.

Preliminaries

In this section, we provide a *very brief* overview of the most important construct in RDF and show how RDF documents may be viewed as graphs.

An *RDF-ontology* is a finite set of triples (r, p, v) where r is a *resource* name, p is a *property* name, and v is a value (which could also be a resource name). RDF-ontologies assume the existence of some set $\mathcal{R}$ of resource names, some set $\mathcal{P}$ of property names, and a set $dom(p)$ of values associated with any property name p. We do not address reification and containers in RDF due to space constraints. *Throughout the rest of this paper, we will assume that $\mathcal{R}, \mathcal{P}, dom$ are all arbitrary, but fixed.*

Definition 1 *(RDF Ontology graph). Suppose $\mathcal{O}$ is an RDF-ontology. An* RDF ontology graph *for O is a labeled graph (V, E, λ) where*

(1) $V = \mathcal{R} \cup \bigcup_{p\in\mathcal{P}} dom(p)$ is the set of nodes.

(2) $E = \{(r, r') \mid$ there exists a property p such that $(r, p, r') \in O\}$ is the set of edges.

[*]Work supported in part by ARO grant DAAD190310202, ARL grants DAAD190320026 and DAAL0197K0135, NSF grants IIS0329851 and 0205489 and UC Berkeley contract number SA451832441 (subcontract from DARPA's REAL program).

[1]In separate research, we have given an interoperation constraint inference algorithm to infer such relationships.

```
<Property ID="affiliateOf">
  <domain resource="#Organization" />
  <range resource="#Person" />
</Property>
<Class ID="Student"> <subClassOf resource="#Person" /> </Class>
<Class ID="GraduateStudent"> <subClassOf resource="#Student" /> </Class>
<Class ID="Organization"> <subClassOf resource="#SHOEEntity" /> </Class>
<Class ID="Faculty"> <subClassOf resource="#Worker" /> </Class>
<Class ID="Person"> <subClassOf resource="#SHOEEntity" /> </Class>
<Class ID="Worker"> <subClassOf resource="#Person" /> </Class>
```

```
<owl:Class rdf:ID="Person"/>
<owl:Class rdf:ID="Organization"/>
<owl:Class rdf:ID="Working-Person"><rdfs:subClassOf rdf:resource="#Person"/></owl:Class>
<owl:Class rdf:ID="Affiliated-Person"><rdfs:subClassOf rdf:resource="#Person"/></owl:Class>
<owl:Class rdf:ID="Organization-Unit"/>
<owl:Class rdf:ID="Researcher"><rdfs:subClassOf rdf:resource="#Working-Person"/></owl:Class>
<owl:Class rdf:ID="Researcher-In-Academia">
  <rdfs:subClassOf rdf:resource="#Academic"/>
  <rdfs:subClassOf rdf:resource="#Researcher"/>
</owl:Class>
<owl:Class rdf:ID="Employee">
  <rdfs:subClassOf rdf:resource="#Working-Person"/>
  <rdfs:subClassOf rdf:resource="#Affiliated-Person"/>
</owl:Class>
<owl:Class rdf:ID="Educational-Employee"><rdfs:subClassOf rdf:resource="#Employee"/></owl:Class>
<owl:Class rdf:ID="Academic"><rdfs:subClassOf rdf:resource="#Educational-Employee"/></owl:Class>
<owl:Class rdf:ID="Student"><rdfs:subClassOf rdf:resource="#Affiliated-Person"/></owl:Class>
<owl:Class rdf:ID="PhD-Student"><rdfs:subClassOf rdf:resource="#Student"/></owl:Class>
<owl:ObjectProperty rdf:ID="has-sub-unit">
  <rdfs:range rdf:resource="#Organization-Unit"/>
  <rdfs:domain rdf:resource="#Organization"/>
</owl:ObjectProperty>
```

Figure 1: RDF (respectively OWL) for the two example ontologies

(3) $\lambda(r, r') = \{p \mid (r, p, r') \in O\}$ *is the edge labeling function.*

