Unit 2 – RDF Formal Semantics in Detail

Axel Polleres

Siemens AG Österreich

VU 184.729 Semantic Web Technologies

Where are we?

- Last time we learnt:
  - Basic ideas about RDF and how it is published
  - Turtle Syntax for RDF - we know how to write RDF
  - Basic SPARQL queries - we (roughly) know how to query RDF
- We skipped:
  - Overview of RDF Schema ... we will get to that today
  - Overview of OWL ... we will get to that later
- Next today:
  - RDF formal semantics...
  - ... which will be the basis for SPARQL's formal semantics
  - ... and also for RDF Schema & OWL
- Next time:
- ... continue with RDFS semantics & SPARQL semantics
- Discuss Assignment 1 (no new assignment today)

Organization

First two organizational items:

1. We need to find a replacement date for 11/05/2012 !
2. Assignments can be improved until Wednesday noon, (re-)send them to **axel.polleres@siemens.com** (please use this email only)

Unit Outline

1. What does that data mean? Ontologies described in RDFS + OWL
2. RDF Graph – Formal Definitions
3. RDF Interpretations and Simple Entailment
4. APPENDIX: Simple RDF Entailment is NP-complete

Unit Outline

1. What does that data mean? Ontologies described in RDFS + OWL
2. RDF Graph – Formal Definitions
3. RDF Interpretations and Simple Entailment
4. APPENDIX: Simple RDF Entailment is NP-complete

What does RDF data mean?

- *Ontologies* are formal descriptions of what the *vocabulary* used in an RDF document means.

[1]"data" rather than "ontology", in DL terminology this distinction is often called ABox vs. TBox.

What does RDF data mean?

- *Ontologies* are formal descriptions of what the *vocabulary* used in an RDF document means.
- By vocabulary, we mean here mostly:
  - *properties*,i.e., predicates
  - *classes*, i.e., objects of `rdf:type` triples
  - (*individuals*, i.e., concrete objects)[1]

[1] "data" rather than "ontology", in DL terminology this distinction is often called ABox vs. TBox.

What does RDF data mean?

- *Ontologies* are formal descriptions of what the *vocabulary* used in an RDF document means.
- By vocabulary, we mean here mostly:
  - *properties*,i.e., predicates
  - *classes*, i.e., objects of `rdf:type` triples
  - (*individuals*, i.e., concrete objects)[1]
- Ontologies describe ***relations*** among properties, classes and individuals (subclasses, subproperties, equivalence, domain, range, etc.)

[1]"data" rather than "ontology", in DL terminology this distinction is often called ABox vs. TBox.

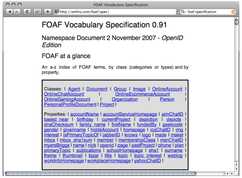
What does RDF data mean?

- *Ontologies* are formal descriptions of what the *vocabulary* used in an RDF document means.
- By vocabulary, we mean here mostly:
  - *properties*,i.e., predicates
  - *classes*, i.e., objects of `rdf:type` triples
  - (*individuals*, i.e., concrete objects)[1]
- Ontologies describe ***relations*** among properties, classes and individuals (subclasses, subproperties, equivalence, domain, range, etc.)
- The W3C has published two standards to describe ontologies, namely *RDF Schema (RDFS)* [Brickley and Guha (eds.), 2004] and the *Web Ontology language (OWL)* [Patel-Schneider *et al.*, 2004]
  - RDFS . . . simple schema language with minimal expressivity, mostly expressible in simple forward chaining inference rules (*Horn Rules*)
  - OWL . . . higher expressivity, foundations in *Description Logics*
  - both RDFS and OWL ontologies are RDF graphs themselves, i.e., OWL and RDFS provide "an RDF vocabulary to describe RDF vocabularies"

[1]"data" rather than "ontology", in DL terminology this distinction is often called ABox vs. TBox.

