

Discovering Simple Mappings Between Relational Database Schemas and Ontologies

Discovering Simple Mappings Between
Relational Database Schemas and Ontologies

Wei Hu and Yuzhong Qu

School of Computer Science and Engineering, Southeast University,
Nanjing 210096, P.R. China
{whu, yzqu}@seu.edu.cn

Abstract. Ontologies proliferate with the growth of the Semantic Web. However, most of data on the Web are still stored in relational databases. Therefore, it is important to establish interoperability between relational databases and ontologies for creating a Web of data. An effective way to achieve interoperability is finding mappings between relational database schemas and ontologies. In this paper, we propose a new approach to discovering simple mappings between a relational database schema and an ontology. It exploits simple mappings based on virtual documents, and eliminates incorrect mappings via validating mapping consistency. Additionally, it also constructs a special type of semantic mappings, called contextual mappings, which is useful for practical applications. Experimental results demonstrate that our approach performs well on several data sets from real world domains.

Wei Hu and Yuzhong Qu
School of Computer Science and
Engineering, Southeast University,
Nanjing 210096, P. R. China
{whu,yzqu}@seu.edu.cn
ISWC 2007, Spotlight Paper

Conference Paper Presentation by Joshua Taylor
October 9, 2008, Advanced Semantic Web
Rensselaer Polytechnic Institute
Troy NY 12180 USA

Friday, October 10, 2008

1

The motivation for recasting information from relational databases into Semantic Webbish formats (ontology languages based on DLs, e.g., OWL DL) is clear. The need is also significant—while the number of ontologies on the Semantic Web continues to grow, more than three quarters (77.3%) of the data on the web today is stored in relational databases. Fortunately, since relational databases can be formalized in first-order logic (FOL), and popular SW ontology languages are based on DLs, which correspond to a fragment of FOL, it is possible to construct **mappings** between relational databases and ontologies.

Review

2

Friday, October 10, 2008

2

Well designed relational databases tend to have two types of tables. Some tables, such as **Author** and **Paper** represent individuals of a particular class. Each row of such a table corresponds to some individual. One (or more) attributes of such a table serve as an identifier for the row: this attribute is called a **primary key**. Other attributes in the table specify information about the individuals described by the table. Other tables, such as **writes**, identify relationships between individuals represented by other tables. These typically have foreign keys as attributes; e.g., **writes** has an **aid** and **pid**, which are ids of authors and papers.

Ontologies typically have three types of entities, **classes**, **properties**, and **individuals**. Classes, such as **Paper**, **Conference Paper**, and **Author**, are arranged into a well-defined hierarchy, perhaps including, for example, subclass (subsumption) relationships and disjointness. Properties, such as **hasAuthor** and **hasID**, represent relationships between individuals. These may have specified domains, ranges, and a property might be a subproperty or inverse of another property.

Review

Relational Databases

Author

email: string

name: string

id: integer

Paper

id: integer

title: string

type: integer

2

Friday, October 10, 2008

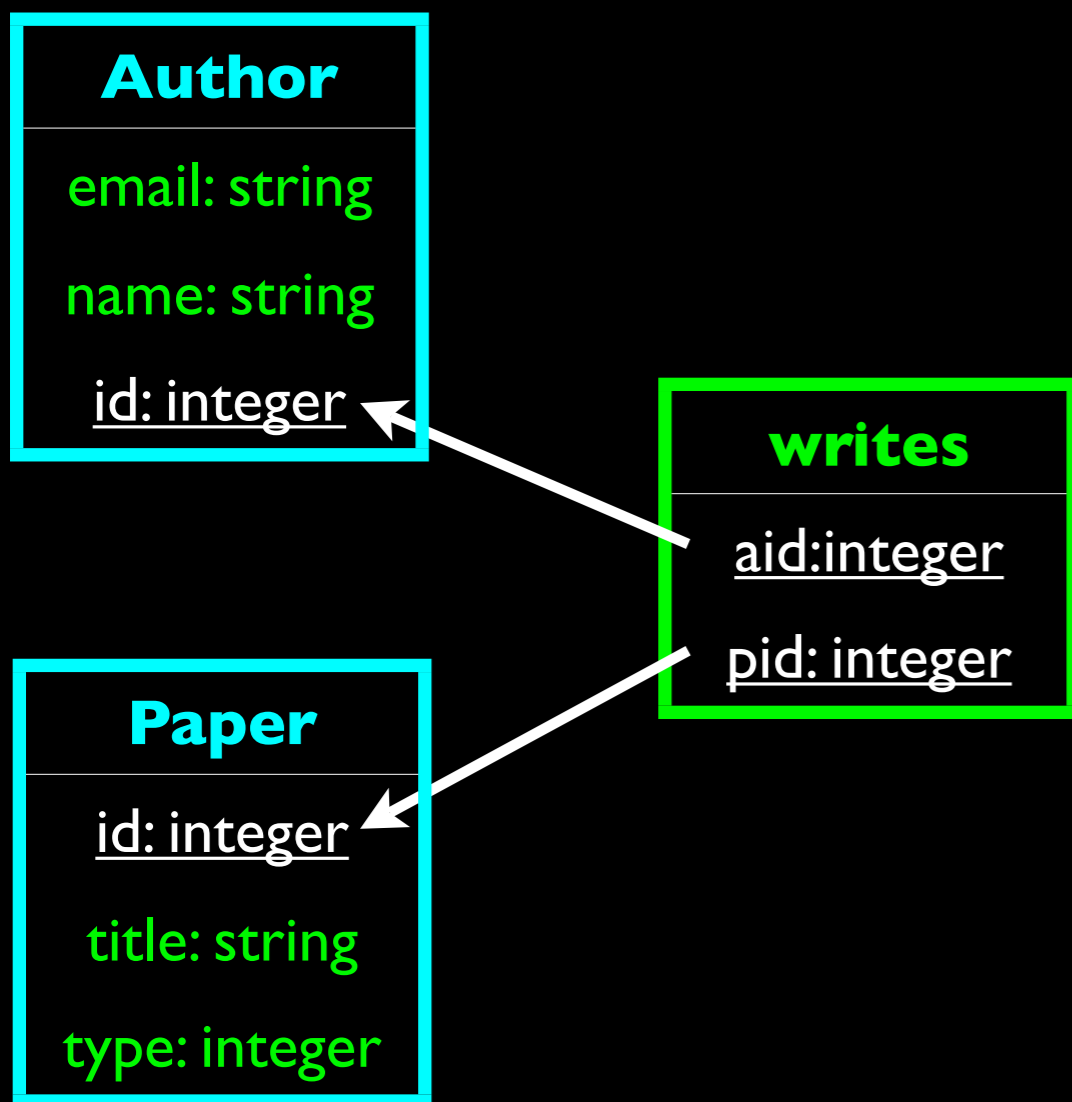
2

Well designed relational databases tend to have two types of tables. Some tables, such as **Author** and **Paper** represent individuals of a particular class. Each row of such a table corresponds to some individual. One (or more) attributes of such a table serve as an identifier for the row: this attribute is called a **primary key**. Other attributes in the table specify information about the individuals described by the table. Other tables, such as **writes**, identify relationships between individuals represented by other tables. These typically have foreign keys as attributes; e.g., **writes** has an aid and pid, which are ids of authors and papers.

Ontologies typically have three types of entities, **classes**, **properties**, and **individuals**. Classes, such as **Paper**, **Conference Paper**, and **Author**, are arranged into a well-defined hierarchy, perhaps including, for example, subclass (subsumption) relationships and disjointness. Properties, such as **hasAuthor** and **hasID**, represent relationships between individuals. These may have specified domains, ranges, and a property might be a subproperty or inverse of another property.

Review

Relational Databases



2

Friday, October 10, 2008

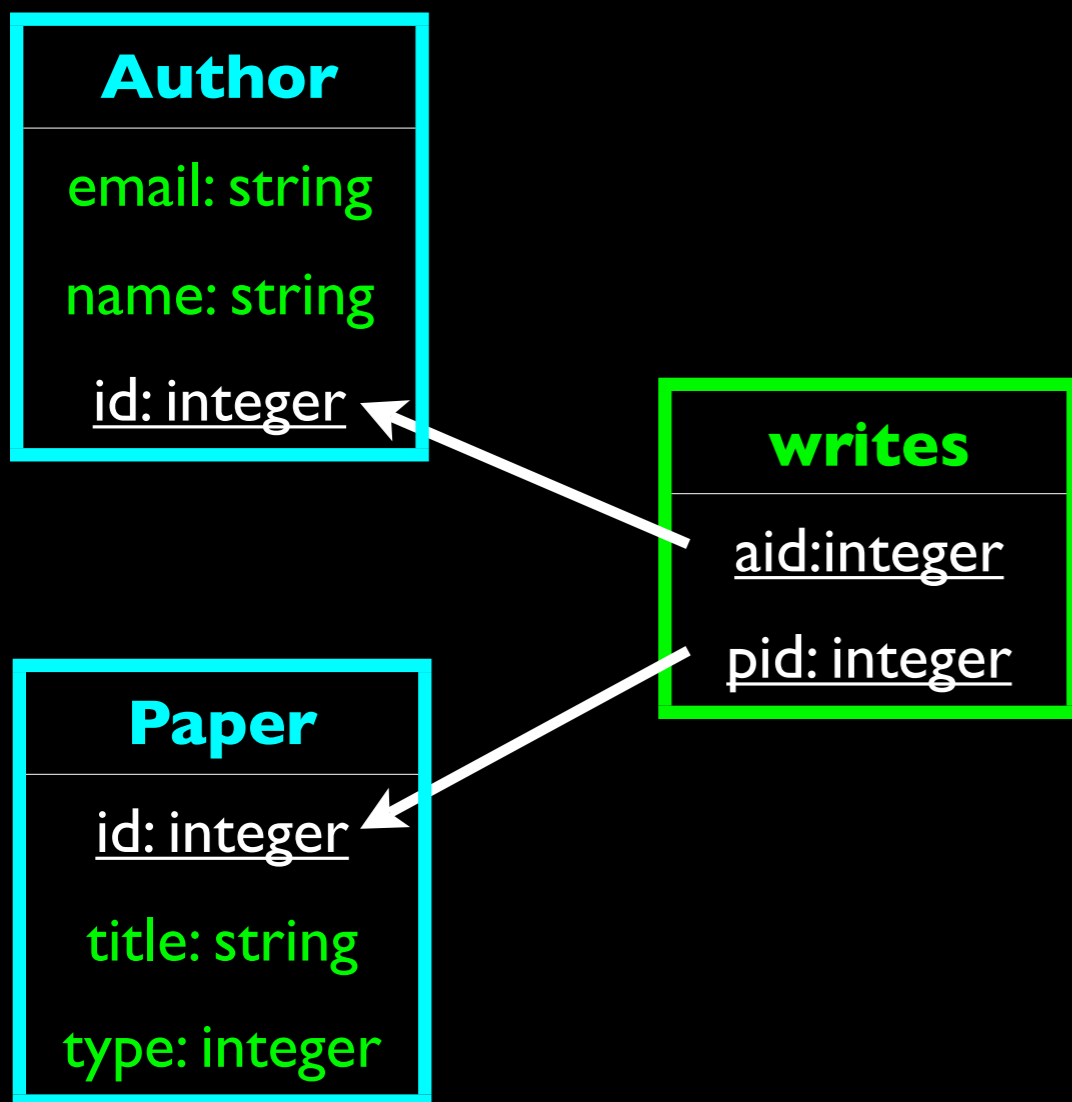
2

Well designed relational databases tend to have two types of tables. Some tables, such as **Author** and **Paper** represent individuals of a particular class. Each row of such a table corresponds to some individual. One (or more) attributes of such a table serve as an identifier for the row: this attribute is called a **primary key**. Other attributes in the table specify information about the individuals described by the table. Other tables, such as **writes**, identify relationships between individuals represented by other tables. These typically have foreign keys as attributes; e.g., **writes** has an `aid` and `pid`, which are ids of authors and papers.

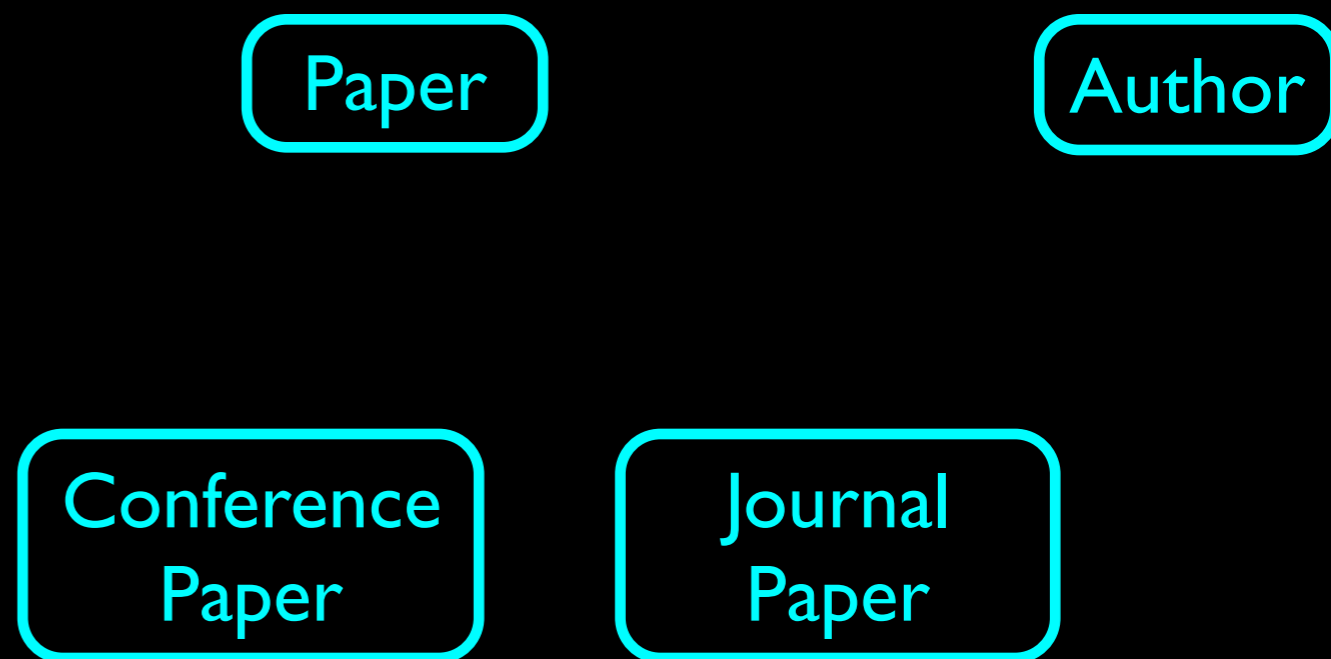
Ontologies typically have three types of entities, **classes**, **properties**, and **individuals**. Classes, such as **Paper**, **Conference Paper**, and **Author**, are arranged into a well-defined hierarchy, perhaps including, for example, subclass (subsumption) relationships and disjointness. Properties, such as **hasAuthor** and **hasID**, represent relationships between individuals. These may have specified domains, ranges, and a property might be a subproperty or inverse of another property.