It is easy to see that there is a one-one correspondence between RDF-ontologies and RDF-ontology graphs. Given one of them, we can uniquely determine the other. As a consequence, we will often abuse notation and interchangeably talk about both RDF-ontologies and RDF-ontology graphs. Figure 1 shows parts of RDF ontologies 64 and 322 from the DAML web site (www.daml.org) — their graphs are shown in Figure 2. Note that both ontologies have had terms renamed (through the attachment of strings ":1" and ":2" respectively). Thus, STUDENT:2 refers to the student concept in ontology 2.

Given two nodes r, r' in an RDF-ontology graph, and a property name p, we say that there exists a *p-path* from r to r' if there is a path from node r to node r' such that every edge along the path contains p in its label. For example, in the second graph of figure 2, there is an S-path from RESEARCHER-IN-ACADEMIA to EMPLOYEE. Here S stands for "subClassOf". However, there is no A-path from STUDENT:2 to ORGANIZATION-UNIT:2 where A stands for "Affiliate-Of".

Our techniques differentiate between *transitive* and *non-transitive* properties. For instance, SUBCLASSOF is a transitive relationship, while AFFILIATEOF as depicted in Figure 2 is not. A cycle involving a transitive relationship could indicate a semantic problem (e.g. a BOSSOF b, b BOSSOF c, c BOSSOF a seems to indicate a problem). However, a cycle involving a non-transitive properties may not be a problem (e.g. a FRIENDOF b, b FRIENDOF c, c FRIENDOF a is not unusual).

Definition 2 (Graph embedding) *Let* $G_1 = (V_1, E_1, \lambda_1)$ *and* $G_2 = (V_2, E_2, \lambda_2)$ *be two RDF-ontology graphs.* G_1 *can be* embedded *into* G_2 *(denoted by* $G_1 \sqsubseteq G_2$*) iff there exists a mapping* $\omega : V_1 \to V_2$ *such that:*

(1) For p transitive, if there is a p-path from r *to* r' *in* G_1*, then there is a p-path from* $\omega(r)$ *to* $\omega(r')$ *in* G_2*.*

(2) For q non-transitive, if there is an edge labeled with q from r *to* r' *in* G_1*, then there is an edge labeled with q from* $\omega(r)$ *to* $\omega(r')$ *in* G_2*.*

Graphs G_1 *and* G_2 *are* equivalent *iff* $G_1 \sqsubseteq G_2$ *and* $G_2 \sqsubseteq G_1$*.*

Note that for any RDF ontology O, we can uniquely determine its RDF-ontology graph — however, there may be many other graphs that are equivalent to it.

Horn constraints

When integrating two RDF ontologies, there may be various kinds of constraints linking the terms in the two ontologies. For instance, we may say that the terms PERSON:1 and PERSON:2 are equivalent. Likewise, we may say that if x is a RESEARCHER-IN-ACADEMIA:2 and also a STUDENT:2, then x is also a GRADUATE-STUDENT:1. In general, for any property p, we may have a Horn clause saying that as far as property p is concerned, if an individual is in all classes in a given set, then he is also in another given class.

Definition 3 (Horn constraint) *If* $r_1, \ldots, r_n, r$ *are all resource names, then*

$$r_1 \wedge \cdots \wedge r_n \to r$$

is a Horn[2] constraint*.*

Note that Horn constraints only specify class subclass relationships, and nothing else.

Definition 4 (Negative constraints) *Suppose* a, b *are nodes and q is a property. Then:*

(1) $a \neq b$ *is a negative constraint.*

(2) $a \not\to_q b$ *is a negative constraint (which states that there is no q-path from a to b in the graph in question).*

Two examples of negative constraints associated with Figure 2 are:

- FACULTY:1$\not\to_S$ STUDENT:2 which states that Faculty is not a subclass of Student.
- STUDENT:2$\not\to_S$ FACULTY:1 which states that STUDENT is not a subclass of FACULTY.

