UiO : Department of Informatics
University of Oslo

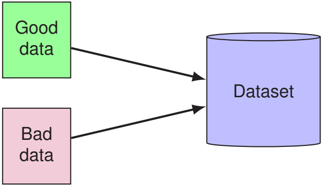
Bounds: Expressing Reservations about Incoming Data

Martin G. Skjæveland   Audun Stolpe

Presented by Kjetil Kjernsmo

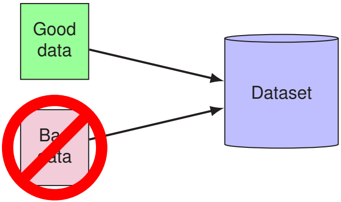
COLD 2013
22 October 2013

Synopsis



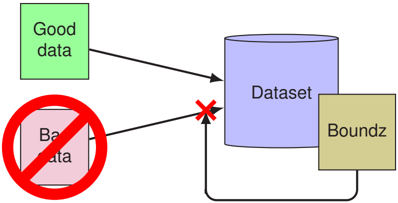
- Protect a dataset from unwanted amendments

Synopsis



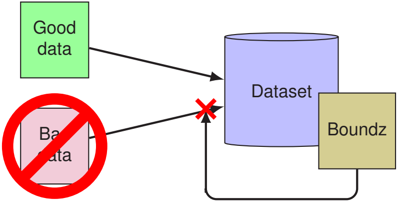
- Protect a dataset from unwanted amendments
- Don't mess (literally) with my dataset, please.

Synopsis



- Protect a dataset from unwanted amendments
- Don't mess (literally) with my dataset, please.
- *Boundz*: Vocabulary for expressing reservations about incoming data

Synopsis



- Protect a dataset from unwanted amendments
- Don't mess (literally) with my dataset, please.
- *Boundz*: Vocabulary for expressing reservations about incoming data
- Theory: Based on *bounded homomorphisms*

Motivation

When do you want to protect a dataset from unwanted amendments?

Motivation

When do you want to protect a dataset from unwanted amendments?

- Suppose you have a triple store, and

Motivation

When do you want to protect a dataset from unwanted amendments?

- Suppose you have a triple store, and
- you'd like it to be community-curated

Motivation

When do you want to protect a dataset from unwanted amendments?

- Suppose you have a triple store, and
- you'd like it to be community-curated
- i.e., you'd like to invite people and institutions to contribute data

Motivation

When do you want to protect a dataset from unwanted amendments?

- Suppose you have a triple store, and
- you'd like it to be community-curated
- i.e., you'd like to invite people and institutions to contribute data
- but you'd like to protect certain elements of structure

What structure?

What structure would you want to protect?

What structure?

What structure would you want to protect?

It could be many things, e.g.,

What structure?

What structure would you want to protect?

It could be many things, e.g.,

- the ontology axioms in your dataset

What structure?

What structure would you want to protect?

It could be many things, e.g.,

- the ontology axioms in your dataset
  - you decide the ontology, they contribute the data

What structure?

What structure would you want to protect?

It could be many things, e.g.,

- the ontology axioms in your dataset
  - you decide the ontology, they contribute the data
- a particular vocabulary or ontology

What structure?

What structure would you want to protect?

It could be many things, e.g.,

- the ontology axioms in your dataset
  - you decide the ontology, they contribute the data
- a particular vocabulary or ontology
  - the core ontology or internal bookkeeping vocabulary should be stable

What structure?

What structure would you want to protect?

It could be many things, e.g.,

- the ontology axioms in your dataset
  - you decide the ontology, they contribute the data
- a particular vocabulary or ontology
  - the core ontology or internal bookkeeping vocabulary should be stable
- a set of facts

What structure?

What structure would you want to protect?

It could be many things, e.g.,

- the ontology axioms in your dataset
  - you decide the ontology, they contribute the data
- a particular vocabulary or ontology
  - the core ontology or internal bookkeeping vocabulary should be stable
- a set of facts
  - metadata about the dataset is your business only

What structure?

What structure would you want to protect?

It could be many things, e.g.,

- the ontology axioms in your dataset
  - you decide the ontology, they contribute the data
- a particular vocabulary or ontology
  - the core ontology or internal bookkeeping vocabulary should be stable
- a set of facts
  - metadata about the dataset is your business only
- whole parts of the dataset or data in a particular namespace

What structure?