Review

Relational Databases



Ontology Languages



2

Friday, October 10, 2008

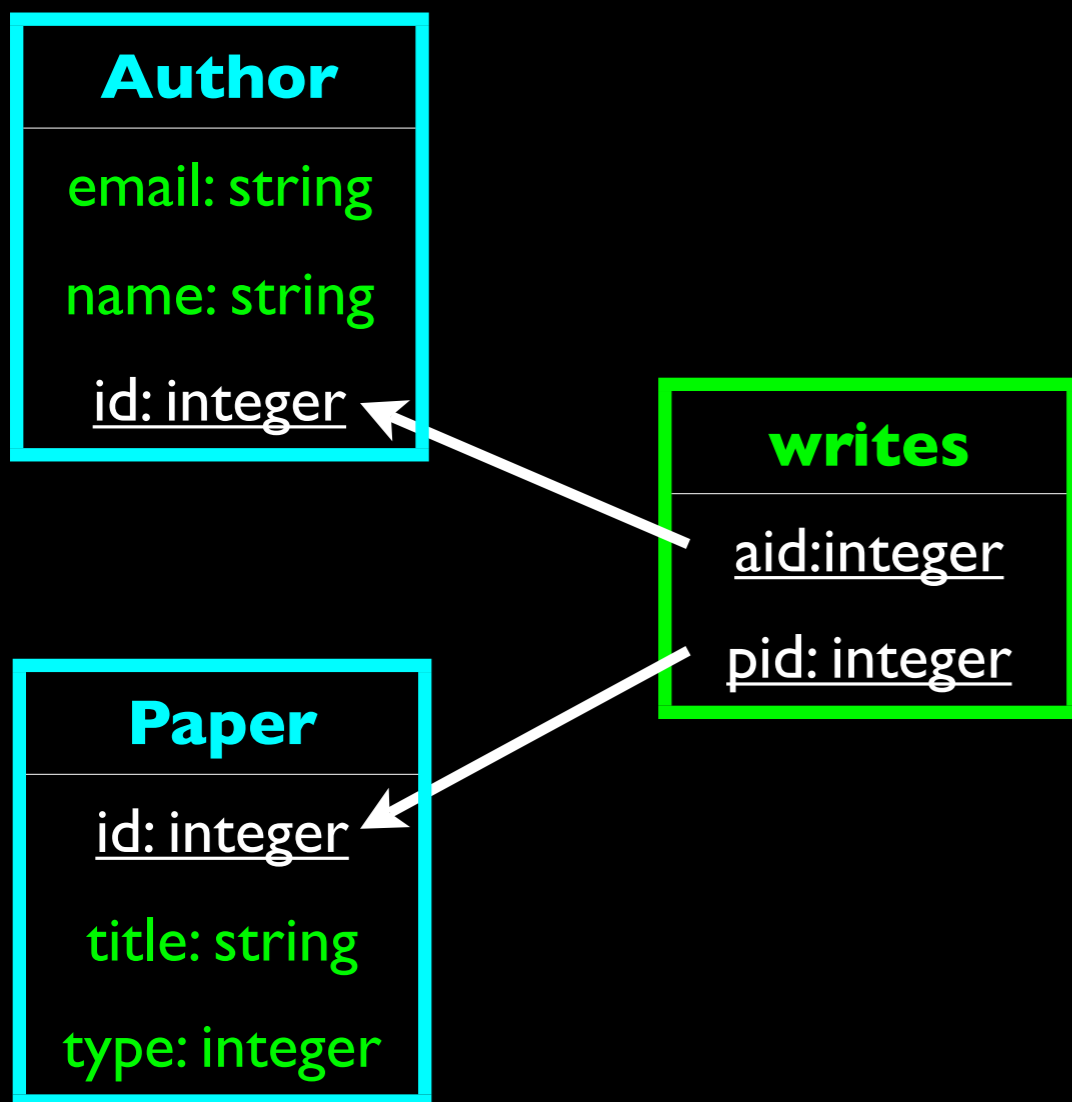
2

Well designed relational databases tend to have two types of tables. Some tables, such as **Author** and **Paper** represent individuals of a particular class. Each row of such a such a table corresponds to some individual. One (or more) attributes of such a table serve as an identifier for the row: this attribute is called a **primary key**. Other attributes in the table specify information about the individuals described by the table. Other tables, such as **writes**, identify relationships between individuals represented by other tables. These typically have foreign keys as attributes; e.g., **writes** has an `aid` and `pid`, which are ids of authors and papers.

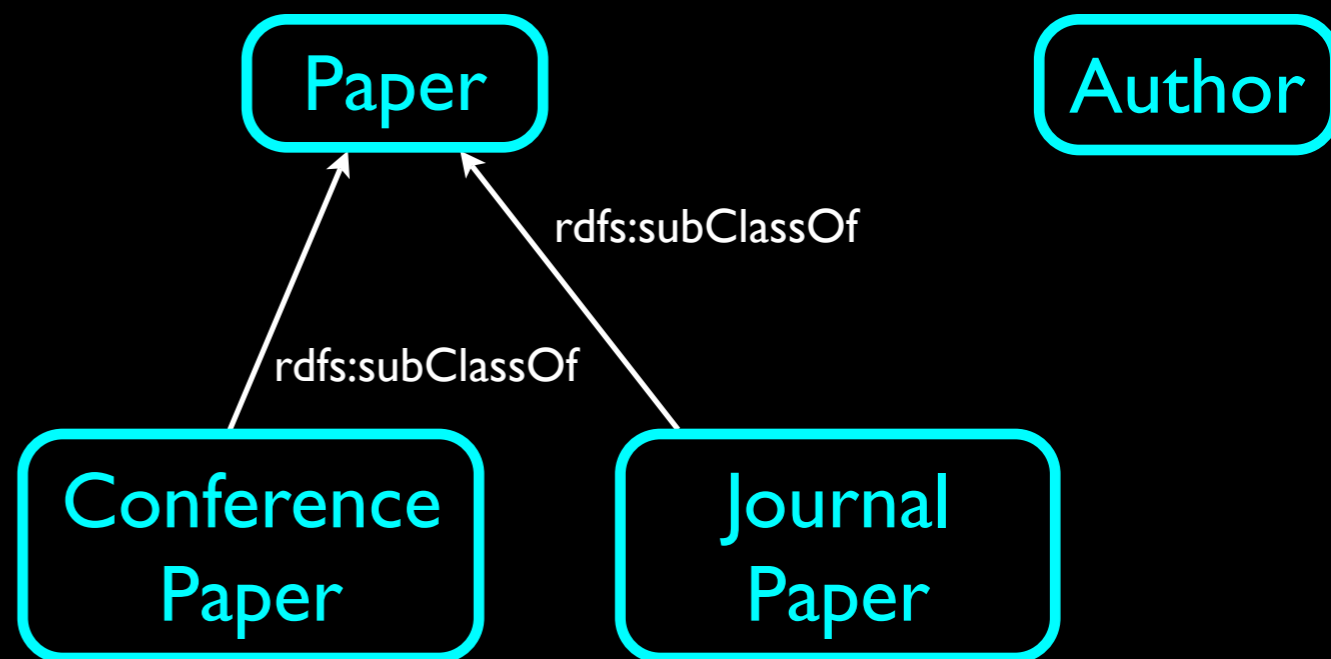
Ontologies typically have three types of entities, **classes**, **properties**, and **individuals**. Classes, such as **Paper**, **Conference Paper**, and **Author**, are arranged into a well-defined hierarchy, perhaps including, for example, subclass (subsumption) relationships and disjointness. Properties, such as **hasAuthor** and **hasID**, represent relationships between individuals. These may have specified domains, ranges, and a property might be a subproperty or inverse of another property.

Review

Relational Databases



Ontology Languages



2

Friday, October 10, 2008

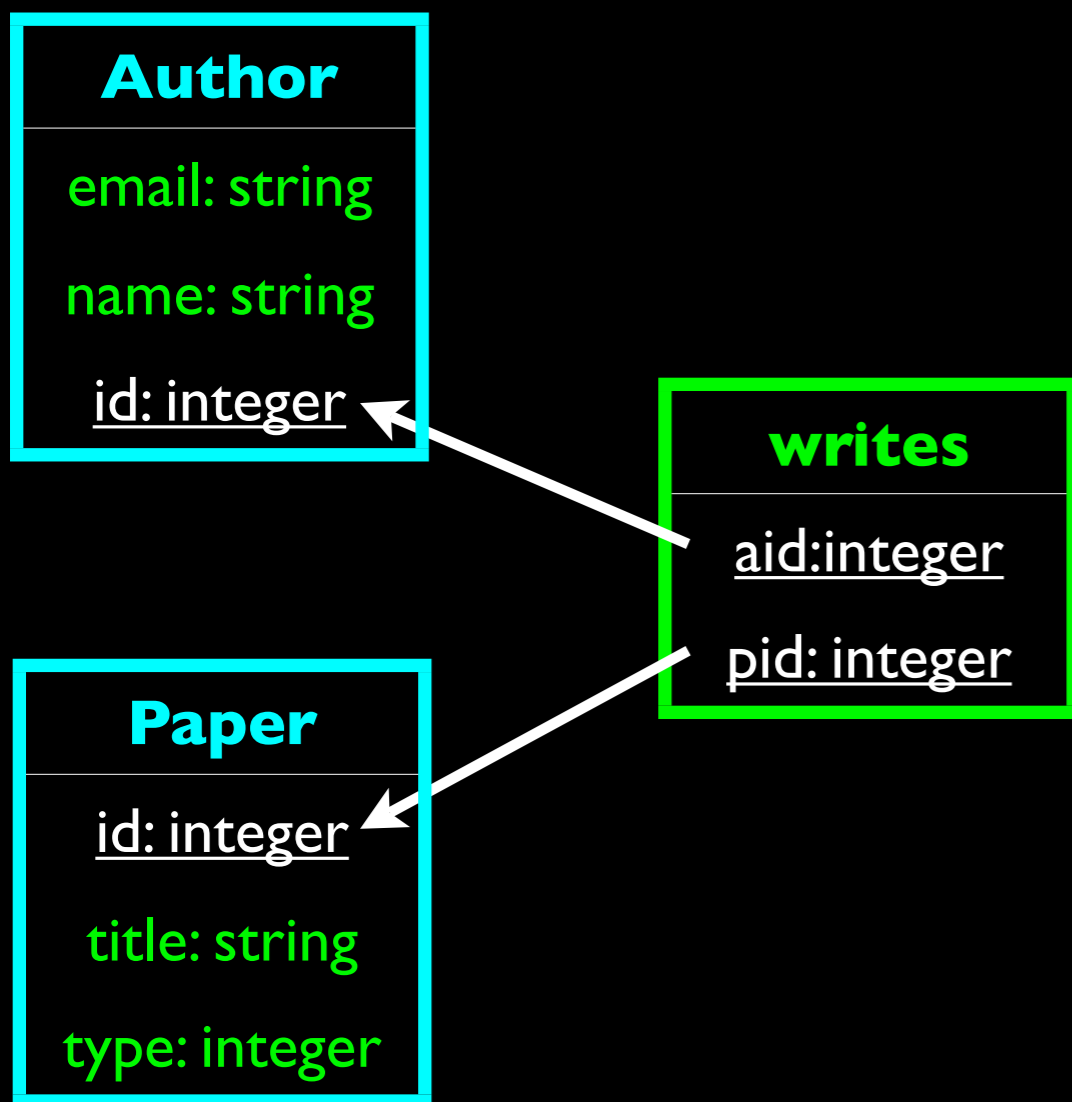
2

Well designed relational databases tend to have two types of tables. Some tables, such as **Author** and **Paper** represent individuals of a particular class. Each row of such a such a table corresponds to some individual. One (or more) attributes of such a table serve as an identifier for the row: this attribute is called a **primary key**. Other attributes in the table specify information about the individuals described by the table. Other tables, such as **writes**, identify relationships between individuals represented by other tables. These typically have foreign keys as attributes; e.g., **writes** has an `aid` and `pid`, which are ids of authors and papers.

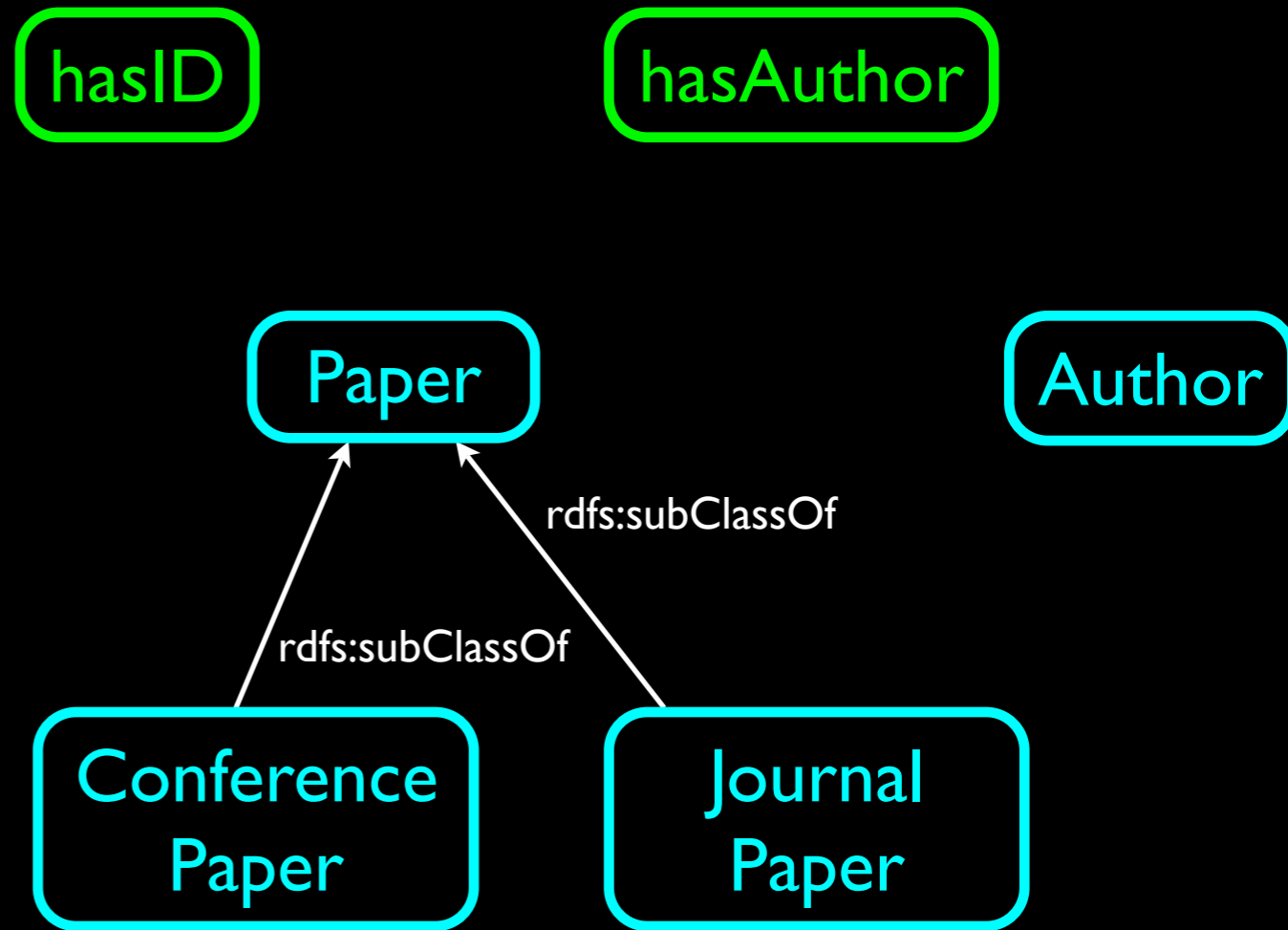
Ontologies typically have three types of entities, **classes**, **properties**, and **individuals**. Classes, such as **Paper**, **Conference Paper**, and **Author**, are arranged into a well-defined hierarchy, perhaps including, for example, subclass (subsumption) relationships and disjointness. Properties, such as **hasAuthor** and **hasID**, represent relationships between individuals. These may have specified domains, ranges, and a property might be a subproperty or inverse of another property.