Satisfaction of a Horn clause or a negative constraint by an ontology is straightforward to define.

[2]Strictly speaking, we should say "definite clause"(Lloyd 1987) constraint here rather than Horn constraint, but we will abuse notation and call these Horn clauses.

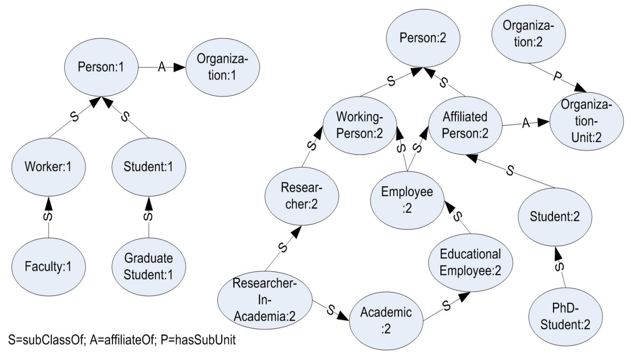


Figure 2: Two simple ontologies

The RDF Ontology Integration Problem

In this section, we will declaratively define the problem of integration of two RDF ontologies under a given set of Horn and negative constraints. To do this, we first define the notion of a "Horn Ontology Graph" (HOG for short) which merges Horn constraints with an ontology.

Definition 5 (Horn Ontology Graph (HOG)) *Suppose O is an ontology and H is a finite set of Horn constraints over $\mathcal{R}, \mathcal{P}, dom$. The* Horn Ontology Graph*, **HOG**(O, H) is defined as the labeled graph $G = (V, E, \lambda)$ where:*

(1) If a is either a resource name or a domain value in O, then $\{a\}$ is a node in V.

(2) If $r_1 \wedge \ldots \wedge r_n \rightarrow r$ is in H, then there is a node in V called " $\{r_1, \ldots, r_n\}$" — sometimes, we may abuse notation and write this node's label as "$r_1 \wedge \ldots \wedge r_n$"

(3) Whenever there are two nodes A, B in V such that $A \subseteq B$, then there is an edge labeled S from B to A.

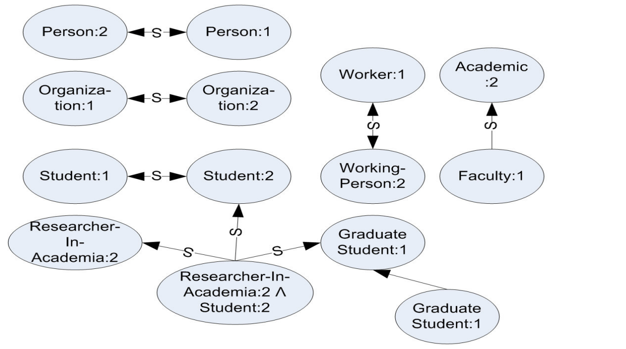
(4) If $r_1 \wedge \ldots \wedge r_n \rightarrow r$ is in H, then there is an edge labeled S in E from " $\{r_1, \ldots, r_n\}$" to $\{r\}$.

(5) If there is an edge in O's graph from a to b labeled p, then there is an edge in E from $\{a\}$ to $\{b\}$ labeled p.

Intuitively, the nodes in the Horn ontology graph are obtained by taking nodes in O's graph and making singleton sets out of them. In addition, we take the body of each Horn constraint and make it a node labeled by the entire body of the Horn constraint.

As far as edges are concerned, condition (5) says that all edges in O are present in the HOG (the only difference is that an edge in the original RDF graph from a to b ends up being an edge from $\{a\}$ to $\{b\}$). In addition, for every Horn clause in H, we add an edge from the set associated with its body to the singleton set associated with its head.