Example Vocabulary – The FOAF ontology:

- **Properties:** `foaf:name`, `foaf:knows`, `foaf:homepage`, `foaf:primaryTopic` etc.
- **Classes:** `foaf:Person`, `foaf:Agent`, `foaf:Document`, `foaf:Organisation`, etc.
- **Relations:** e.g.
  - *Each `Person` is a `Agent`* (subclass)



FOAF Vocabulary Specification 0.91

Namespace Document 2 November 2007 - *OpenID Edition*

FOAF at a glance

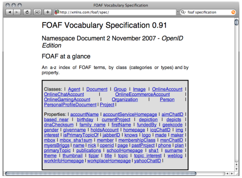
An a-z index of FOAF terms, by class (categories or types) and by property.

Classes: | Agent | Document | Group | Image | OnlineAccount | OnlineChatAccount | OnlineEcommerceAccount | OnlineGamingAccount | Organization | Person | PersonalProfileDocument | Project |

Properties: | accountName | accountServiceHomepage | aimChatID | based_near | birthday | currentProject | depiction | depicts | dnaChecksum | family_name | firstName | fundedBy | geekcode | gender | givenname | holdsAccount | homepage | icqChatID | img | interest | isPrimaryTopicOf | jabberID | knows | logo | made | maker | mbox | mbox_sha1sum | member | membershipClass | msnChatID | myersBriggs | name | nick | openid | page | pastProject | phone | plan | primaryTopic | publications | schoolHomepage | sha1 | surname | theme | thumbnail | tipjar | title | topic | topic_interest | weblog | workInfoHomepage | workplaceHomepage | yahooChatID |

Example Vocabulary – The FOAF ontology:

- **Properties:** `foaf:name`, `foaf:knows`, `foaf:homepage`, `foaf:primaryTopic` etc.
- **Classes:** `foaf:Person`, `foaf:Agent`, `foaf:Document`, `foaf:Organisation`, etc.
- **Relations:** e.g.
  - *Each `Person` is a `Agent`* (subclass)
  - *The `img` property is more specific than `depiction`* (subproperty)



FOAF Vocabulary Specification 0.91

Namespace Document 2 November 2007 - *OpenID Edition*

FOAF at a glance

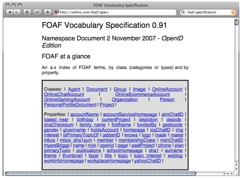
An a-z index of FOAF terms, by class (categories or types) and by property.

Classes: | Agent | Document | Group | Image | OnlineAccount | OnlineChatAccount | OnlineEcommerceAccount | OnlineGamingAccount | Organization | Person | PersonalProfileDocument | Project |

Properties: | accountName | accountServiceHomepage | aimChatID | based_near | birthday | currentProject | depiction | depicts | dnaChecksum | family_name | firstName | fundedBy | geekcode | gender | givenname | holdsAccount | homepage | icqChatID | img | interest | isPrimaryTopicOf | jabberID | knows | logo | made | maker | mbox | mbox_sha1sum | member | membershipClass | msnChatID | myersBriggs | name | nick | openid | page | pastProject | phone | plan | primaryTopic | publications | schoolHomepage | sha1 | surname | theme | thumbnail | tipjar | title | topic | topic_interest | weblog | workInfoHomepage | workplaceHomepage | yahooChatID |

Example Vocabulary – The FOAF ontology:

- **Properties:** `foaf:name`, `foaf:knows`, `foaf:homepage`, `foaf:primaryTopic` etc.
- **Classes:** `foaf:Person`, `foaf:Agent`, `foaf:Document`, `foaf:Organisation`, etc.
- **Relations:** e.g.
  - *Each `Person` is a `Agent`* (subclass)
  - *The `img` property is more specific than `depiction`* (subproperty)
  - *`img` is a relation between `Persons` and `Imgages`* (domain/range)



FOAF Vocabulary Specification 0.91

Namespace Document 2 November 2007 - *OpenID Edition*

FOAF at a glance

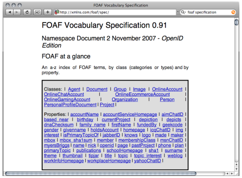
An a-z index of FOAF terms, by class (categories or types) and by property.