Review

Relational Databases



Ontology Languages



2

Friday, October 10, 2008

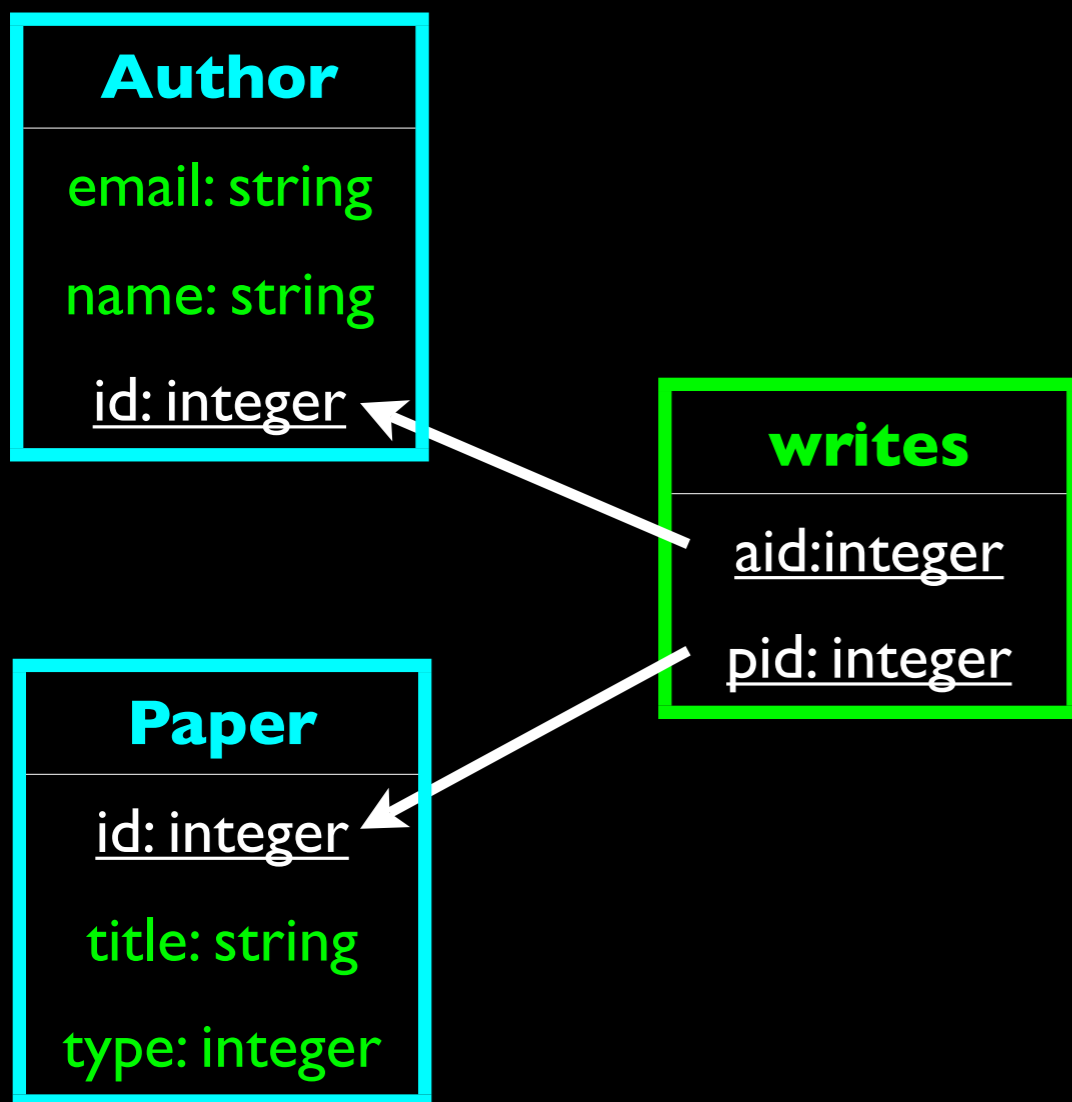
2

Well designed relational databases tend to have two types of tables. Some tables, such as **Author** and **Paper** represent individuals of a particular class. Each row of such a such a table corresponds to some individual. One (or more) attributes of such a table serve as an identifier for the row: this attribute is called a **primary key**. Other attributes in the table specify information about the individuals described by the table. Other tables, such as **writes**, identify relationships between individuals represented by other tables. These typically have foreign keys as attributes; e.g., **writes** has an `aid` and `pid`, which are ids of authors and papers.

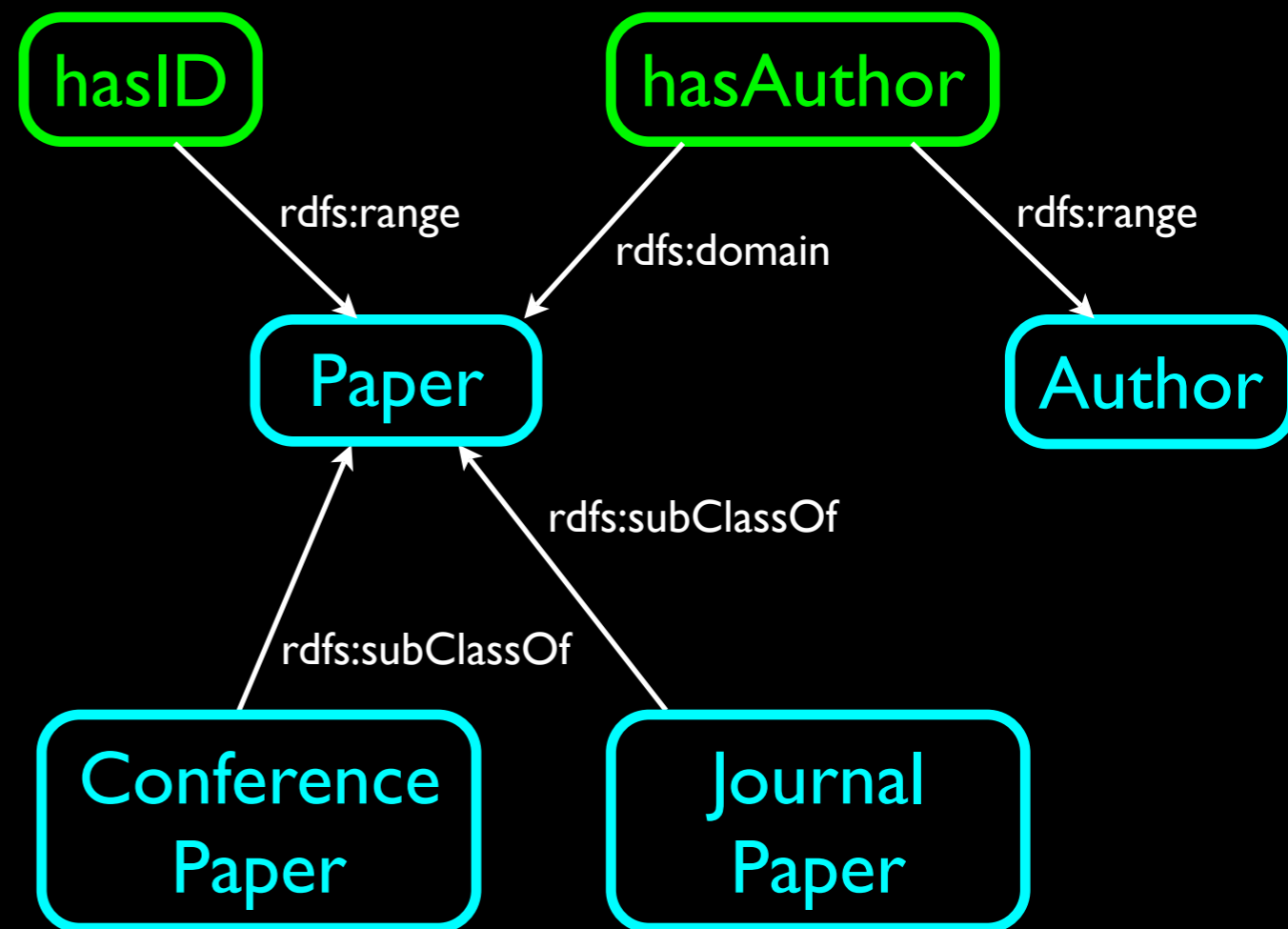
Ontologies typically have three types of entities, **classes**, **properties**, and **individuals**. Classes, such as **Paper**, **Conference Paper**, and **Author**, are arranged into a well-defined hierarchy, perhaps including, for example, subclass (subsumption) relationships and disjointness. Properties, such as **hasAuthor** and **hasID**, represent relationships between individuals. These may have specified domains, ranges, and a property might be a subproperty or inverse of another property.

Review

Relational Databases



Ontology Languages



2

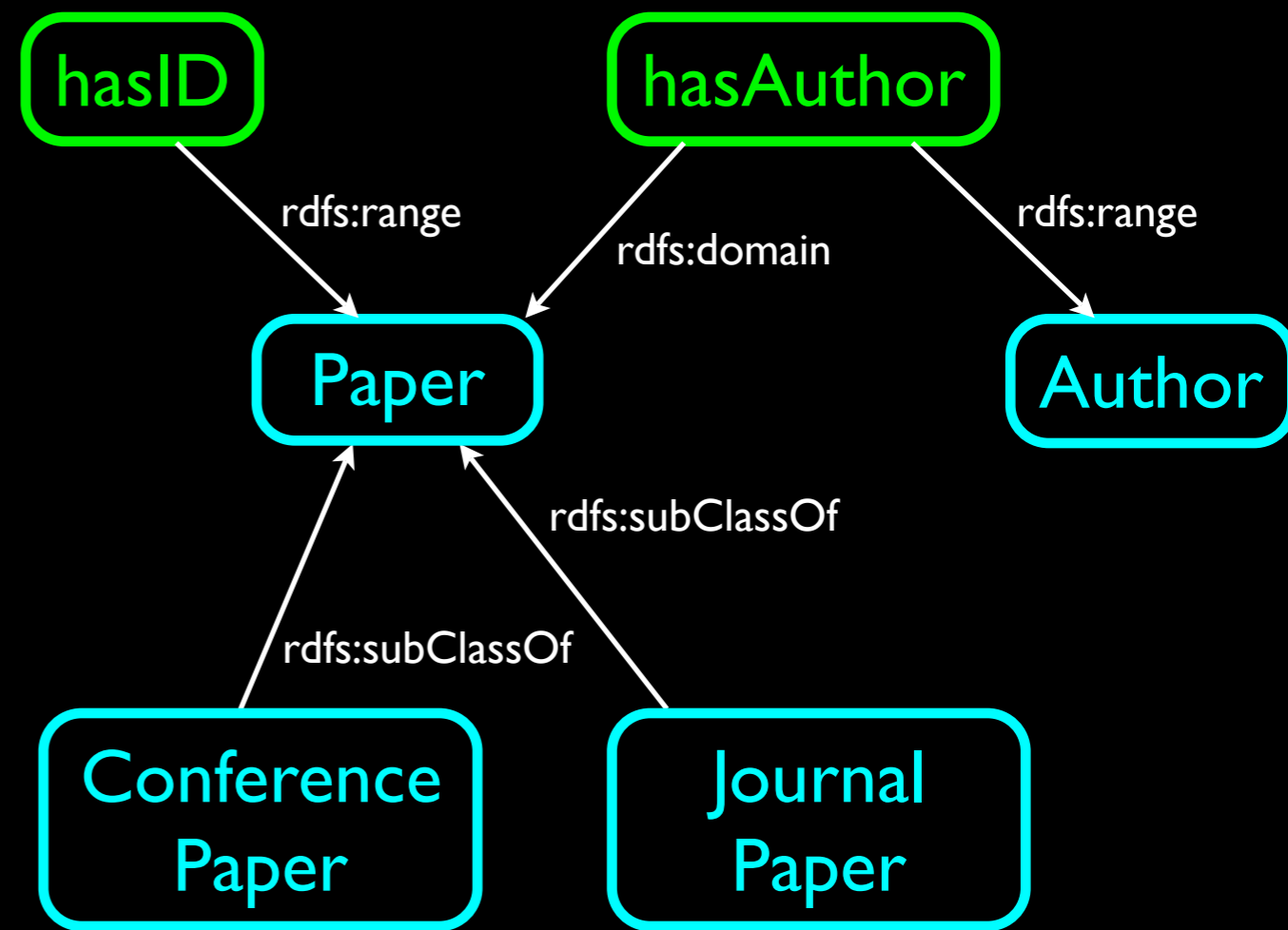
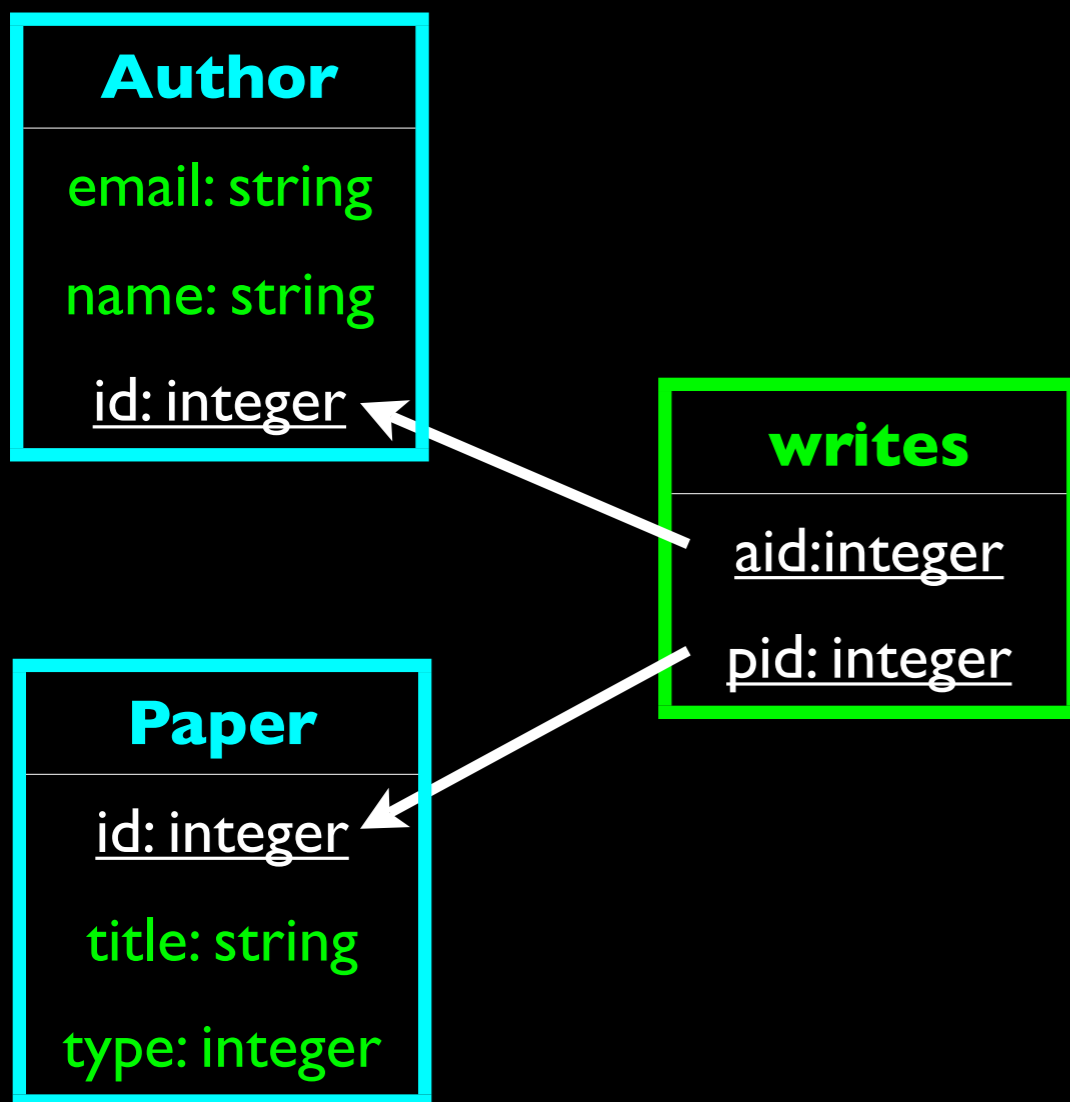
Friday, October 10, 2008