Strictly speaking, we may be able to eliminate some edges from $HOG(O, H)$. For instance, if we have one Horn clause with the body $\{a, b, c\}$, another with the body $\{a, b\}$ and yet another with the body $\{a\}$, then condition (3) above states that there should be an edge from both $\{a, b\}$ and $\{a, b, c\}$ to $\{a\}$ as well as an edge from $\{a, b, c\}$ to $\{a, b\}$. We henceforth assume all such redundant edges are eliminated. It is easy to see that the Horn Ontology Graph thus



Person:1 ↔ Person:2
Organization:1 ↔ Organization:2
Worker:1 ↔ WorkingPerson:2
Student:1 ↔ Student:2
Faculty:1 → Academic:2
PhD-Student:2 → GraduateStudent:1
Researcher-In-Academia:2 ∧ Student:2 → GraduateStudent:1

Figure 3: HOG example (partial)

defined is unique and can be constructed from H and O in time $\mathcal{O}(|H| \cdot |\mathcal{R} \cup \bigcup_{p \in \mathcal{P}} dom(p)|)$.

Figure 3 shows the HOG associated with the union of the two RDF graphs shown in Figure 2 under a given set of Horn constraints.

Definition 6 (Integrability witness) *Suppose O_1, O_2 are two ontologies, H is a finite set of Horn clauses and NC is a finite set of negative clauses. An ontology graph $G = (V, E)$ is said to be a* witness to the integrability *of O_1, O_2 w.r.t. H and NC iff:*

(1) G contains no p-cycles[3] for any transitive property p,

(2) $HOG(O_1 \cup O_2, H) \sqsubseteq G$.

We define the *distance* between a witness G to the integrability of ontologies O_1, O_2 subject to Horn constraints H and negative constraints NC as: $d(G, \langle O_1, O_2, H\rangle) = |G_1| + |G_2| + |H| - |G|$ where G_i is the graph associated with ontology O_i.

Definition 7 (Minimal integrability witness) *Suppose O_1, O_2 are two ontologies with associated graphs G_1, G_2, H is a finite set of Horn constraints and NC is a finite set of negative constraints. A witness G to the integrability of G_1, G_2 w.r.t. H and NC is* minimal *if and only if there is no strict subgraph[4] G' of G which is a witness to the integrability of O_1, O_2 w.r.t. H and NC such that $0 \leq d(G', \langle O_1, O_2, H\rangle) \leq d(G, \langle O_1, O_2, H\rangle)$.*

Note that the minimal integrability witness is minimal in two respects: first, its graph structure is minimal (i.e. it does not contain unnecessary nodes and edges) and in addition, the distance according to the d-metric is minimized. Figure 4 shows a minimal witness to the integrability of the ontologies in Figure 2 using the Horn Constraints shown in Figure 3 and the negative constraints FACULTY:1$\not\to_S$STUDENT:2 and STUDENT:2$\not\to_S$FACULTY:1 discussed earlier.

[3]A p-cycle is a p-path of length 1 or more from a node to itself.
[4]We use the standard definition of subgraph.

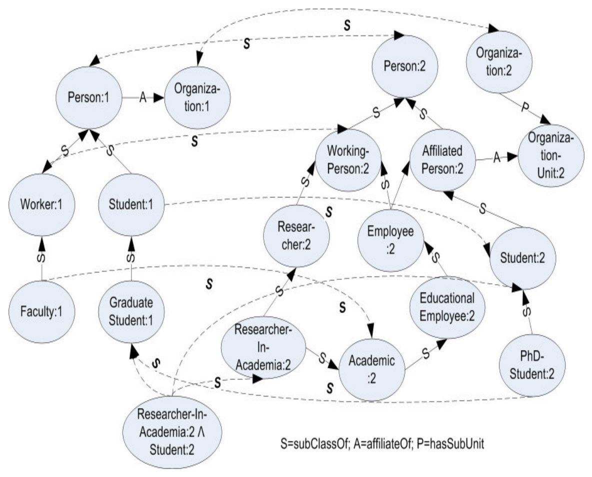


Figure 4: Minimal integrability witness

RDF Ontology Integration Problem. The RDF ontology integration problem is specified as follows:
INPUT: RDF ontology graphs O_1, O_2, a finite set H or Horn constraints, and a finite set NC of negative constraints.
OUTPUT: Return a minimal witness to the integrability of O_1, O_2 w.r.t. H and NC if a witness exists — otherwise return NULL.