What structure would you want to protect?

It could be many things, e.g.,

- the ontology axioms in your dataset
  - you decide the ontology, they contribute the data
- a particular vocabulary or ontology
  - the core ontology or internal bookkeeping vocabulary should be stable
- a set of facts
  - metadata about the dataset is your business only
- whole parts of the dataset or data in a particular namespace
  - don't touch the finished/perfect/imported parts

Examples

- Ontology hijacking:

Examples

- Ontology hijacking:
    - say your ontology uses the `dcterms:subject` property

Examples

- Ontology hijacking:
  - say your ontology uses the `dcterms:subject` property
  - then incoming data says

```
dcterms:subject  rdfs:subClassOf  ex:topic
```

Examples

- Ontology hijacking:
  - say your ontology uses the `dcterms:subject` property
  - then incoming data says

    ```
    dcterms:subject  rdfs:subClassOf  ex:topic
    ```
  - as a result the number of statements that are inferred will increase

Examples

- Ontology hijacking:
  - say your ontology uses the `dcterms:subject` property
  - then incoming data says

    ```
    dcterms:subject  rdfs:subClassOf  ex:topic
    ```

  - as a result the number of statements that are inferred will increase
  - entailments over unrelated data will be affected

Examples

- Ontology hijacking:
  - say your ontology uses the `dcterms:subject` property
  - then incoming data says

    `dcterms:subject  rdfs:subClassOf  ex:topic`

  - as a result the number of statements that are inferred will increase
  - entailments over unrelated data will be affected
- Breach of contract:

Examples

- Ontology hijacking:
  - say your ontology uses the `dcterms:subject` property
  - then incoming data says

    `dcterms:subject  rdfs:subClassOf  ex:topic`
  - as a result the number of statements that are inferred will increase
  - entailments over unrelated data will be affected
- Breach of contract:
  - say your triple store advertises a particular vocabulary, i.e., API,

Examples

- Ontology hijacking:
  - say your ontology uses the `dcterms:subject` property
  - then incoming data says

    ```
    dcterms:subject  rdfs:subClassOf  ex:topic
    ```

  - as a result the number of statements that are inferred will increase
  - entailments over unrelated data will be affected

- Breach of contract:
  - say your triple store advertises a particular vocabulary, i.e., API,
  - used to drive faceted browsing or search

Examples

- Ontology hijacking:
  - say your ontology uses the `dcterms:subject` property
  - then incoming data says

    ```
    dcterms:subject  rdfs:subClassOf  ex:topic
    ```
  - as a result the number of statements that are inferred will increase
  - entailments over unrelated data will be affected

- Breach of contract:
  - say your triple store advertises a particular vocabulary, i.e., API,
  - used to drive faceted browsing or search
  - then incoming data interferes with it

Examples

- Ontology hijacking:
  - say your ontology uses the `dcterms:subject` property
  - then incoming data says

    `dcterms:subject  rdfs:subClassOf  ex:topic`
  - as a result the number of statements that are inferred will increase
  - entailments over unrelated data will be affected
- Breach of contract:
  - say your triple store advertises a particular vocabulary, i.e., API,
  - used to drive faceted browsing or search
  - then incoming data interferes with it
  - which affects the uses of the API

Examples cont.

- Skewing content:

Examples cont.

- Skewing content:
  - say your triple store contains election results

Examples cont.

- Skewing content:
  - say your triple store contains election results
  - à la `data.gov.uk` and `openelectiondata.org`

Examples cont.

- Skewing content:
  - say your triple store contains election results
  - à la `data.gov.uk` and `openelectiondata.org`
  - then incoming data makes erroneous claims about existing elections

Examples cont.

- Skewing content:
  - say your triple store contains election results
  - à la `data.gov.uk` and `openelectiondata.org`
  - then incoming data makes erroneous claims about existing elections
  - resulting in noisy data

Examples cont.

- Skewing content:
  - say your triple store contains election results
  - à la `data.gov.uk` and `openelectiondata.org`
  - then incoming data makes erroneous claims about existing elections
  - resulting in noisy data
  - lowering overall data quality and re-usability

Examples cont.

- Skewing content:
  - say your triple store contains election results
  - à la `data.gov.uk` and `openelectiondata.org`
  - then incoming data makes erroneous claims about existing elections
  - resulting in noisy data
  - lowering overall data quality and re-usability
  - breaks trust

What is Boundz?