2

Well designed relational databases tend to have two types of tables. Some tables, such as **Author** and **Paper** represent individuals of a particular class. Each row of such a such a table corresponds to some individual. One (or more) attributes of such a table serve as an identifier for the row: this attribute is called a **primary key**. Other attributes in the table specify information about the individuals described by the table. Other tables, such as **writes**, identify relationships between individuals represented by other tables. These typically have foreign keys as attributes; e.g., **writes** has an `aid` and `pid`, which are ids of authors and papers.

Ontologies typically have three types of entities, **classes**, **properties**, and **individuals**. Classes, such as **Paper**, **Conference Paper**, and **Author**, are arranged into a well-defined hierarchy, perhaps including, for example, subclass (subsumption) relationships and disjointness. Properties, such as **hasAuthor** and **hasID**, represent relationships between individuals. These may have specified domains, ranges, and a property might be a subproperty or inverse of another property.

Goal



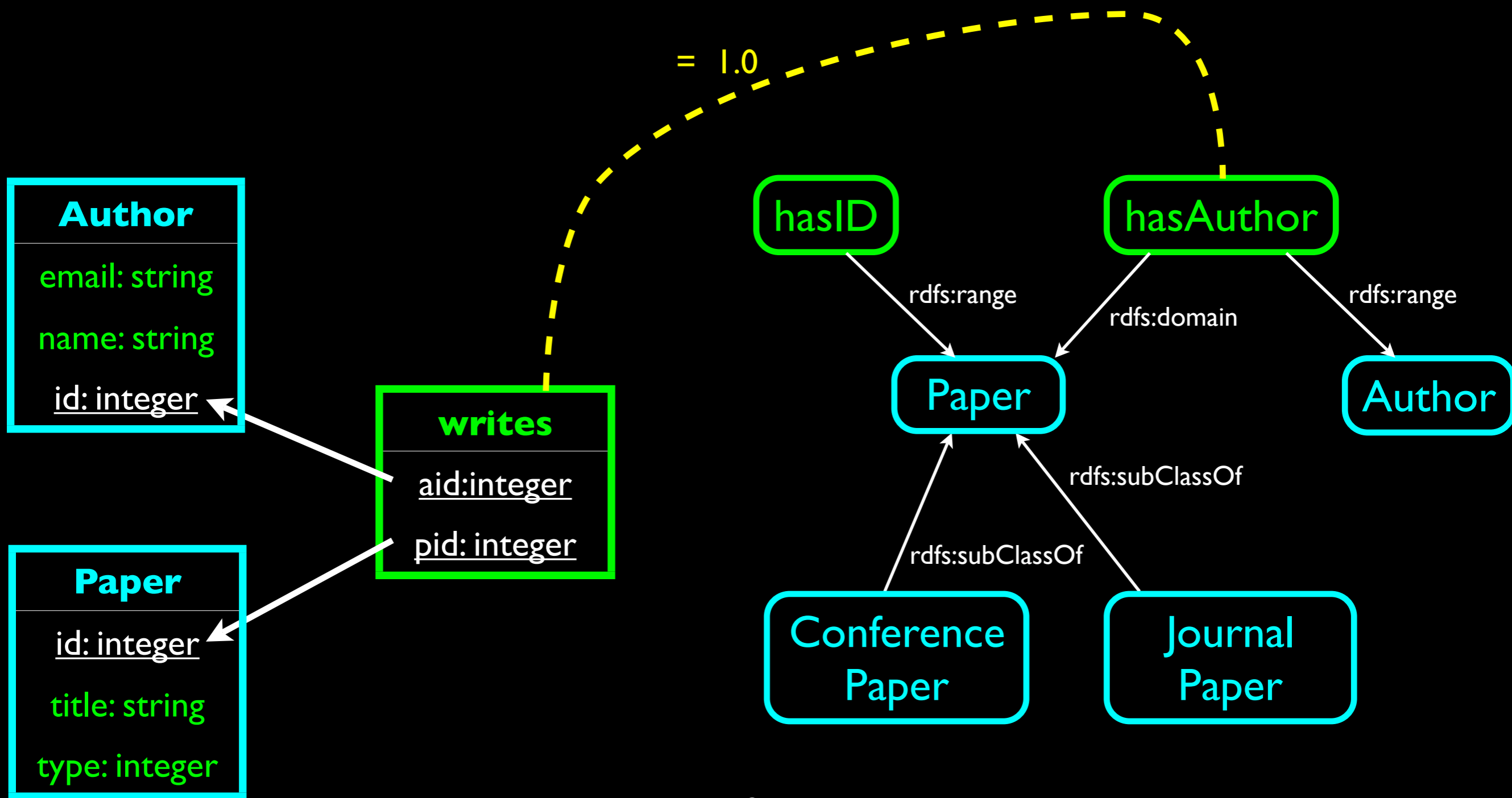
3

Friday, October 10, 2008

3

- We recognize that **write** and **hasAuthor** are intended to represent exactly the same information, and so, ideally, they could 'magically' compute their equivalence, and with perfect confidence.
- Similarly, **hasID**, the range of which is **Paper**, seems like a relationship equivalent to the id: integer attribute of the **Paper** table. Could this also be discovered automatically?
- We might recognize that the **Paper** table corresponds to the **Paper** class, but could we determine, at least with some certainty that the class corresponding to the table **Paper** subsumes the class **Conference Paper**?

Goal



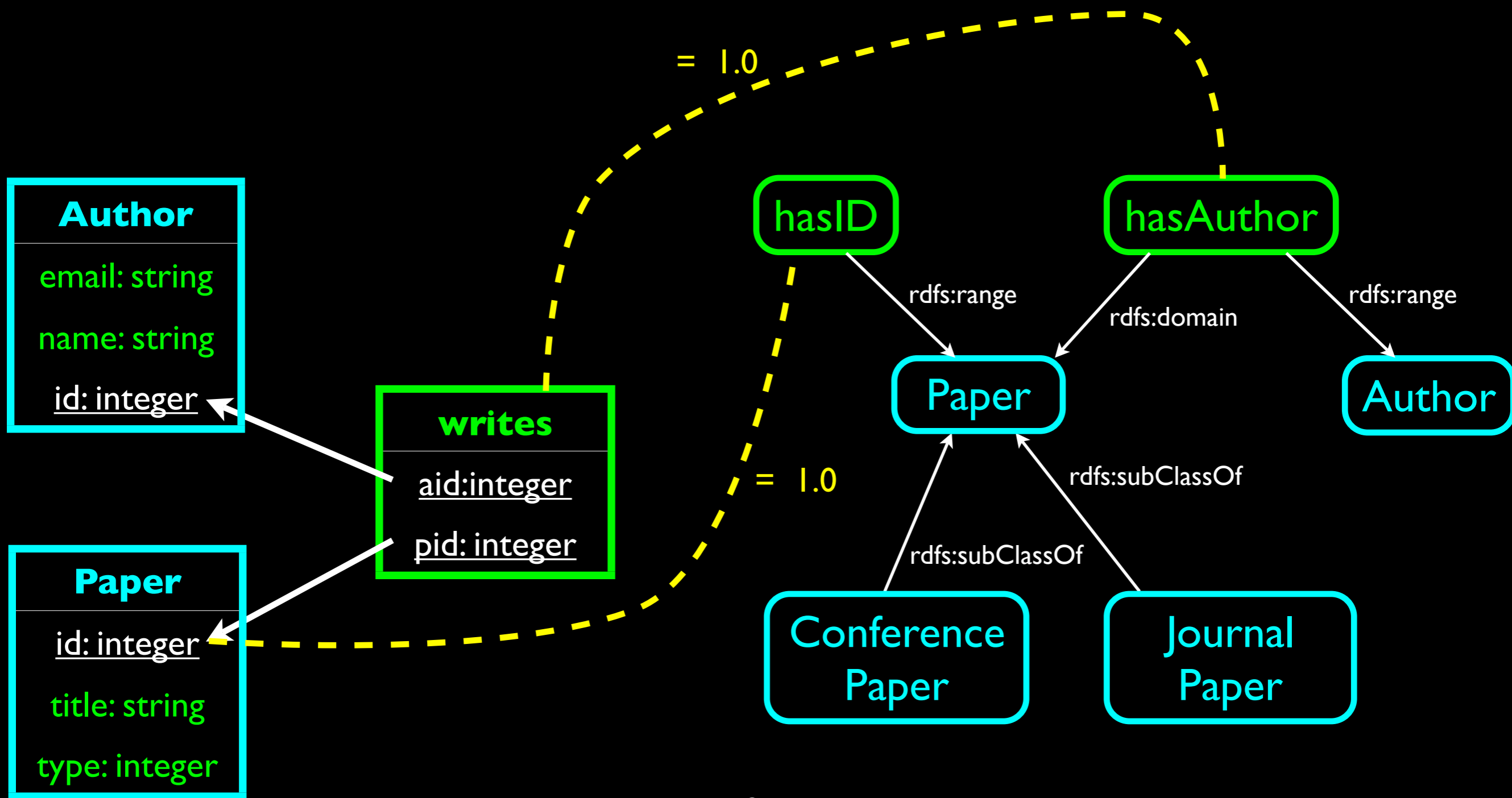
3

Friday, October 10, 2008

3

- We recognize that **write** and **hasAuthor** are intended to represent exactly the same information, and so, ideally, they could 'magically' compute their equivalence, and with perfect confidence.
- Similarly, **hasID**, the range of which is **Paper**, seems like a relationship equivalent to the id: integer attribute of the **Paper** table. Could this also be discovered automatically?
- We might recognize that the **Paper** table corresponds to the **Paper** class, but could we determine, at least with some certainty that the class corresponding to the table **Paper** subsumes the class **Conference Paper**?

Goal



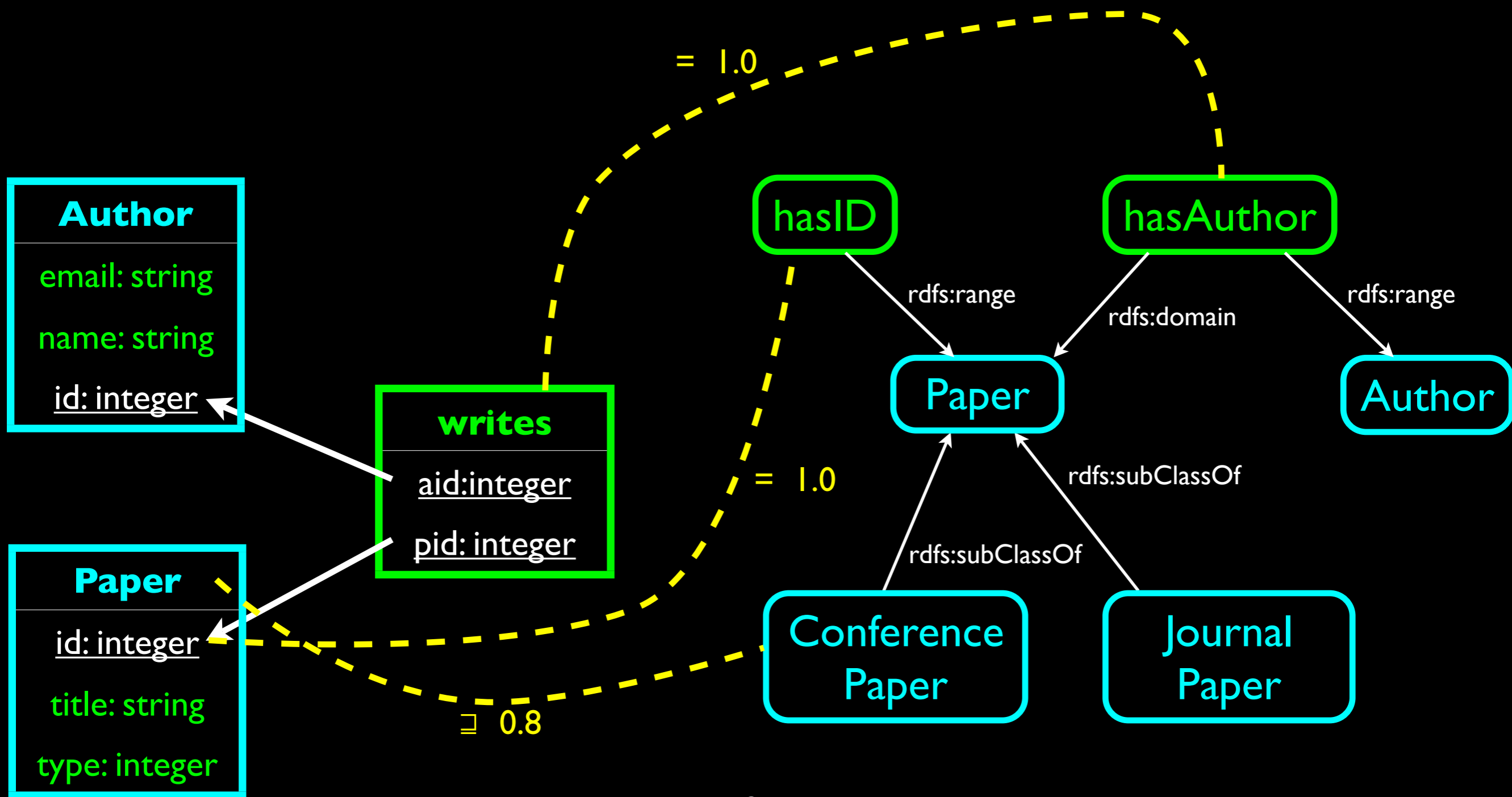
3

Friday, October 10, 2008

3

- We recognize that **write** and **hasAuthor** are intended to represent exactly the same information, and so, ideally, they could 'magically' compute their equivalence, and with perfect confidence.
- Similarly, **hasID**, the range of which is **Paper**, seems like a relationship equivalent to the id: integer attribute of the **Paper** table. Could this also be discovered automatically?
- We might recognize that the **Paper** table corresponds to the **Paper** class, but could we determine, at least with some certainty that the class corresponding to the table **Paper** subsumes the class **Conference Paper**?