The CROW Algorithm to integrate Ontologies

In this section, we present the CROW (Computing RDF Ontology Witness) algorithm to solve the RDF ontology integration problem. The CROW algorithm has four distinct phases:

(1) In the *pre-processing phase*, we rename the ontologies being merged so that terms in different ontologies are different (this is done by merely appending the ontology id to the end of each term).

(2) In the *graph construction phase*, CROW constructs $\mathsf{HOG}(O, H)$.

(3) In the *graph transformation phase*, the above graph is simplified using various kinds of strongly connected computations which we will describe shortly. Redundant edges are also eliminated during this phase.

(4) In the *negative constraint check phase*, we check whether the graph produced above satisfies the negative constraints. If not, we return "NULL" otherwise we return the graph produced in (3) above.

The heart of the algorithm is in step (3) above — we have already explained steps (1) and (2) earlier on in the paper. We apply two graph transformations.

Definition 8 *Suppose G is an RDF ontology graph, and p is any relation in $\mathcal{P}$ that is known to be transitive.*

- *A* p-strongly connected component *(or* p-SCC *for short) is any set S of nodes in G such that for all $a, b \in S$, there is a p-path from a to b.*
- *The operator $\varsigma_p(G)$ returns that graph $G' = (V', E')$ where:*
  - *V' is the set of all p-SCCs in G and*
  - *E' contains an edge from an SCC S_1 to an SCC S_2 labeled p iff there is an edge in G from some node in S_1 to some node from S_2 labeled p.*

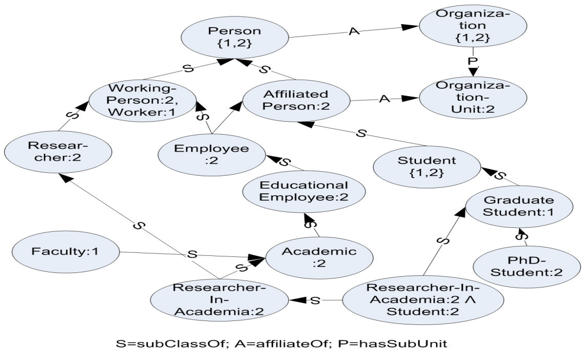


Figure 5: Intermediate result *CROW* phase (2)

$\varsigma_p(G)$ reduces all p-SCCs in G to a single node and then draws edges between two such reduced nodes if there was some node in one of the SCCs that was connected in the original graph G to some node in the other SCC. This is an intuitive method for reducing cycles for transitive relationships (e.g.: if WORKER:1 SUBCLASSOF WORKINGPERSON:2 and WORKINGPERSON:2 SUBCLASSOF WORKER:1, it would be safe to conclude that WORKER:1=WORKINGPERSON:2). Note that $\varsigma_p(G)$ is a lossy transformation as edges not labeled with p between nodes in the p-SCCs are lost. The second transformation - $\upsilon_p(O)$ - is used to eliminate redundant edges and thus obtain a minimal integration witness.

Definition 9 *An edge from a to b labeled p is said to be* redundant *w.r.t. graph G iff there is a p-path from a to b in the graph obtained by deleting this edge from G.*

The operator $\upsilon_p(G)$ returns a graph G' by eliminating as many redundant edges on transitive properties from G as possible.

There may be many ways in which redundant edges are removed from a graph G — all we require here is that $\upsilon_p(G)$ return any subgraph of G with no redundant edges.[5]

Theorem 1 *For any graph G:*

(1) $G \sqsubseteq \varsigma_p(G)$

(2) $G \sqsubseteq \upsilon_p(G)$.