- A vocabulary for expressing reservations about incoming data

What is Boundz?

- A vocabulary for expressing reservations about incoming data
- not what shape incoming data should have

What is Boundz?

- A vocabulary for expressing reservations about incoming data
- not what shape incoming data should have
- but what shape it *should not have*

What is Boundz?

- A vocabulary for expressing reservations about incoming data
- not what shape incoming data should have
- but what shape it *should not have*
- Vocabulary elements:

What is Boundz?

- A vocabulary for expressing reservations about incoming data
- not what shape incoming data should have
- but what shape it *should not have*
- Vocabulary elements:
  - Fine-grained restrictions on types and predicates with `Bounds`

What is Boundz?

- A vocabulary for expressing reservations about incoming data
- not what shape incoming data should have
- but what shape it *should not have*
- Vocabulary elements:
  - Fine-grained restrictions on types and predicates with `Bounds`
  - `Exceptions`, i.e., what to do in case of a violation

What is Boundz?

- A vocabulary for expressing reservations about incoming data
- not what shape incoming data should have
- but what shape it *should not have*
- Vocabulary elements:
  - Fine-grained restrictions on types and predicates with `Bounds`
  - `Exceptions`, i.e., what to do in case of a violation
    - abort

What is Boundz?

- A vocabulary for expressing reservations about incoming data
- not what shape incoming data should have
- but what shape it *should not have*
- Vocabulary elements:
  - Fine-grained restrictions on types and predicates with `Bounds`
  - `Exceptions`, i.e., what to do in case of a violation
    - abort
    - accept parts of the incoming data

What is Boundz?

- A vocabulary for expressing reservations about incoming data
- not what shape incoming data should have
- but what shape it *should not have*
- Vocabulary elements:
  - Fine-grained restrictions on types and predicates with `Bounds`
  - `Exceptions`, i.e., what to do in case of a violation
    - abort
    - accept parts of the incoming data
    - ignore only violating triples

What is Boundz?

- A vocabulary for expressing reservations about incoming data
- not what shape incoming data should have
- but what shape it *should not have*
- Vocabulary elements:
  - Fine-grained restrictions on types and predicates with `Bounds`
  - `Exceptions`, i.e., what to do in case of a violation
    - abort
    - accept parts of the incoming data
    - ignore only violating triples
  - Also `Data exchanges`, `Payloads`, Validation results

What is Boundz?

- A vocabulary for expressing reservations about incoming data
- not what shape incoming data should have
- but what shape it *should not have*
- Vocabulary elements:
  - Fine-grained restrictions on types and predicates with `Bounds`
  - `Exceptions`, i.e., what to do in case of a violation
    - abort
    - accept parts of the incoming data
    - ignore only violating triples
  - Also `Data exchanges`, `Payloads`, Validation results
    - See `http://sws.ifi.uio.no/vocab/boundz` for details

What is Boundz?

- A vocabulary for expressing reservations about incoming data
- not what shape incoming data should have
- but what shape it *should not have*
- Vocabulary elements:
  - Fine-grained restrictions on types and predicates with `Bounds`
  - `Exceptions`, i.e., what to do in case of a violation
    - abort
    - accept parts of the incoming data
    - ignore only violating triples
  - Also `Data exchanges`, `Payloads`, Validation results
    - See `http://sws.ifi.uio.no/vocab/boundz` for details
  - Publish, re-use and combine bounds

Boundz’ Capabilities

- No validation in the “traditional” sense, e.g.,

Boundz' Capabilities

- No validation in the "traditional" sense, e.g.,
  - x is an integer and $0 \leq x \leq 100$

Boundz' Capabilities

- No validation in the "traditional" sense, e.g.,
  - x is an integer and $0 \leq x \leq 100$
  - an x must have n y's

Boundz' Capabilities

- No validation in the "traditional" sense, e.g.,
  - x is an integer and $0 \leq x \leq 100$
  - an x must have n y's
  - $x \leq y$

Boundz' Capabilities

- No validation in the "traditional" sense, e.g.,
  - x is an integer and $0 \leq x \leq 100$
  - an x must have n y's
  - $x \leq y$
- but validation of data relative to receiving dataset, e.g.,

Boundz' Capabilities

- No validation in the "traditional" sense, e.g.,
  - x is an integer and $0 \leq x \leq 100$
  - an x must have n y's
  - $x \leq y$
- but validation of data relative to receiving dataset, e.g.,
  - Do not add more superclasses to my dataset