Classes: | Agent | Document | Group | Image | OnlineAccount | OnlineChatAccount | OnlineEcommerceAccount | OnlineGamingAccount | Organization | Person | PersonalProfileDocument | Project |

Properties: | accountName | accountServiceHomepage | aimChatID | based_near | birthday | currentProject | depiction | depicts | dnaChecksum | family_name | firstName | fundedBy | geekcode | gender | givenname | holdsAccount | homepage | icqChatID | img | interest | isPrimaryTopicOf | jabberID | knows | logo | made | maker | mbox | mbox_sha1sum | member | membershipClass | msnChatID | myersBriggs | name | nick | openid | page | pastProject | phone | plan | primaryTopic | publications | schoolHomepage | sha1 | surname | theme | thumbnail | tipjar | title | topic | topic_interest | weblog | workInfoHomepage | workplaceHomepage | yahooChatID |

Example Vocabulary – The FOAF ontology:

- **Properties:** `foaf:name`, `foaf:knows`, `foaf:homepage`, `foaf:primaryTopic` etc.
- **Classes:** `foaf:Person`, `foaf:Agent`, `foaf:Document`, `foaf:Organisation`, etc.
- **Relations:** e.g.
  - *Each `Person` is a `Agent`* (subclass)
  - *The `img` property is more specific than `depiction`* (subproperty)
  - *`img` is a relation between `Persons` and `Imgages`* (domain/range)
  - *`knows` is a relation between two `Persons`* (domain/range)



FOAF Vocabulary Specification 0.91

Namespace Document 2 November 2007 - *OpenID Edition*

FOAF at a glance

An a-z index of FOAF terms, by class (categories or types) and by property.

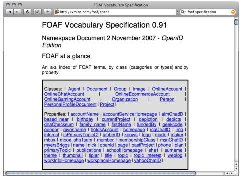
Classes: | Agent | Document | Group | Image | OnlineAccount | OnlineChatAccount | OnlineEcommerceAccount | OnlineGamingAccount | Organization | Person | PersonalProfileDocument | Project |

Properties: | accountName | accountServiceHomepage | aimChatID | based_near | birthday | currentProject | depiction | depicts | dnaChecksum | family_name | firstName | fundedBy | geekcode | gender | givenname | holdsAccount | homepage | icqChatID | img | interest | isPrimaryTopicOf | jabberID | knows | logo | made | maker | mbox | mbox_sha1sum | member | membershipClass | msnChatID | myersBriggs | name | nick | openid | page | pastProject | phone | plan | primaryTopic | publications | schoolHomepage | sha1 | surname | theme | thumbnail | tipjar | title | topic | topic_interest | weblog | workInfoHomepage | workplaceHomepage | yahooChatID |

Example Vocabulary – The FOAF ontology:

- **Properties:** `foaf:name`, `foaf:knows`, `foaf:homepage`, `foaf:primaryTopic` etc.
- **Classes:** `foaf:Person`, `foaf:Agent`, `foaf:Document`, `foaf:Organisation`, etc.
- **Relations:** e.g.
  - *Each `Person` is a `Agent`* (subclass)
  - *The `img` property is more specific than `depiction`* (subproperty)
  - *`img` is a relation between `Persons` and `Imgages`* (domain/range)
  - *`knows` is a relation between two `Persons`* (domain/range)
  - *`homepage` denotes **unique** homepage of an `Agent`* (uniquely identifying property)

⋮



FOAF Vocabulary Specification 0.91

Namespace Document 2 November 2007 - *OpenID Edition*

FOAF at a glance

An a-z index of FOAF terms, by class (categories or types) and by property.

Classes: | Agent | Document | Group | Image | OnlineAccount | OnlineChatAccount | OnlineEcommerceAccount | OnlineGamingAccount | Organization | Person | PersonalProfileDocument | Project |

Properties: | accountName | accountServiceHomepage | aimChatID | based_near | birthday | currentProject | depiction | depicts | dnaChecksum | family_name | firstName | fundedBy | geekcode | gender | givenname | holdsAccount | homepage | icqChatID | img | interest | isPrimaryTopicOf | jabberID | knows | logo | made | maker | mbox | mbox_sha1sum | member | membershipClass | msnChatID | myersBriggs | name | nick | openid | page | pastProject | phone | plan | primaryTopic | publications | schoolHomepage | sha1 | surname | theme | thumbnail | tipjar | title | topic | topic_interest | weblog | workInfoHomepage | workplaceHomepage | yahooChatID |

RDF(S) vocabulary: RDF and RDFS themselves are vocabularies!