The proof is based on the analysis of a single p-SCC or redundant edge elimination operation; each such operation preserves the $\sqsubseteq$ relation.

We are now ready to present the CROW algorithm (after the pre-processing step). Although the CROW algorithm is

[5]As such there can be many different implementations of $\upsilon_p(G)$. One such algorithm works as follows. First, it computes an adjacency matrix A for G. It then computes a path matrix B such that $B[i, j] = 1$ iff there is a path of length two or more from node i to j. This can be computed by a straightforward adaptation of Dijkstra's algorithm. Now remove all edges (i, j) for which $A[i, j] = 1$ and $B[i, j] = 1$.

designed for integrating two ontologies, a simple extension can be used to integrate sets of ontologies [6].

Algorithm: CROW(O_1, O_2, H, NC)
Inputs: Ontologies O_1, O_2, set of Horn constraints H and set of negative constraints NC. Let $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$ be the associated graphs.
Output: $G = (\mathcal{C}, E, \lambda)$, minimal witness to the integrability of O_1, O_2 w.r.t. H and NC or $NULL$ if no minimal witness exists.

```
1. $\mathcal{C} \leftarrow V_1 \cup V_2$;
2. $E \leftarrow E_1 \cup E_2$;
3. for $r_1 \wedge \ldots \wedge r_k \rightarrow r \in H$ in ascending order of $k$ do
4.   $\mathcal{C} \leftarrow \mathcal{C} \cup \{r_1, \ldots, r_k\}$;
5.   $E \leftarrow E \cup \{(\{r_1, \ldots, r_k\}, r)\}$;
6.   for all $n \in \mathcal{C}$, $n \subset \{r_1, \ldots, r_k\}$ do
7.     $E \leftarrow E \cup \{(\{r_1, \ldots, r_k\}, n)\}$;
8.   end
9. end
10. for all properties $p$
11.   if $p$ transitive[a] then
12.     while $\exists\, S$, $p$-SCC in $G$ do
13.       for $a \rightharpoonup b \in V$ do
(* by $\rightharpoonup$ we denote any type of negative constraint *)
14.         if $a \in S$ and $b \in S$ return NULL;
15.       end
16.       $G = \varsigma_p^S(G)$;[b]
17.     end
18.     $G = \upsilon_p(G)$;
19.   end
20.   for $cons \in V$ labeled with $p$ do
21.     if $G$ does not satisfy $cons$ then return NULL;
22.   end
23. end
24. return $O$;
```

[a]Specified a priori.
[b]Collapses only S into one node.

In lines 1-9, CROW first constructs the Horn Ontology Graph. The result of this first phase is depicted in Figure 5. The reader may notice that this graph contains both strongly connected components for the SUBCLASSOF hierarchy (STUDENT:1 $\leftrightarrow_S$ STUDENT:2) as well as redundant edges (PHD-STUDENT:2 SUBCLASSOF STUDENT:2). The reduction of strongly connected components and redundant edges is suitable only for transitive properties, as discussed in the Preliminaries Section. After these transformations (lines 16-18) the algorithm verifies the satisfiability of the negative constraints against the integrated ontology (lines 20-22). Note that some negative constraints can checked directly while detecting SCC, which optimizes the response time for cases when there is no minimal integrability witness. Figure 4 shows the result of integrating the ontologies in Figure 2 using the Horn constraints in Figure 3 and using the negative constraints: $NC = \{$*Faculty:1*$\not\rightharpoonup_S$ *Student:2*,*Student:2*$\not\rightharpoonup_S$ *Faculty:1*$\}$.

[6]For instance, by using CROW repeatedly or by merging all the ontology graphs at the same time.

The following result shows that CROW is correct.

Theorem 2 (Correctness of CROW) *CROW is correct, i.e. (i) If CROW(O_1, O_2, H, NC) does not return NULL, then it returns a minimal witness to the integrability of G_1, G_2 w.r.t. H and*
(ii) If CROW(G_1, G_2, H, NC) returns NULL, then there is no witness to the integrability of G_1, G_2 w.r.t. H.