Boundz' Capabilities

- No validation in the "traditional" sense, e.g.,
  - x is an integer and $0 \leq x \leq 100$
  - an x must have n y's
  - $x \leq y$
- but validation of data relative to receiving dataset, e.g.,
  - Do not add more superclasses to my dataset
  - New `foaf:knows` relationships must relate new persons only

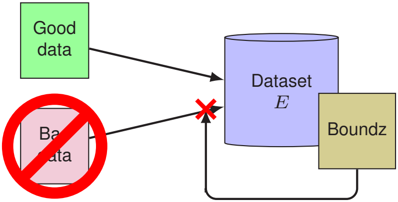
Boundz' Capabilities

- No validation in the "traditional" sense, e.g.,
  - x is an integer and $0 \leq x \leq 100$
  - an x must have n y's
  - $x \leq y$
- but validation of data relative to receiving dataset, e.g.,
  - Do not add more superclasses to my dataset
  - New `foaf:knows` relationships must relate new persons only
  - Adding new data must not rearrange existing data

Boundz' Capabilities

- No validation in the "traditional" sense, e.g.,
  - x is an integer and $0 \leq x \leq 100$
  - an x must have n y's
  - $x \leq y$
- but validation of data relative to receiving dataset, e.g.,
  - Do not add more superclasses to my dataset
  - New `foaf:knows` relationships must relate new persons only
  - Adding new data must not rearrange existing data
  - Write-protecting (parts of) a dataset

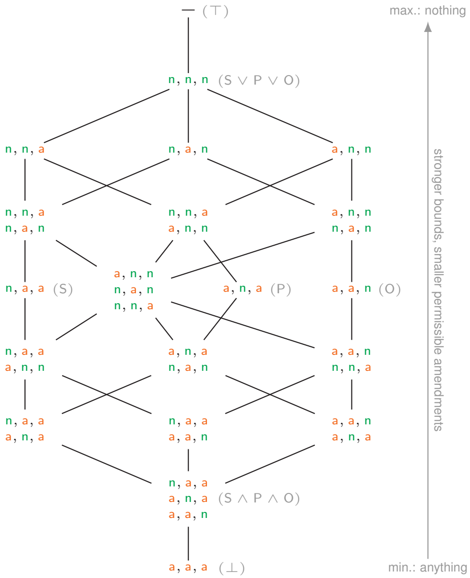
Bounds



Bounds control what new triples can be added to a dataset E based on the elements in E.

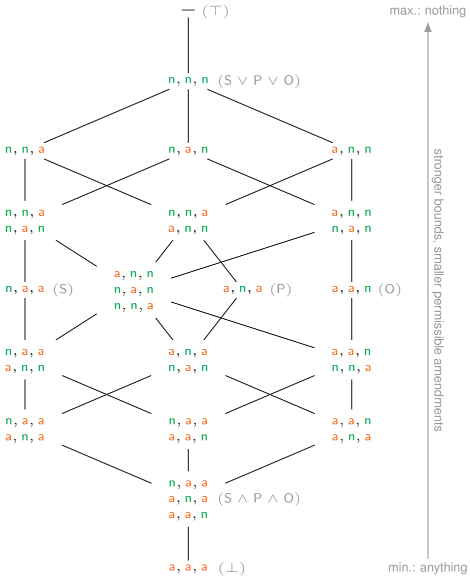
- Can express this with patterns:
  - If an incoming triple does not exist in E,
  - and matches the pattern, then add it.