Goal



3

Friday, October 10, 2008

3

- We recognize that **write** and **hasAuthor** are intended to represent exactly the same information, and so, ideally, they could 'magically' compute their equivalence, and with perfect confidence.
- Similarly, **hasID**, the range of which is **Paper**, seems like a relationship equivalent to the id:integer attribute of the **Paper** table. Could this also be discovered automatically?
- We might recognize that the **Paper** table corresponds to the **Paper** class, but could we determine, at least with some certainty that the class corresponding to the table **Paper** subsumes the class **Conference Paper**?

But How?

1. Classify entity types.
2. Discover *simple* mappings.
3. Validate mapping consistency.
4. Construct *contextual* mappings.

1. Database and ontology entities are partitioned so as to limit the mapping search space. This goes beyond the simple relation and attribute to class and property mapping discussed in **Definition 3**.

2. Simple mappings are found using “virtual documents” (vectors corresponding to entities) and confidence metrics computed from the virtual documents.

3. The consistency of the generated mappings is checked against type restrictions for attributes and properties. Some inference is incorporated into this phase.

4. Contextual mappings are sophisticated mappings that could be used to construct views for information using selection constraints.

Classifying Entity Types

1. $\{SER \cup WER\} \times C$

e.g., **Author**

2. $\{RRR \cup SRR\} \times P_0$

e.g., **writes**

3. $FKA \times P_0$

e.g., publisher:integer

4. $FKA \times \{P_D \cup P_0\}$

e.g., title:string

As mentioned before, database tables are either entity tables or relationship tables. The difference between strong and weak entity tables, and between regular and specific relationship tables doesn't seem to have large role here, so we can ignore the distinctions. Foreign key attributes, however, are those whose values indicates rows in other entity tables, and so indicate object properties. Non-Foreign key attributes may have literal values or identifiers for other entities, and so may correspond to object or datatype properties.

Discovering Simple Mappings

6

Friday, October 10, 2008

6

Simple mappings are “discovered” by building “virtual documents” (statistical representations), or VDs, of entities in the database and ontology, and then the confidence of a mapping between two entities is the cosine of these statistical representations. How these are defined is different for the various types of entities, but once the VDs are built, computing confidence is a simple cosine operation.

In the paper, the description of an entities is simply the name of the entity. The authors do not specify, but presumably the descriptions in real applications might use extra annotations, like `rdfs:label`, or `rdfs:comment`.

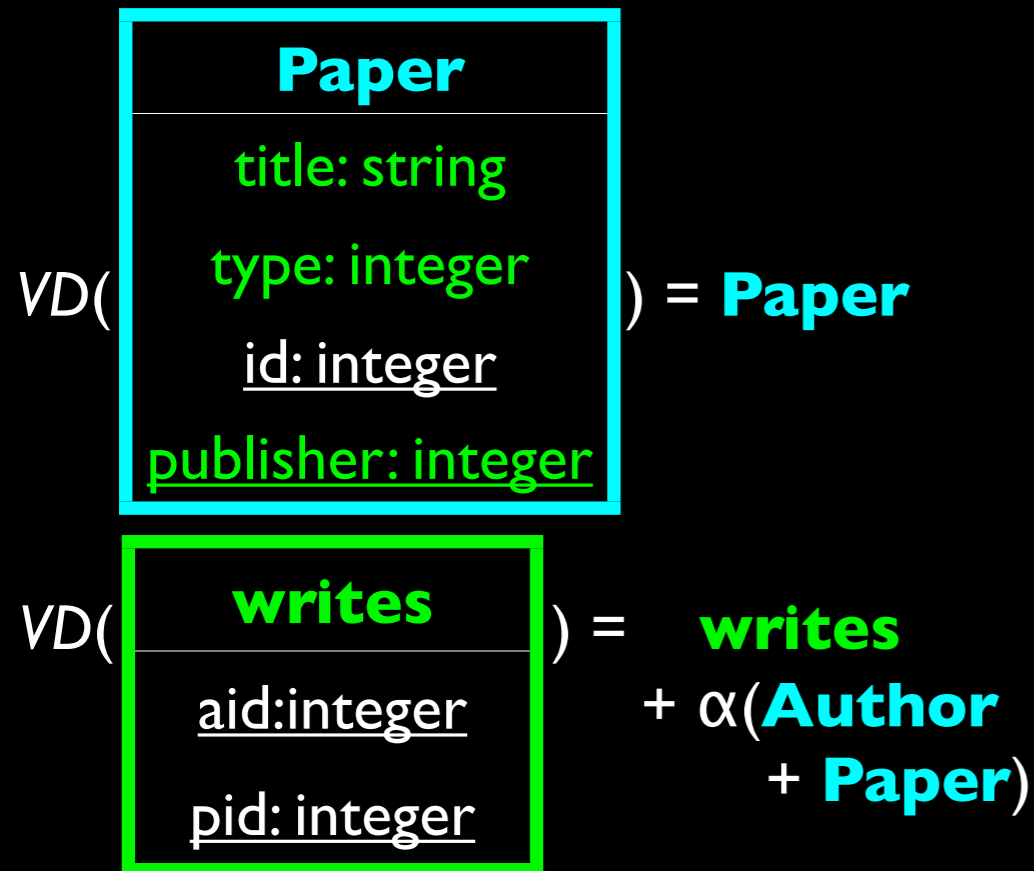
Discovering Simple Mappings

$VD(\begin{array}{l} \mathbf{Paper} \\ \text{title: string} \\ \text{type: integer} \\ \text{id: integer} \\ \text{publisher: integer} \end{array}) = \mathbf{Paper}$

Simple mappings are “discovered” by building “virtual documents” (statistical representations), or VDs, of entities in the database and ontology, and then the confidence of a mapping between two entities is the cosine of these statistical representations. How these are defined is different for the various types of entities, but once the VDs are built, computing confidence is a simple cosine operation.

In the paper, the description of an entities is simply the name of the entity. The authors do not specify, but presumably the descriptions in real applications might use extra annotations, like `rdfs:label`, or `rdfs:comment`.

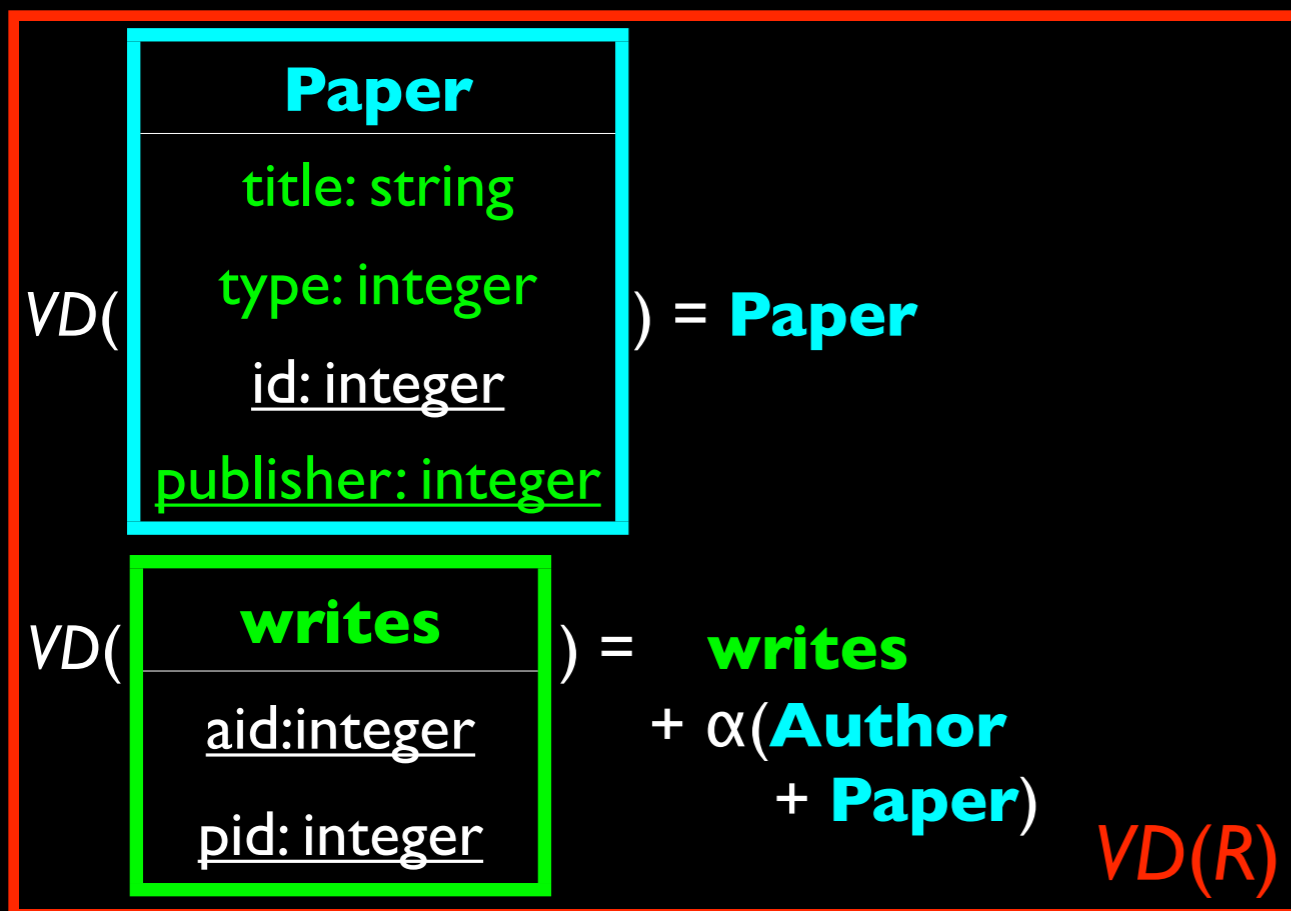
Discovering Simple Mappings



Simple mappings are “discovered” by building “virtual documents” (statistical representations), or VDs, of entities in the database and ontology, and then the confidence of a mapping between two entities is the cosine of these statistical representations. How these are defined is different for the various types of entities, but once the VDs are built, computing confidence is a simple cosine operation.

In the paper, the description of an entities is simply the name of the entity. The authors do not specify, but presumably the descriptions in real applications might use extra annotations, like `rdfs:label`, or `rdfs:comment`.

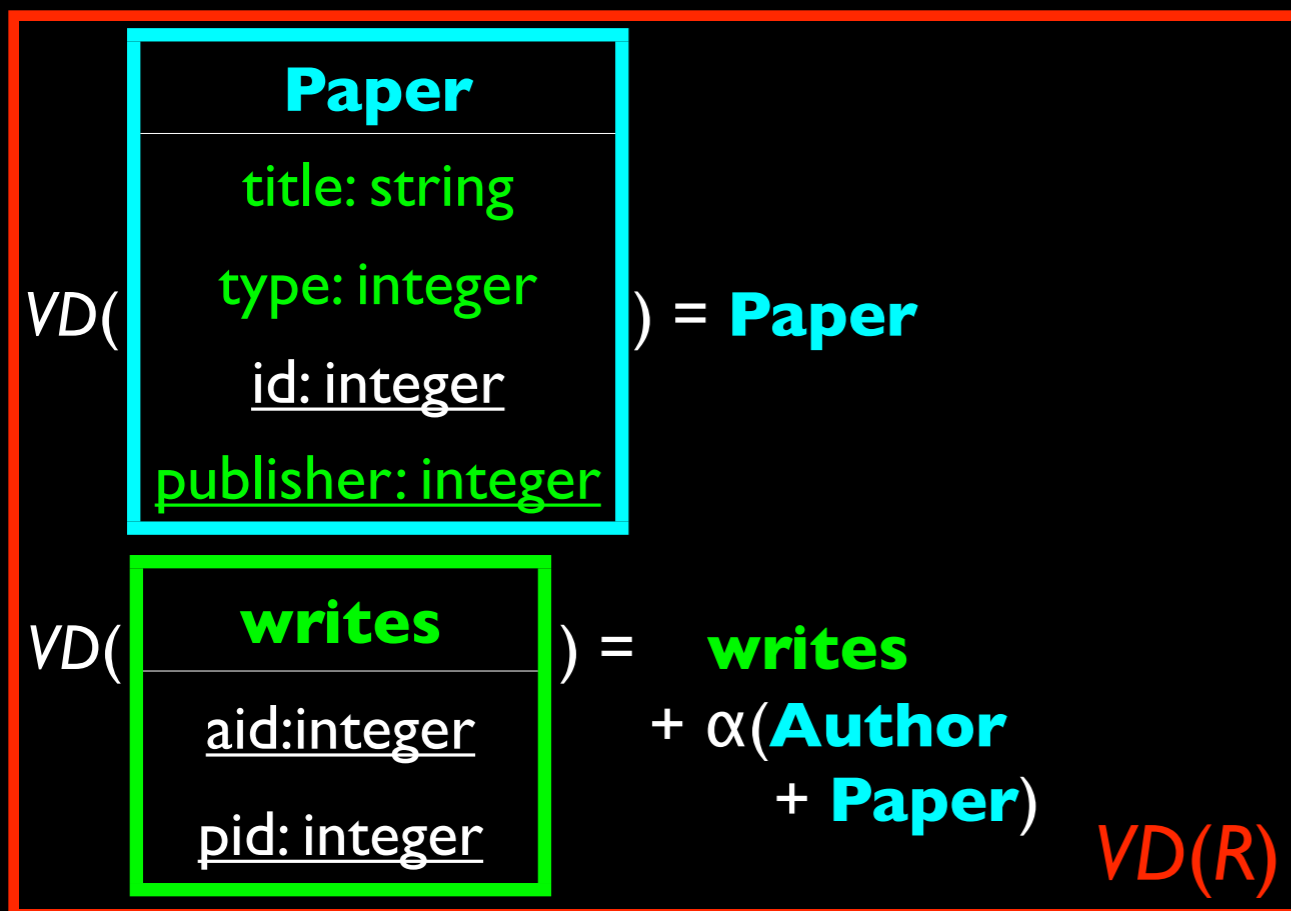
Discovering Simple Mappings



Simple mappings are “discovered” by building “virtual documents” (statistical representations), or VDs, of entities in the database and ontology, and then the confidence of a mapping between two entities is the cosine of these statistical representations. How these are defined is different for the various types of entities, but once the VDs are built, computing confidence is a simple cosine operation.