- **Properties:** `rdf:type`, `rdfs:domain`, `rdfs:range`, `rdfs:subClassOf`, `rdfs:subPropertyOf`, `rdf:first`, `rdf:rest` etc.
- **Classes:** `rdf:XMLLiteral`, `rdf:Literal`, `rdfs:Resource`,`rdf:Property`, `rdfs:Class`, `rdf:List`, etc.
- **Relations:**

RDF(S) vocabulary: RDF and RDFS themselves are vocabularies!

- **Properties:** `rdf:type`, `rdfs:domain`, `rdfs:range`, `rdfs:subClassOf`, `rdfs:subPropertyOf`, `rdf:first`, `rdf:rest` etc.
- **Classes:** `rdf:XMLLiteral`, `rdf:Literal`, `rdfs:Resource`,`rdf:Property`, `rdfs:Class`, `rdf:List`, etc.
- **Relations:**The semantics of the RDFs vocabulary is defined in [Hayes, 2004]; it is a "meta vocabulary" used to define the semantics of other vocabularies

The Semantics of RDF graphs:

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
<http://www.mat.unical.it/~ianni/foaf.rdf> a foaf:PersonalProfileDocument.
<http://www.mat.unical.it/~ianni/foaf.rdf> foaf:maker _:me .
<http://www.mat.unical.it/~ianni/foaf.rdf> foaf:primaryTopic _:me .
:me a foaf:Person .
:me foaf:name "Giovambattista Ianni" .
:me foaf:homepage <http://www.gibbi.com> .
:me foaf:phone <tel:+39-0984-496430> .
:me foaf:knows [ a foaf:Person ;
                 foaf:name "Wolfgang Faber" ;
                 rdfs:seeAlso <http://www.kr.tuwien.ac.at/staff/faber/foaf.rdf>].
:me foaf:knows [ a foaf:Person .
                 foaf:name "Axel Polleres" ;
                 rdfs:seeAlso <http://www.polleres.net/foaf.rdf>].
:me foaf:knows [ a foaf:Person .
                 foaf:name "Thomas Eiter" ] .
:me foaf:knows [ a foaf:Person .
                 foaf:name "Alessandra Martello" ] .
```

The Semantics of RDF graphs:

As we will see in the next Units, each RDF graph can – essentially – be viewed as a first-order formula:

$$
\begin{array}{l}
\exists b1, b2, b3, b4 \\
(\mathit{triple}(\texttt{foaf.rdf}, \texttt{rdf:type}, \texttt{PersonalProfileDocument}) \\
\wedge\ \mathit{triple}(\texttt{foaf.rdf}, \texttt{maker}, \mathit{me}) \\
\wedge\ \mathit{triple}(\texttt{foaf.rdf}, \texttt{primaryTopic}, \mathit{me}) \\
\wedge\ \mathit{triple}(\mathit{me}, \texttt{rdf:type}, \texttt{Person}) \\
\wedge\ \mathit{triple}(\mathit{me}, \texttt{name}, \texttt{"Giovambattista Ianni"}) \\
\wedge\ \mathit{triple}(\mathit{me}, \texttt{homepage}, \texttt{http://www.gibbi.com}) \\
\wedge\ \mathit{triple}(\mathit{me}, \texttt{phone}, \texttt{tel:+39-0984-496430}) \\
\wedge\ \mathit{triple}(\mathit{me}, \texttt{knows}, b2) \wedge\ \mathit{triple}(b1, \texttt{type}, \texttt{Person}) \\
\wedge\ \mathit{triple}(b1, \texttt{name}, \texttt{"Wolfgang Faber"}) \\
\wedge\ \mathit{triple}(b1, \texttt{rdfs:seeAlso}, \texttt{http://www.kr.tuwien...}) \\