- *Bounded homomorphism*



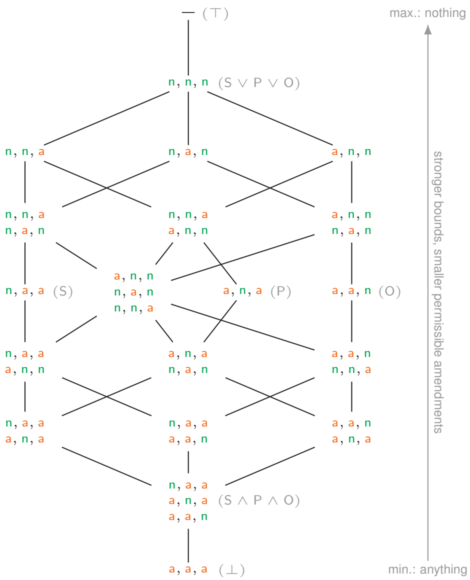
Theoretical background

- *Bounded homomorphism*
- between incoming and existing data



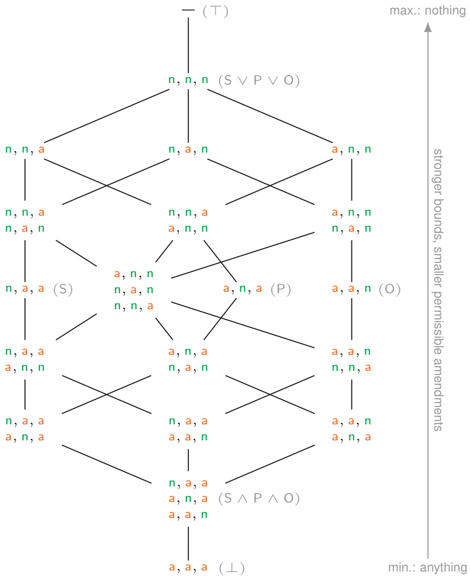
Theoretical background

- *Bounded homomorphism*
- between incoming and existing data
- 20 generic bounds in total



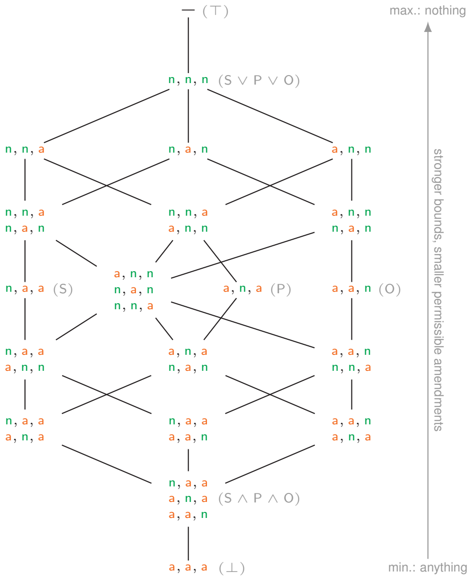
Theoretical background

- *Bounded homomorphism*
- between incoming and existing data
- 20 generic bounds in total
- arranged in lattice according to strength



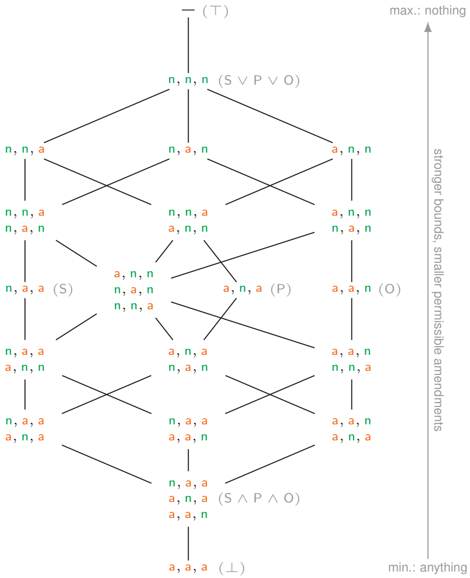
- *Bounded homomorphism*
- between incoming and existing data

- 20 generic bounds in total
- arranged in lattice according to strength
- Pattern legend:



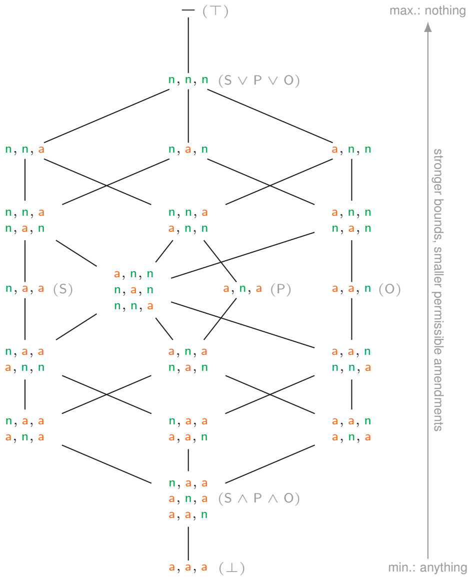
Theoretical background

- *Bounded homomorphism*
- between incoming and existing data
- 20 generic bounds in total
- arranged in lattice according to strength
- Pattern legend:
    - any element