Proof sketch. (i) Since all sets are finite and operations on lines 10-19 reduce the number of edges and/or nodes in the ontology, the algorithm will terminate. By the construction of the HOG in lines 1-9 we can easily see that $G_1 \sqsubseteq HOG(G_1 \cup G_2, H)$ and $G_2 \sqsubseteq HOG(G_1 \cup G_2, H)$. Furthermore, by Theorem 1 the graph transformations applied to $G = HOG(G_1 \cup G_2, H)$ preserve order. Also, G will satisfy NC according to lines 14, 20-22. Hence, the returned ontology G is an integration witness for G_1, G_2 w.r.t H and NC. Let us assume that G is not a minimal integrability witness and $\exists\, G'$ an integrability witness which is a strict subgraph of G. If G' has fewer nodes than G, it would violate the distance condition in Definition 7. Otherwise, if G' has fewer edges than G, since all redundant edges for transitive properties have been eliminated in G, that would mean G' violates condition (2) of Definition 6. Our hypothesis was false, hence G is a minimal integrability witness. (ii) Let us assume that there is a minimal integrability witness G that the CROW algorithm does not find. This means that CROW has returned NULL on either line 14 or 21. However, these directly correspond to checks against the negative constraints in NC. Therefore, G does not satisfy NC and cannot be a minimal integrability witness.

The following theorem states that CROW runs quite fast.

Theorem 3 (CROW complexity) *Let n_i, e_i be the number of nodes and respectively edges in G_i, $i \in \{1, 2\}$. Let n_h be the number of constraints in H and t the number of transitive properties. We denote by n, e the number of nodes and respectively edges in G. Then according to the CROW algorithm:*

(1) $n \leq n_1 + n_2 + n_h$ *and* $e \leq e_1 + e_2 + 2 \cdot n_h$.
(2) CROW is $\mathcal{O}(t \cdot n \cdot e \cdot (n + e))$.

The proof follows directly from the analysis of the CROW algorithm.

Implementation and Experiments

Our prototype is implemented in Java and currently consists of 3041 lines of code. It consists of the following components:

1. The *RDF Graph Generator*. We use Jena 2.2, a Semantic Web toolkit from Hewlett-Packard Labs, to digest an ontology data file. Then the *RDF Graph Generator* constructs an ontology graph by reading a set of RDF triples stored in Jena.
2. The *HOG Generator* generates a HOG for a set of Horn clauses.
3. The *Synthetic Data Generator* randomly generates ontology graphs, Horn constraints and negative constraints.

Ontology 1	Ontology 2	# constraints	$size_1$	$size_2$	$size_3$	$size_4$ (KB)	# transitive	Time (ms)
DAML - 64	DAML - 189	17	485	820	826	17.944	3	40.2
DAML - 4	DAML - 62	25	1171	2142	2147	52.564	1	32
DAML - 62	OntoBroker - ka2.daml	20	1019	1864	1868	51.972	1	22.2
SchemaWeb - 236	SchemaWeb - 197	8	959	1829	1841	51.152	1	24
DAML - 276	SchemaWeb - 162	32	1706	2938	2946	81.667	4	58.4