In the paper, the description of an entities is simply the name of the entity. The authors do not specify, but presumably the descriptions in real applications might use extra annotations, like `rdfs:label`, or `rdfs:comment`.

Discovering Simple Mappings

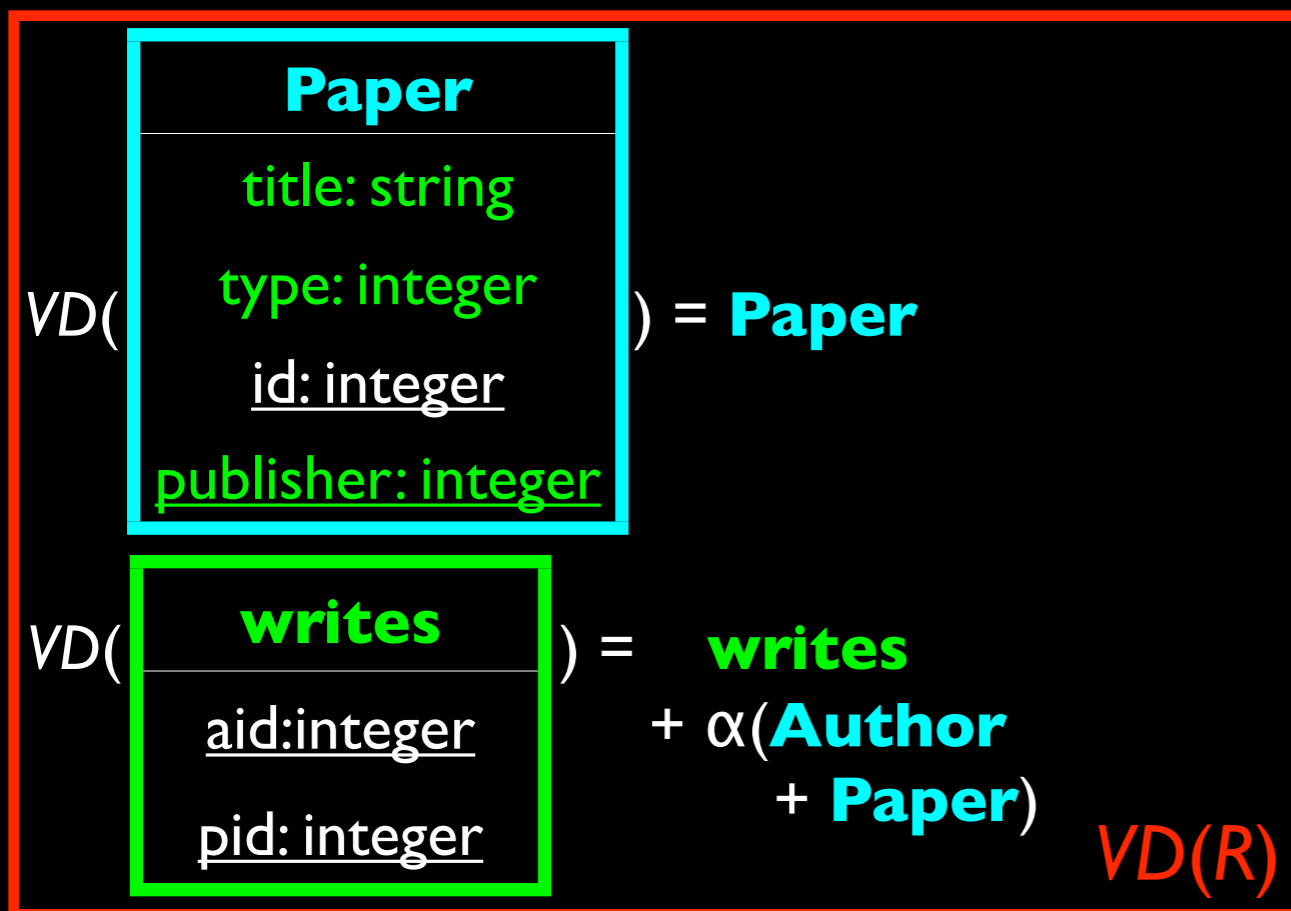


$$VD(\underline{\text{publisher: integer}}) = \underline{\text{publisher}} + \alpha(\mathbf{Paper} + \mathbf{Publisher})$$

Simple mappings are “discovered” by building “virtual documents” (statistical representations), or VDs, of entities in the database and ontology, and then the confidence of a mapping between two entities is the cosine of these statistical representations. How these are defined is different for the various types of entities, but once the VDs are built, computing confidence is a simple cosine operation.

In the paper, the description of an entities is simply the name of the entity. The authors do not specify, but presumably the descriptions in real applications might use extra annotations, like rdfs:label, or rdfs:comment.

Discovering Simple Mappings



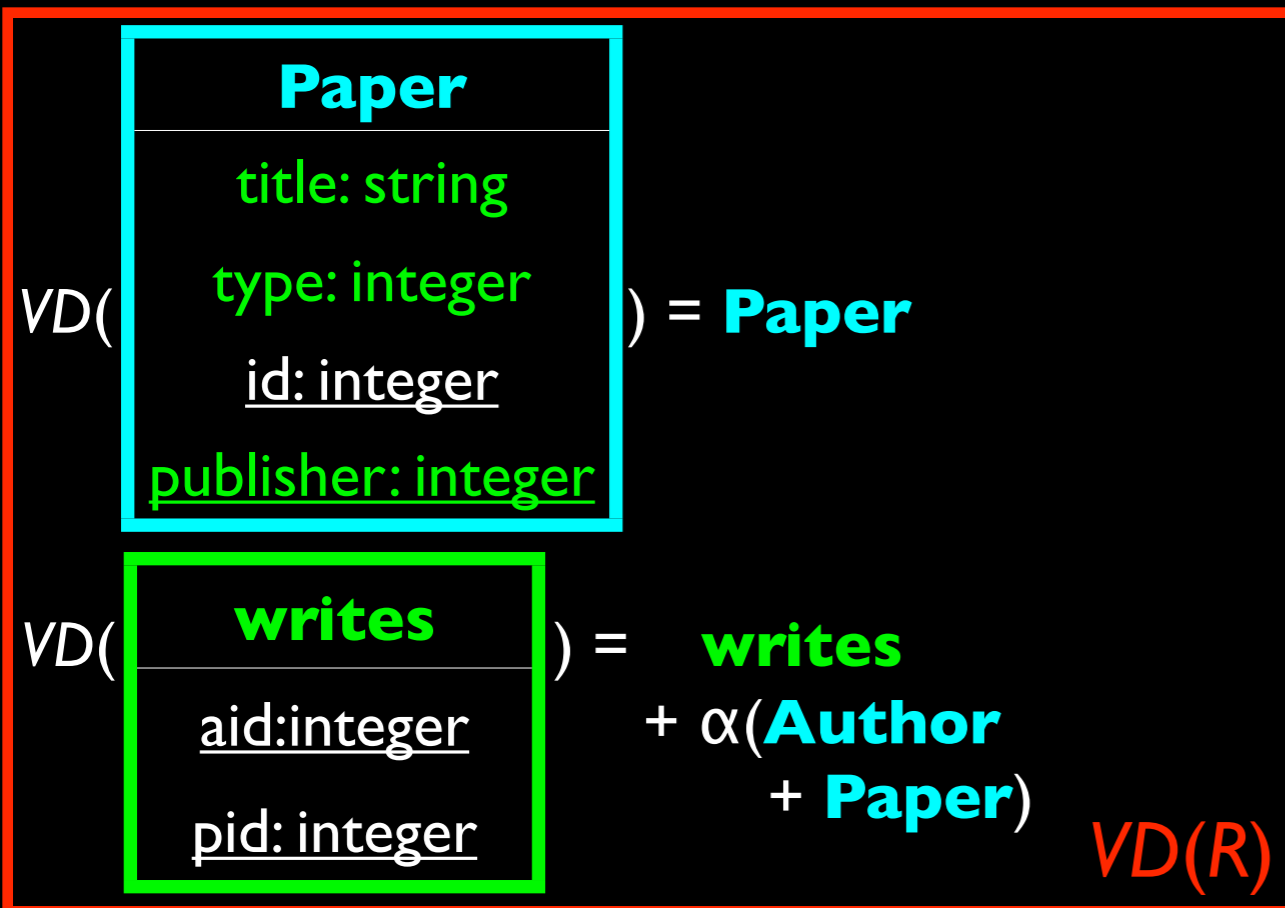
$$VD(\text{publisher: integer}) = \text{publisher} + \alpha(\text{Paper} + \text{Publisher})$$

$$VD(\text{title: string}) = \text{title} + \alpha \cdot \text{Paper} + \beta \cdot \text{string}$$

Simple mappings are “discovered” by building “virtual documents” (statistical representations), or VDs, of entities in the database and ontology, and then the confidence of a mapping between two entities is the cosine of these statistical representations. How these are defined is different for the various types of entities, but once the VDs are built, computing confidence is a simple cosine operation.

In the paper, the description of an entities is simply the name of the entity. The authors do not specify, but presumably the descriptions in real applications might use extra annotations, like `rdfs:label`, or `rdfs:comment`.

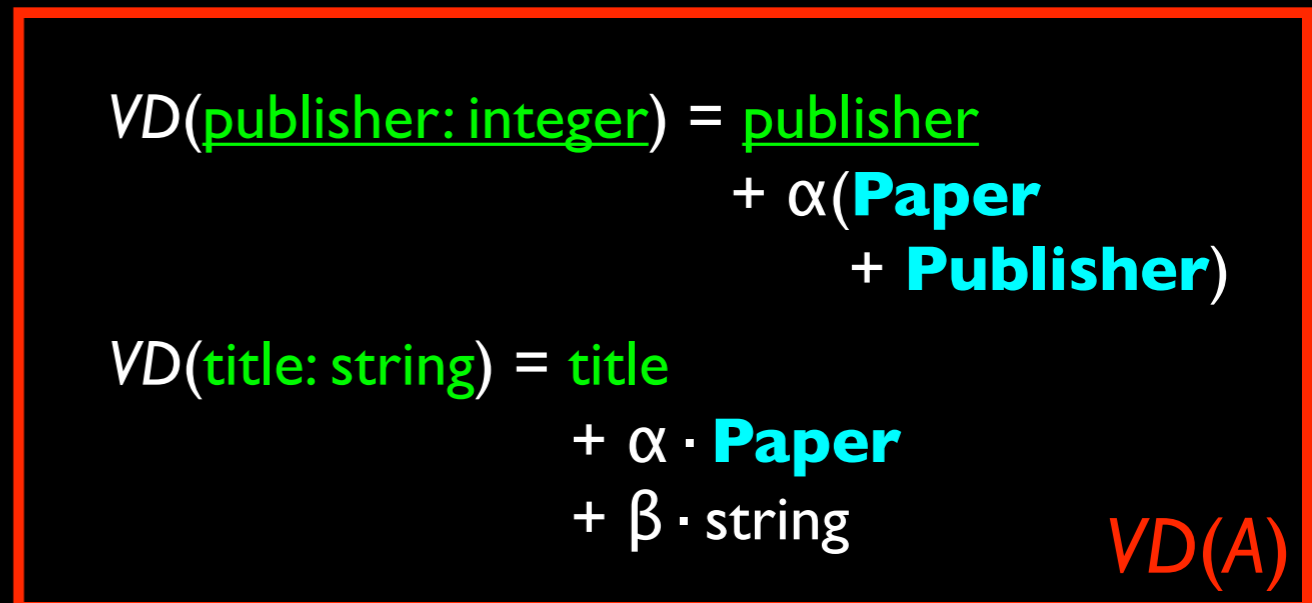
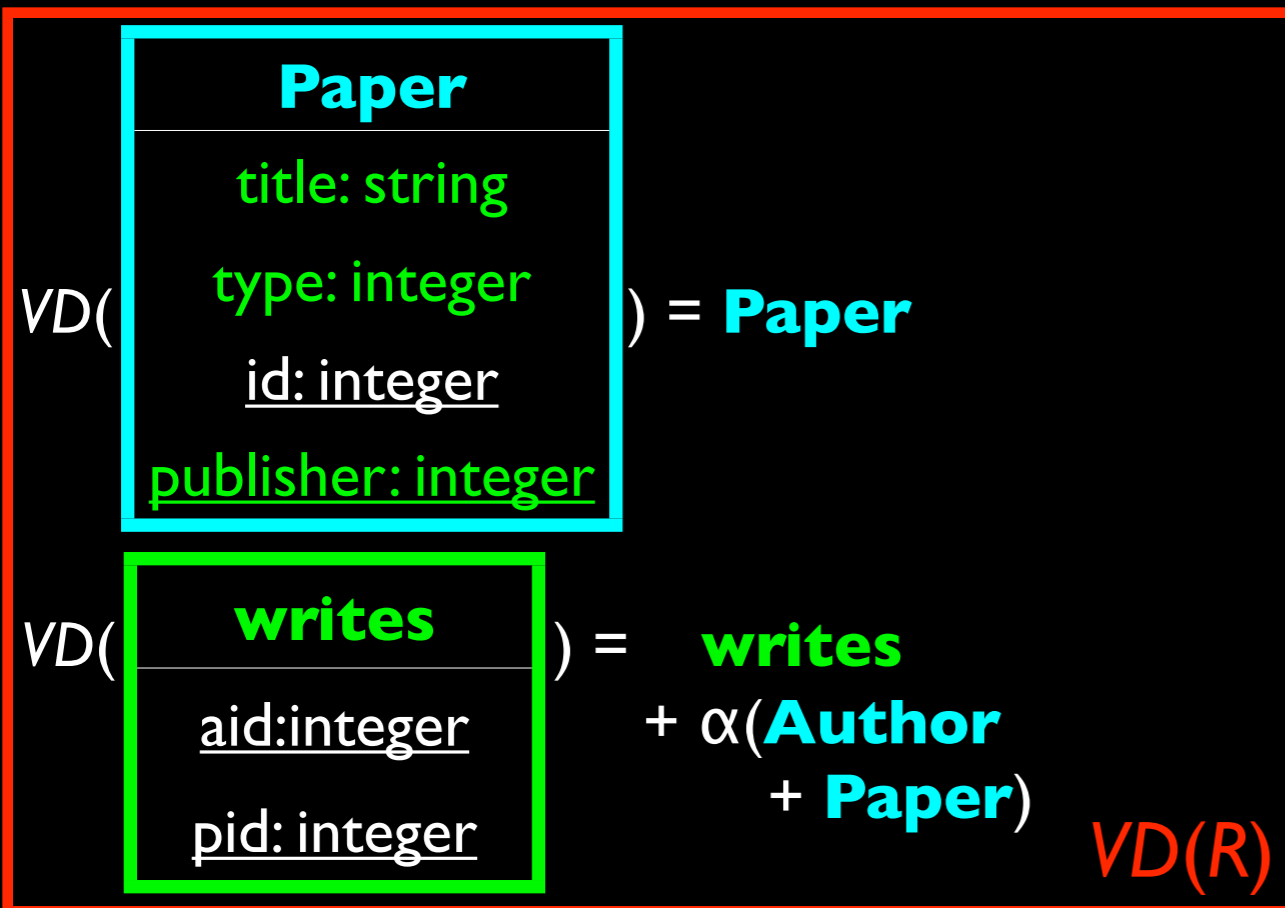
Discovering Simple Mappings



Simple mappings are “discovered” by building “virtual documents” (statistical representations), or VDs, of entities in the database and ontology, and then the confidence of a mapping between two entities is the cosine of these statistical representations. How these are defined is different for the various types of entities, but once the VDs are built, computing confidence is a simple cosine operation.