Theoretical background

- *Bounded homomorphism*
- between incoming and existing data
- 20 generic bounds in total
- arranged in lattice according to strength
- Pattern legend:
  - any element
  - new, element must be new: n $\notin E$



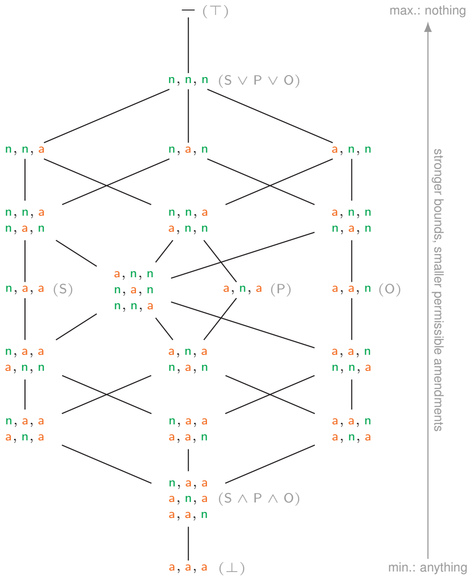
UiO : University of Oslo

Theoretical background

- *Bounded homomorphism*
- between incoming and existing data

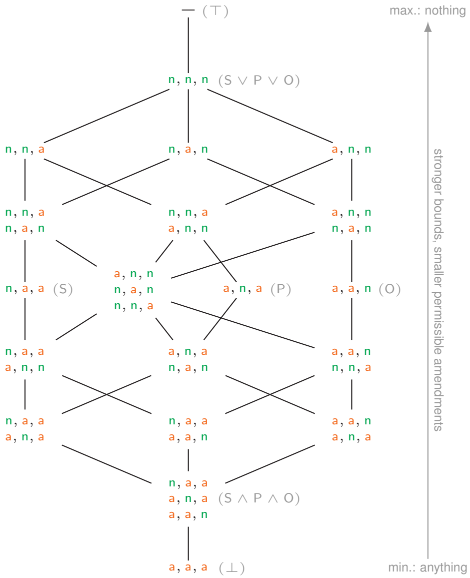
- 20 generic bounds in total
- arranged in lattice according to strength
- Pattern legend:
  - any element
  - new, element must be new: n $\notin E$

- Controls what new triples can be added to a dataset



Theoretical background

- *Bounded homomorphism*
- between incoming and existing data
- 20 generic bounds in total
- arranged in lattice according to strength
- Pattern legend:
  - any element
  - new, element must be new: n $\notin E$
- Controls what new triples can be added to a dataset
- For details, see paper.



Example

- Do not add superclasses (but new subclasses allowed)

Example

- Do not add superclasses (but new subclasses allowed)
  - Bound: $\langle n, \texttt{rdfs:subClassOf}, a \rangle$

Example

- Do not add superclasses (but new subclasses allowed)
  - Bound: $\langle \mathsf{n}, \texttt{rdfs:subClassOf}, \mathsf{a} \rangle$
  - New subclass axioms must have new subjects

Example

- Do not add superclasses (but new subclasses allowed)
  - Bound: $\langle \mathsf{n}, \texttt{rdfs:subClassOf}, \mathsf{a} \rangle$
  - New subclass axioms must have new subjects
  - i.e., adding new subclasses to existing classes is OK

Example

- Do not add superclasses (but new subclasses allowed)
  - Bound: $\langle n, \texttt{rdfs:subClassOf}, a \rangle$
  - New subclass axioms must have new subjects
  - i.e., adding new subclasses to existing classes is OK
  - but old subjects in subclass axioms are not allowed,

Example

- Do not add superclasses (but new subclasses allowed)
  - Bound: $\langle$n, `rdfs:subClassOf`, a$\rangle$
  - New subclass axioms must have new subjects
  - i.e., adding new subclasses to existing classes is OK
  - but old subjects in subclass axioms are not allowed,
  - i.e., no new *super*classes to existing classes

Example

- Do not add superclasses (but new subclasses allowed)
  - Bound: $\langle n, \texttt{rdfs:subClassOf}, a \rangle$
  - New subclass axioms must have new subjects
  - i.e., adding new subclasses to existing classes is OK
  - but old subjects in subclass axioms are not allowed,
  - i.e., no new *super*classes to existing classes
  - and no new subclass relationships between existing classes.