Table 1: Experimental results

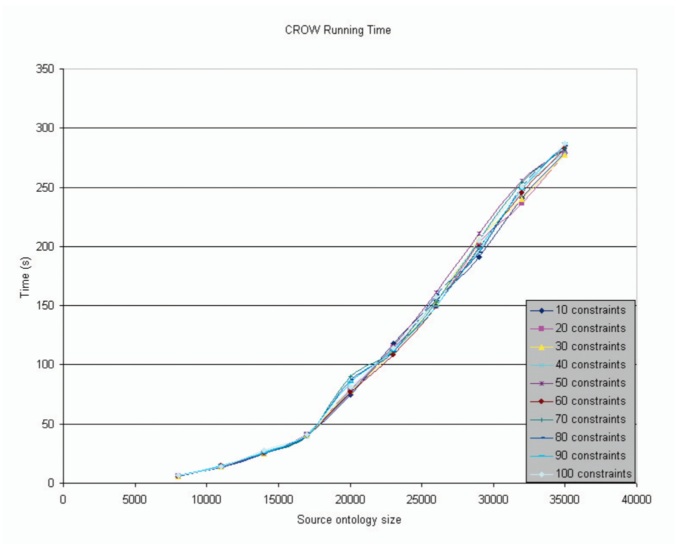


Figure 6: CROW running time

4. The *Ontology Integrator* integrates ontologies by taking a set of Horn constraints and a set of negative constraints into account.

Scalability experiments. We report here the performance of CROW algorithm on both real-life and synthetically generated ontologies. The experiments were run on a PC with 1.4GHz CPU, 524MB memory and Windows 2000 Professional platform. The times taken in the experiments include *the time to construct an HOG* and *the time to integrate ontologies*.

Scalability on real-life ontologies. We have applied our CROW algorithm to five pairs of similar ontologies[7] selected from DAML, SchemaWeb and OntoBroker ontology libraries. We list the detailed experimental results in Table 1. We have used various metrics for the size of ontologies (using the notations from Theorem 3): (i) $size_1 = n_1 + n_2 + n_h$,(ii) $size_2 = n_1 + n_2 + e_1 + e_2 + n_h$, (iii) $size_3$ is the number of nodes and edges in the HOG and (iv) $size_4$ is the total file size of the source ontologies. The running time of CROW algorithm is related to not only the number of terms and edges, but also to the number of transitive properties. We can see the effect of this factor in Table 1 by comparing the first and third pair of ontologies.

[7]Two on academic departments, one each on publications, travel and music.

Scalability on synthetic data. We randomly generated ten pairs of ontology graphs (each with only one transitive property). The total number of terms in each pair of ontologies varies from 8000 to 35000 with a step of 3000. For each pair, we generate ten sets of constraints (including Horn and negative constraints) whose sizes vary from 10 to 100 with a step of 10.

Figure 6 shows the scalability of CROW algorithm on the ten pairs of ontologies. As the total number of terms increased, we had a slightly faster than linear increase in the time taken to do the integration. The increase in running time is mainly due to the increase in number of terms (and the size of the HOG) more than to the increase in number of constraints, as shown in Figure 6.

Conclusions

In this paper, we have developed a formal model to integrate RDF ontologies in the presence of both Horn constraints and negative constraints. We have explained the concept of a witness to the integrability of two ontologies and we have developed the CROW algorithm to integrate two ontologies. We have tested CROW and found that it is very fast, both on ontologies at the DAML, SchemaWeba and OntoBroker sites, but also on synthetically generated ontologies.

References

Bouquet, P.; Giunchiglia, F.; van Harmelen, F.; Serafini, L.; and Stuckenschmidt, H. 2003. C-owl: Contextualizing ontologies. In *International Semantic Web Conference*, 164–179.

Calvanese, D.; Giacomo, G. D.; and Lenzerini, M. 2001. A framework for ontology integration. In *SWWS*, 303–316.

Lloyd, J. 1987. *Foundations of Logic Programming*. Springer Verlag.

McGuinness, D. L.; Fikes, R.; Rice, J.; and Wilder, S. 2000. An environment for merging and testing large ontologies. In *Proc. 7th Int. Conf. on Principles of Knowledge Representation and Reasoning*.

Mitra, P.; Wiederhold, G.; and Jannink, J. 1999. Semi-automatic integration of knowledge sources. In *Proc. of the 2nd Int. Conf. On Information FUSION'99*.

Stumme, G., and Maedche, A. 2001. Fca-merge: Bottom-up merging of ontologies. In *IJCAI*, 225–234.