In the paper, the description of an entities is simply the name of the entity. The authors do not specify, but presumably the descriptions in real applications might use extra annotations, like rdfs:label, or rdfs:comment.

Discovering Simple Mappings



$VD(\text{foaf:Person}) = \text{foaf:Person}$

Simple mappings are “discovered” by building “virtual documents” (statistical representations), or VDs, of entities in the database and ontology, and then the confidence of a mapping between two entities is the cosine of these statistical representations. How these are defined is different for the various types of entities, but once the VDs are built, computing confidence is a simple cosine operation.

In the paper, the description of an entities is simply the name of the entity. The authors do not specify, but presumably the descriptions in real applications might use extra annotations, like `rdfs:label`, or `rdfs:comment`.

Discovering Simple Mappings

$$\text{VD}\left(\begin{array}{l} \mathbf{Paper} \\ \text{title: string} \\ \text{type: integer} \\ \text{id: integer} \\ \text{publisher: integer} \end{array}\right) = \mathbf{Paper}$$
$$\text{VD}\left(\begin{array}{l} \mathbf{writes} \\ \text{aid: integer} \\ \text{pid: integer} \end{array}\right) = \mathbf{writes} \\ + \alpha(\mathbf{Author} \\ + \mathbf{Paper}) \quad \text{VD}(R)$$

$$\text{VD}(\text{publisher: integer}) = \text{publisher} \\ + \alpha(\mathbf{Paper} \\ + \mathbf{Publisher})$$
$$\text{VD}(\text{title: string}) = \text{title} \\ + \alpha \cdot \mathbf{Paper} \\ + \beta \cdot \text{string} \quad \text{VD}(A)$$

$$\text{VD}(\text{foaf:Person}) = \text{foaf:Person}$$

VD(C)

Simple mappings are “discovered” by building “virtual documents” (statistical representations), or VDs, of entities in the database and ontology, and then the confidence of a mapping between two entities is the cosine of these statistical representations. How these are defined is different for the various types of entities, but once the VDs are built, computing confidence is a simple cosine operation.

In the paper, the description of an entities is simply the name of the entity. The authors do not specify, but presumably the descriptions in real applications might use extra annotations, like rdfs:label, or rdfs:comment.

Discovering Simple Mappings

$$\text{VD}\left(\begin{array}{l} \mathbf{Paper} \\ \text{title: string} \\ \text{type: integer} \\ \text{id: integer} \\ \text{publisher: integer} \end{array}\right) = \mathbf{Paper}$$
$$\text{VD}\left(\begin{array}{l} \mathbf{writes} \\ \text{aid: integer} \\ \text{pid: integer} \end{array}\right) = \mathbf{writes} + \alpha(\mathbf{Author} + \mathbf{Paper})$$

VD(R)

$$\text{VD}(\text{publisher: integer}) = \text{publisher} + \alpha(\mathbf{Paper} + \mathbf{Publisher})$$
$$\text{VD}(\text{title: string}) = \text{title} + \alpha \cdot \mathbf{Paper} + \beta \cdot \text{string}$$

VD(A)

$$\text{VD}(\text{foaf:member}) = \text{foaf:member} + \alpha(\text{foaf:Group} + \text{foaf:Agent})$$

$$\text{VD}(\text{foaf:Person}) = \text{foaf:Person}$$

VD(C)

Simple mappings are “discovered” by building “virtual documents” (statistical representations), or VDs, of entities in the database and ontology, and then the confidence of a mapping between two entities is the cosine of these statistical representations. How these are defined is different for the various types of entities, but once the VDs are built, computing confidence is a simple cosine operation.

In the paper, the description of an entities is simply the name of the entity. The authors do not specify, but presumably the descriptions in real applications might use extra annotations, like `rdfs:label`, or `rdfs:comment`.

Discovering Simple Mappings

$$\text{VD}\left(\begin{array}{l} \mathbf{Paper} \\ \text{title: string} \\ \text{type: integer} \\ \text{id: integer} \\ \text{publisher: integer} \end{array}\right) = \mathbf{Paper}$$
$$\text{VD}\left(\begin{array}{l} \mathbf{writes} \\ \text{aid: integer} \\ \text{pid: integer} \end{array}\right) = \mathbf{writes} + \alpha(\mathbf{Author} + \mathbf{Paper})$$

VD(R)

$$\text{VD}(\text{foaf:Person}) = \text{foaf:Person}$$

VD(C)

$$\text{VD}(\text{publisher: integer}) = \text{publisher} + \alpha(\mathbf{Paper} + \mathbf{Publisher})$$

$$\text{VD}(\text{title: string}) = \text{title} + \alpha \cdot \mathbf{Paper} + \beta \cdot \text{string}$$

VD(A)

$$\text{VD}(\text{foaf:member}) = \text{foaf:member} + \alpha(\text{foaf:Group} + \text{foaf:Agent})$$

$$\text{VD}(\text{foaf:firstName}) = \text{foaf:firstName} + \alpha \cdot \text{foaf:Person} + \beta \cdot \text{rdf:Literal}$$

Simple mappings are “discovered” by building “virtual documents” (statistical representations), or VDs, of entities in the database and ontology, and then the confidence of a mapping between two entities is the cosine of these statistical representations. How these are defined is different for the various types of entities, but once the VDs are built, computing confidence is a simple cosine operation.

In the paper, the description of an entities is simply the name of the entity. The authors do not specify, but presumably the descriptions in real applications might use extra annotations, like `rdfs:label`, or `rdfs:comment`.

Discovering Simple Mappings

$$\text{VD}\left(\begin{array}{l} \mathbf{Paper} \\ \text{title: string} \\ \text{type: integer} \\ \text{id: integer} \\ \text{publisher: integer} \end{array}\right) = \mathbf{Paper}$$
$$\text{VD}\left(\begin{array}{l} \mathbf{writes} \\ \text{aid: integer} \\ \text{pid: integer} \end{array}\right) = \mathbf{writes} + \alpha(\mathbf{Author} + \mathbf{Paper})$$

VD(R)

$$\text{VD}(\text{publisher: integer}) = \text{publisher} + \alpha(\mathbf{Paper} + \mathbf{Publisher})$$
$$\text{VD}(\text{title: string}) = \text{title} + \alpha \cdot \mathbf{Paper} + \beta \cdot \text{string}$$

VD(A)

$$\text{VD}(\text{foaf:member}) = \text{foaf:member} + \alpha(\text{foaf:Group} + \text{foaf:Agent})$$

$$\text{VD}(\text{foaf:firstName}) = \text{foaf:firstName} + \alpha \cdot \text{foaf:Person} + \beta \cdot \text{rdf:Literal}$$

VD(P)

$$\text{VD}(\text{foaf:Person}) = \text{foaf:Person}$$

VD(C)

Simple mappings are “discovered” by building “virtual documents” (statistical representations), or VDs, of entities in the database and ontology, and then the confidence of a mapping between two entities is the cosine of these statistical representations. How these are defined is different for the various types of entities, but once the VDs are built, computing confidence is a simple cosine operation.

In the paper, the description of an entities is simply the name of the entity. The authors do not specify, but presumably the descriptions in real applications might use extra annotations, like rdfs:label, or rdfs:comment.

Discovering Simple Mappings

$$\text{VD}\left(\begin{array}{l} \mathbf{Paper} \\ \text{title: string} \\ \text{type: integer} \\ \text{id: integer} \\ \text{publisher: integer} \end{array}\right) = \mathbf{Paper}$$
$$\text{VD}\left(\begin{array}{l} \mathbf{writes} \\ \text{aid: integer} \\ \text{pid: integer} \end{array}\right) = \mathbf{writes} + \alpha(\mathbf{Author} + \mathbf{Paper})$$

VD(R)

$$\text{VD}(\text{publisher: integer}) = \text{publisher} + \alpha(\mathbf{Paper} + \mathbf{Publisher})$$
$$\text{VD}(\text{title: string}) = \text{title} + \alpha \cdot \mathbf{Paper} + \beta \cdot \text{string}$$

VD(A)

$$\text{VD}(\text{foaf:member}) = \text{foaf:member} + \alpha(\text{foaf:Group} + \text{foaf:Agent})$$

$$\text{VD}(\text{foaf:firstName}) = \text{foaf:firstName} + \alpha \cdot \text{foaf:Person} + \beta \cdot \text{rdf:Literal}$$

VD(P)

$$\text{VD}(\text{foaf:Person}) = \text{foaf:Person}$$

VD(C)

$$\text{confidence}(\text{VD}_i, \text{VD}_j) = \cos(N_i, N_j)$$

6

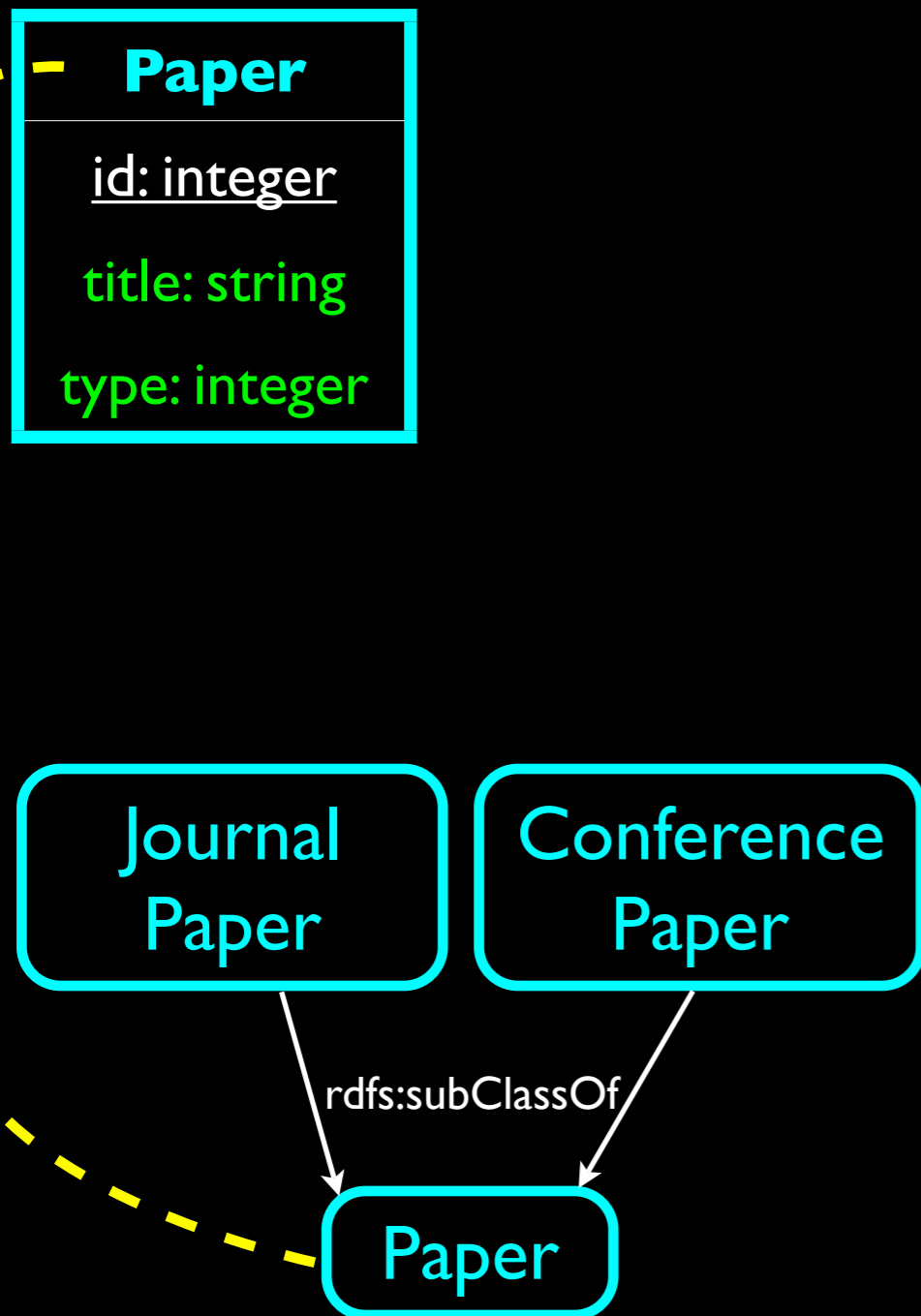
Simple mappings are “discovered” by building “virtual documents” (statistical representations), or VDs, of entities in the database and ontology, and then the confidence of a mapping between two entities is the cosine of these statistical representations. How these are defined is different for the various types of entities, but once the VDs are built, computing confidence is a simple cosine operation.

In the paper, the description of an entities is simply the name of the entity. The authors do not specify, but presumably the descriptions in real applications might use extra annotations, like rdfs:label, or rdfs:comment.

Validating Consistency

- Assume that mappings between ontology classes and entity relations are correct.
- Check whether proposed property mappings satisfy constraints (e.g., `rdfs:domain`, `rdfs:range`).