Example

- Do not add superclasses (but new subclasses allowed)
  - Bound: $\langle$n, rdfs:subClassOf, a$\rangle$
  - New subclass axioms must have new subjects
  - i.e., adding new subclasses to existing classes is OK
  - but old subjects in subclass axioms are not allowed,
  - i.e., no new *super*classes to existing classes
  - and no new subclass relationships between existing classes.
  - Prevents basic ontology-hijacking, protects vocabulary.

Example

- New `foaf:knows` relationships must relate new persons

Example

- New `foaf:knows` relationships must relate new persons
- or, “keep away from my friend graph”

Example

- New `foaf:knows` relationships must relate new persons
- or, “keep away from my friend graph”
  - $\langle$n, `foaf:knows`, n$\rangle$

Example

- New `foaf:knows` relationships must relate new persons
- or, “keep away from my friend graph”
  - $\langle$n, `foaf:knows`, n$\rangle$
  - Both subject and object of `foaf:knows` triples must be new

Example

- New `foaf:knows` relationships must relate new persons
- or, “keep away from my friend graph”
  - $\langle \texttt{n}, \texttt{foaf:knows}, \texttt{n} \rangle$
  - Both subject and object of `foaf:knows` triples must be new
  - Keeping amendments separated from receiver’s data

Example

- Adding new data must not re-arrange existing data

Example

- Adding new data must not re-arrange existing data
  - $\langle n, a, a \rangle$
    $\langle a, n, a \rangle$
    $\langle a, a, n \rangle$

Example

- Adding new data must not re-arrange existing data
  - $\langle n, a, a \rangle$
    $\langle a, n, a \rangle$
    $\langle a, a, n \rangle$
  - Disallows $\langle a, a, a \rangle$

Example

- Adding new data must not re-arrange existing data
  - $\langle n, a, a \rangle$
    $\langle a, n, a \rangle$
    $\langle a, a, n \rangle$
  - Disallows $\langle a, a, a \rangle$
  - i.e., adding new triples with old elements only

Example

- Adding new data must not re-arrange existing data
  - $\langle n, a, a \rangle$
    $\langle a, n, a \rangle$
    $\langle a, a, n \rangle$
  - Disallows $\langle a, a, a \rangle$
  - i.e., adding new triples with old elements only
  - Prevents skewing of existing data and noise

Example

- Adding new data must not re-arrange existing data
  - $\langle n, a, a \rangle$
    $\langle a, n, a \rangle$
    $\langle a, a, n \rangle$
  - Disallows $\langle a, a, a \rangle$
  - i.e., adding new triples with old elements only
  - Prevents skewing of existing data and noise
- Implements “conservative extensions” for RDF graphs

Example

- Adding new data must not re-arrange existing data
  - $\langle n, a, a \rangle$
    $\langle a, n, a \rangle$
    $\langle a, a, n \rangle$
  - Disallows $\langle a, a, a \rangle$
  - i.e., adding new triples with old elements only
  - Prevents skewing of existing data and noise
- Implements “conservative extensions” for RDF graphs
- Good fit for OWL LD (Linked Data) profile

Example

- Adding new data must not re-arrange existing data
  - $\langle n, a, a \rangle$
    $\langle a, n, a \rangle$
    $\langle a, a, n \rangle$
  - Disallows $\langle a, a, a \rangle$
  - i.e., adding new triples with old elements only
  - Prevents skewing of existing data and noise
- Implements "conservative extensions" for RDF graphs
- Good fit for OWL LD (Linked Data) profile
  - OWL LD = OWL RL $\cap$ single triple statements

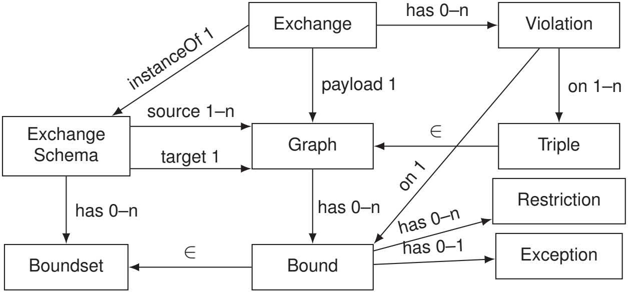
Thanks

Implementation available:

- Prototype implementation: checks bounds, computes payloads
- Complexity: P

- More info/material on the web:
  - `http://sws.ifi.uio.no/project/boundz/`
  - `http://sws.ifi.uio.no/vocab/boundz`

Boundz vocabulary



Simplified overview of:

`http://sws.ifi.uio.no/vocab/boundz`

Vocabulary example

BBC Music dataset restrictions:

1. The vocabulary that the BBC uses must not be hijacked by adding new superclasses or superproperties.
2. Adding new `foaf:made` relationships is not tolerated, unless both artist and record is new to the BBC dataset; their current library is regarded as complete with respect to the albums of enlisted artists, but is open for extensions with new artists.
3. More fanpages may be added, but an existing fanpage cannot be related to more artists.
4. No new information about existing genres may be added.
5. Also, assume the BBC keeps a special dataset about the Beatles which is not under their management, so they want to disallow any new information using only elements from this dataset. However, new information may *relate* to the Beatles dataset.

```
ex:bbcmusic a bz:BoundedGraph ;
  bz:hasBound bzs:RDFS ,
              [ a bz:Aso ;  bz:predicateValue foaf:made ] ,
              [ a bz:o ;    bz:predicateValue mo:fanpage ;
                            bz:hasException bz:ignoreViolations ] ,
              [ a bz:T ;    bz:subjectClass mo:Genre ] ,
              [ a bz:T ;    bz:objectClass mo:Genre ] .
ex:beatles a bz:BoundedGraph ;
  bz:hasBound [ a bz:KKspo ] .
```

Boundz output example

```
<file:///test/test>
      a       :ExchangeSchema ;
      :hasBound _:b1 , _:b2 , _:b3 , _:b4 , _:b5 , _:b6 , _:b7 , _:b8 ;
      :hasSource <file:///test/uni1_0.ttl> , <file:///test/uni1_1.ttl> ;
      :hasTarget <file:///test/uni500_2.ttl> ;
      :outputPayload "false"^^xsd:boolean ;
      :outputViolations "false"^^xsd:boolean ;
      :sourceReasoning "false"^^xsd:boolean ;
      :targetReasoning "false"^^xsd:boolean .

<http:///test/test/1372168337206/31381>
      a       :Exchange ;
      :hasPayload   [ :noOfTriples "6498"^^xsd:long ] ;
      :hasViolation [ a       :Violation ;
                      :hasSource <file:///test/uni1_1.ttl> ;
                      :noOfTriples "566"^^xsd:long ;
                      :onBound _:b7 ] ;
      :instanceOf <file:///test/test> ;
      :timestamp "1372168337206"^^xsd:long .

_:b7  a       :T ;
      :classRestriction ub:University ;
      :hasException :ignoreViolations ;
      :hasRestriction ub:University ;
      :objectClass ub:University ;
      :objectRestriction ub:University .
```

Excerpt of `http://sws.ifi.uio.no/project/boundz/impl/outputExchanges.ttl`

SPARQL query

SPARQL representation of `bz:KKspo`:

```
CONSTRUCT
  { ?s ?p ?o .}
WHERE
  { GRAPH <SOURCE>
      { ?s ?p ?o
        GRAPH <TARGET>
          { { { { ?s ?_5 _:b0 } UNION { _:b1 ?s _:b2 } UNION { _:b3 ?_6 ?s } }
              { { ?p ?_3 _:b4 } UNION { _:b5 ?p _:b6 } UNION { _:b7 ?_4 ?p } } }
            {   { ?o ?_1 _:b8 } UNION { _:b9 ?o _:b10 } UNION { _:b11 ?_2 ?o } }
          }
      }
    MINUS
      { GRAPH <TARGET>
          { ?s ?p ?o }
      }
  }
```

References

- A. Hogan, A. Harth, and A. Polleres. “Scalable Authoritative OWL Reasoning for the Web”. In: *Int. Journal on Semantic Web and Information Systems* 5.2 (2009), pp. 49–90
- A. Stolpe and M. G. Skjæveland. “Preserving Information Content in RDF Using Bounded Homomorphisms”. In: *The Semantic Web: Research and Applications*. Vol. 7295. LNCS. Proc. of the 9th ESWC 2012. 2012, pp. 72–86
- B. Glimm et al. “OWL: Yet to arrive on the Web of Data?” In: *Proc. of the WWW2012 Workshop on Linked Data on the Web (LDOW 2012)*. 2012