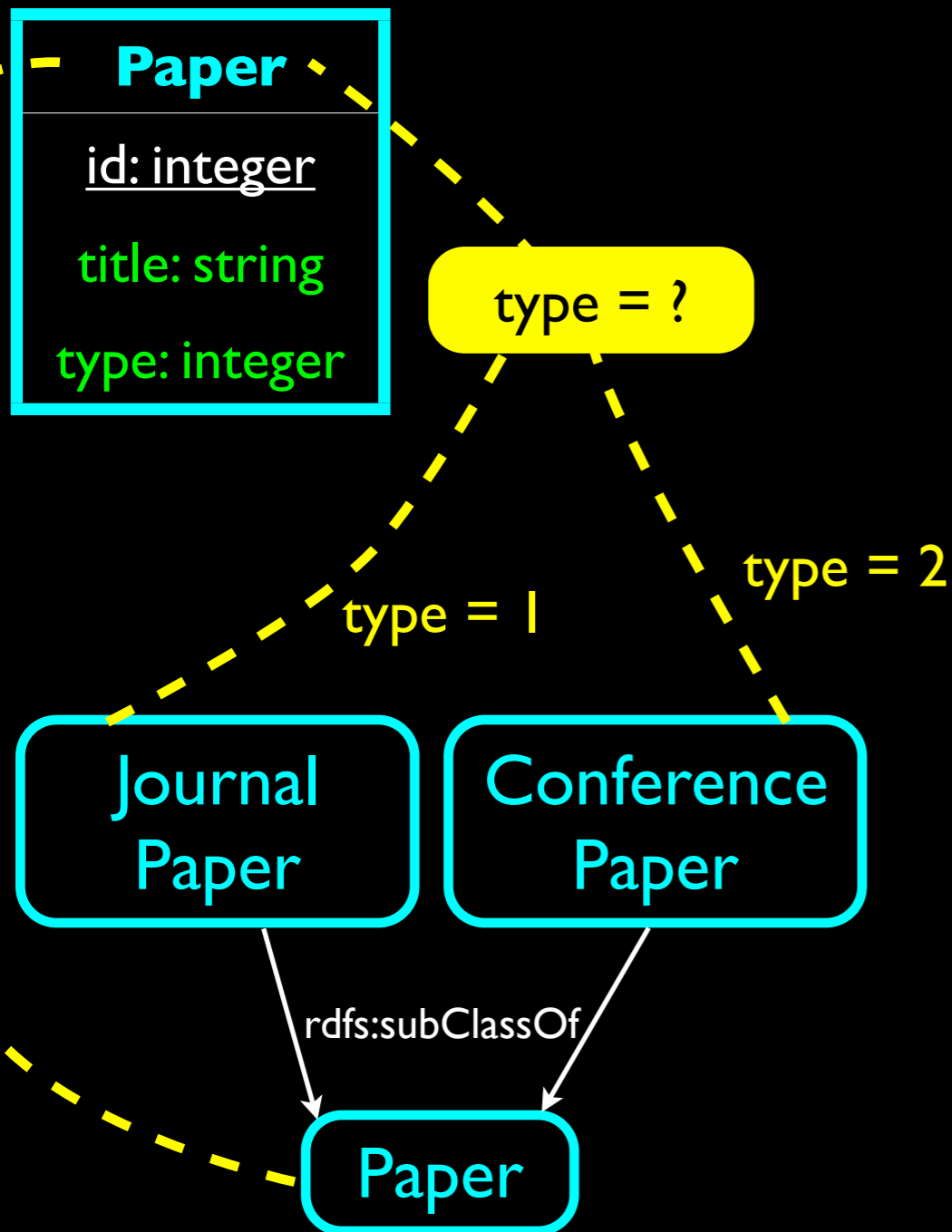
Constructing Contextual Mappings



8

Contextual mappings are mappings in which attributes are used to map individuals in an entity table to classes in the ontology that are **subclasses** of the class to which the entire entity relation corresponds. Of course, some such relations could be extracted from `rdfs:subClassOf` properties in the ontology, but here we are looking for those subclasses whose elements can be determined by particular attribute **values**. To be able to do this, of course, requires instance data. Fortunately, databases are full of instance data, and it turns out that most ontologies have some too.

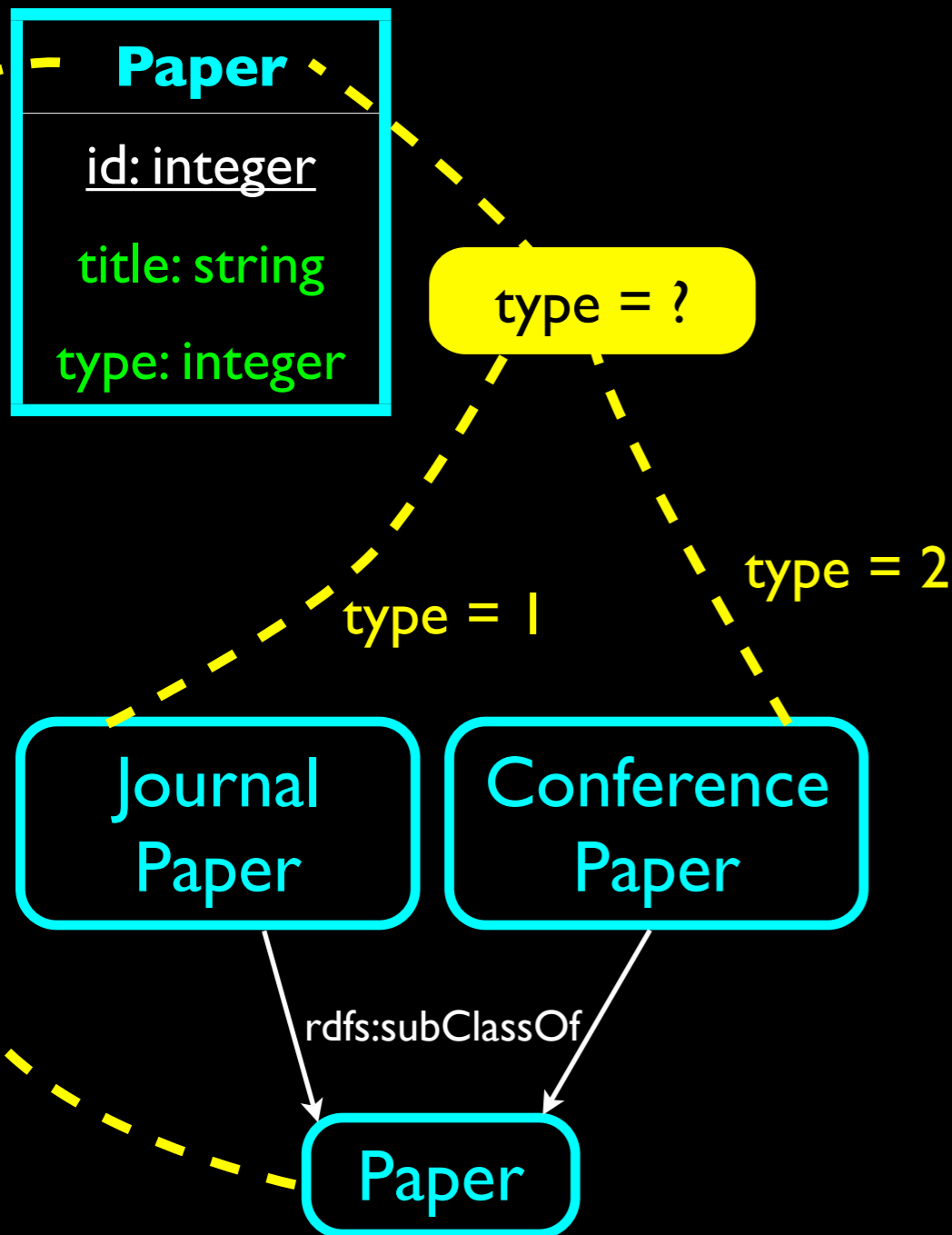
Constructing Contextual Mappings



8

Contextual mappings are mappings in which attributes are used to map individuals in an entity table to classes in the ontology that are **subclasses** of the class to which the entire entity relation corresponds. Of course, some such relations could be extracted from `rdfs:subClassOf` properties in the ontology, but here we are looking for those subclasses whose elements can be determined by particular attribute **values**. To be able to do this, of course, requires instance data. Fortunately, databases are full of instance data, and it turns out that most ontologies have some too.

Constructing Contextual Mappings



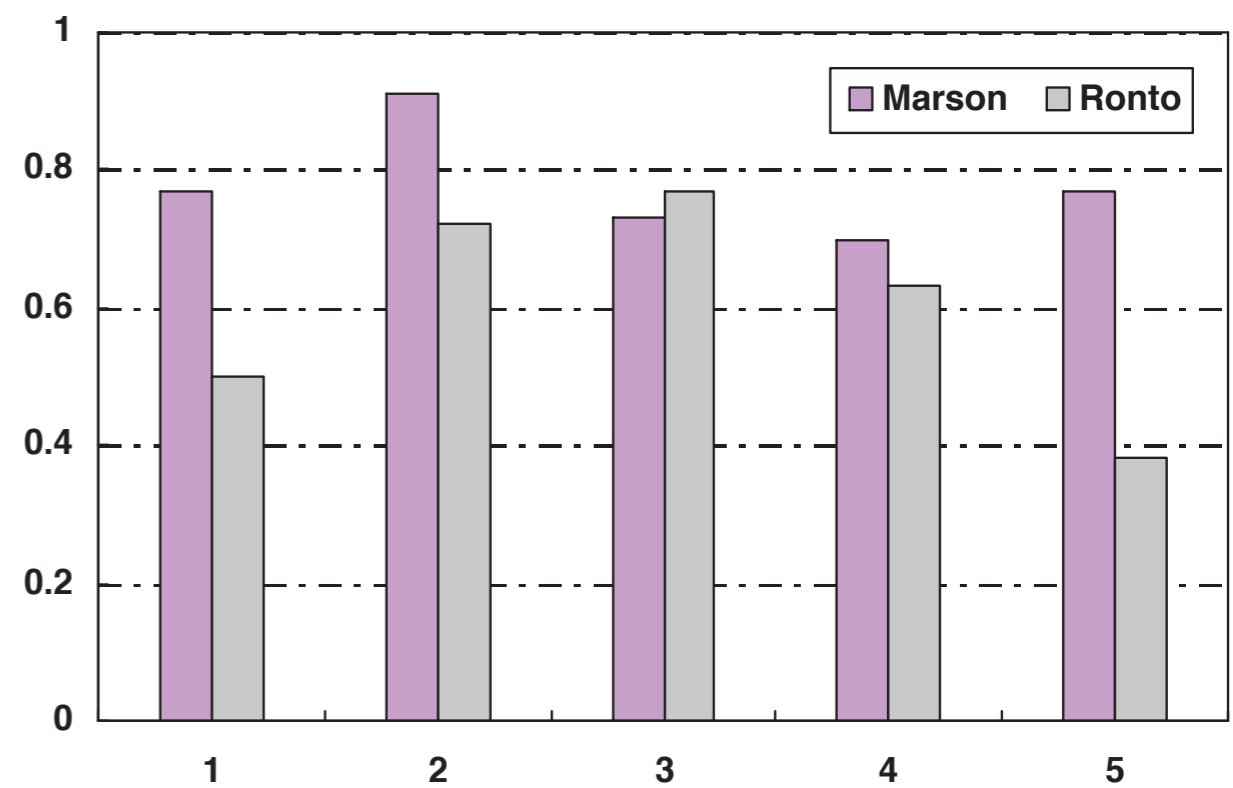
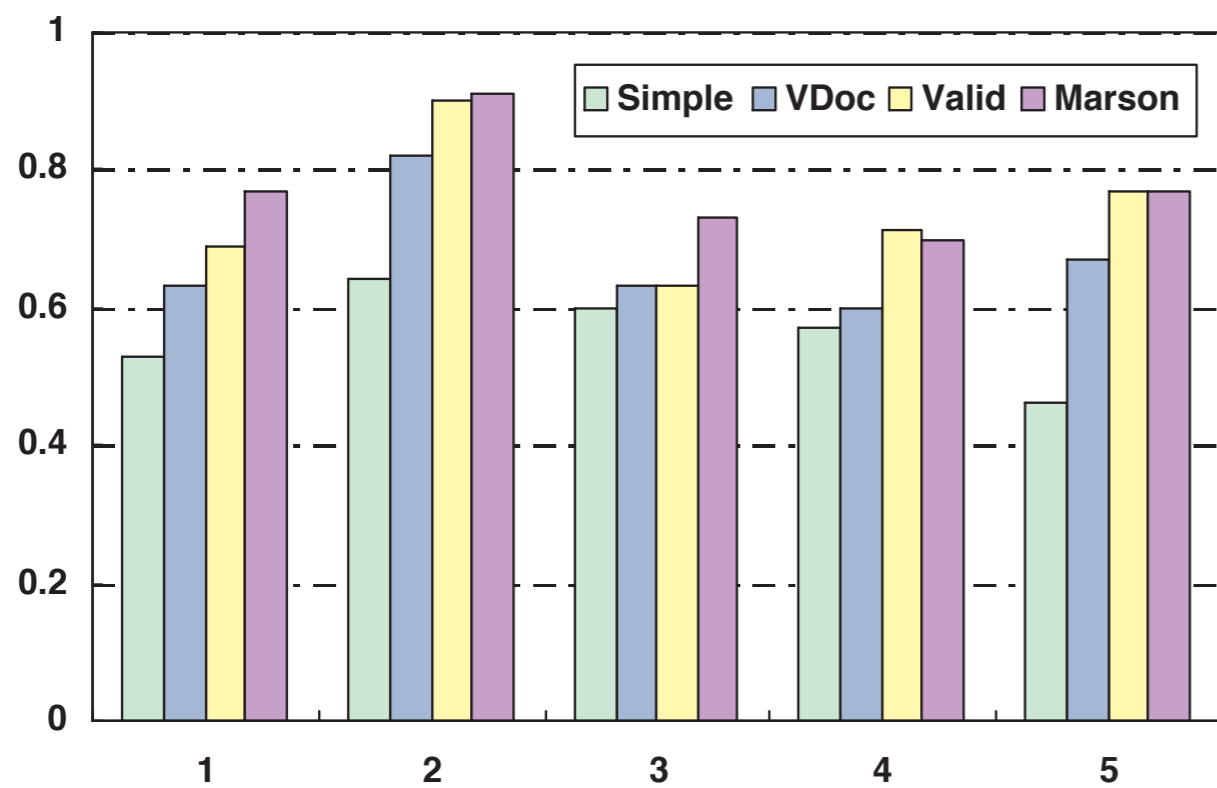
For each relation u to class v mapping:
For each disjoint subclass of v :
Compute the databases instances corresponding to the subclass.
Find a maximizing information gain.
When $\text{gain} > \text{threshold}$
add contextual mappings for a and the subclasses of v .

Contextual mappings are mappings in which attributes are used to map individuals in an entity table to classes in the ontology that are **subclasses** of the class to which the entire entity relation corresponds. Of course, some such relations could be extracted from `rdfs:subClassOf` properties in the ontology, but here we are looking for those subclasses whose elements can be determined by particular attribute **values**. To be able to do this, of course, requires instance data. Fortunately, databases are full of instance data, and it turns out that most ontologies have some too.

Evaluation

- *Simple*—local descriptions only, $\alpha=0$, $\beta=0$, no consistency checking
- *VDoc*—virtual documents, no consistency checking, $\alpha=0.2$, $\beta=0.1$
- *Valid*—*Simple* with consistency checking
- *Marson*—All four phases, $\alpha=0.2$, $\beta=0.1$
- *Ronto*—Papapanagiotou &al. in 2006

ID	R	#R	#A	○	#C	#P	Ref.
1	UTCS	8	32	Univ. CS	53	35	18
2	VLDB	9	38	Conference	18	29	27
3	DBLP	5	27	Bibliography	66	81	21
4	OBSERVER	8	115	Bibliography	66	81	72
5	Country	6	18	Factbook	43	209	22



(a) SIMPLE, VDOC, VALID and MARSON₁₀

(b) MARSON and RONTO

The table on top shows the five databases and ontologies for which mappings were constructed. The Ref column shows the number of mappings generated by trained individuals. There is no breakdown of these between simple mappings between relations and class, simple mappings between attributes and properties, and contextual mappings. Note that the graph on the left compares Simple, VDoc, Valid, and Marson, though all of these are simply subsets of Marson; i.e., they are Marson with various phases turned **off**.