\wedge\ \mathit{triple}(\mathit{me}, \texttt{knows}, b1) \wedge\ \mathit{triple}(b1, \texttt{rdf:type}, \texttt{Person}) \\
\wedge\ \mathit{triple}(b2, \texttt{name}, \texttt{"Axel Polleres"}) \\
\wedge\ \mathit{triple}(b2, \texttt{rdfs:seeAlso}, \texttt{http://www.polleres...}) \\
\wedge\ \mathit{triple}(\mathit{me}, \texttt{knows}, b3) \wedge\ \mathit{triple}(b1, \texttt{rdf:type}, \texttt{Person}) \\
\wedge\ \mathit{triple}(b3, \texttt{name}, \texttt{"Thomas Eiter"}) \\
\wedge\ \mathit{triple}(\mathit{me}, \texttt{knows}, b4) \wedge\ \mathit{triple}(b1, \texttt{type}, \texttt{Person}) \\
\wedge\ \mathit{triple}(b4, \texttt{name}, \texttt{"Alessandra Martello"}))
\end{array}
$$

The Semantics of the RDFS vocabulary:

The formal semantics of RDF(S) [Hayes, 2004] is accompanied by a set of (informative) entailment rules . . . can be written down as the following first-order formulas:

$$
\begin{array}{l}
\forall S, P, O\ (\mathit{triple}(S, P, O) \supset \mathit{triple}(S, \texttt{rdf:type}, \texttt{rdfs:Resource}))\\
\forall S, P, O\ (\mathit{triple}(S, P, O) \supset \mathit{triple}(P, \texttt{rdf:type}, \texttt{rdf:Property}))\\
\forall S, P, O\ (\mathit{triple}(S, P, O) \supset \mathit{triple}(O, \texttt{rdf:type}, \texttt{rdfs:Resource}))\\
\forall S, P, O\ (\mathit{triple}(S, P, O) \wedge \mathit{triple}(P, \texttt{rdfs:domain}, C) \supset \mathit{triple}(S, \texttt{rdf:type}, C))\\
\forall S, P, O, C\ (\mathit{triple}(S, P, O) \wedge \mathit{triple}(P, \texttt{rdfs:range}, C) \supset \mathit{triple}(O, \texttt{rdf:type}, C))\\
\forall C\ (\mathit{triple}(C, \texttt{rdf:type}, \texttt{rdfs:Class}) \supset \mathit{triple}(C, \texttt{rdfs:subClassOf}, \texttt{rdfs:Resource}))\\
\forall C_1, C_2, C_3\ (\mathit{triple}(C_1, \texttt{rdfs:subClassOf}, C_2) \wedge\\
\qquad\qquad \mathit{triple}(C_2, \texttt{rdfs:subClassOf}, C_3) \supset \mathit{triple}(C_1, \texttt{rdfs:subClassOf}, C_3))\\
\forall S, C_1, C_2\ (\mathit{triple}(S, \texttt{rdf:type}, C_1)\ \wedge \mathit{triple}(C_1, \texttt{rdfs:subClassOf}, C_2) \supset \mathit{triple}(S, \texttt{rdf:type}, C_2))\\
\forall S, C\ (\mathit{triple}(S, \texttt{rdf:type}, C) \supset \mathit{triple}(C, \texttt{rdf:type}, \texttt{rdfs:Class}))\\
\forall C\ (\mathit{triple}(C, \texttt{rdf:type}, \texttt{rdfs:Class}) \supset \mathit{triple}(C, \texttt{rdfs:subClassOf}, C))\\
\forall P_1, P_2, P_3\ (\mathit{triple}(P_1, \texttt{rdfs:subPropertyOf}, P_2) \wedge\\
\qquad\qquad \mathit{triple}(P_2, \texttt{rdfs:subPropertyOf}, P_3) \supset \mathit{triple}(P_1, \texttt{rdfs:subPropertyOf}, P_3))\\
\forall S, P_1, P_2, O\ (\mathit{triple}(S, P_1, O) \wedge \mathit{triple}(P_1, \texttt{rdfs:subPropertyOf}, P_2) \supset \mathit{triple}(S, P_2, O))\\
\forall P\ (\mathit{triple}(P, \texttt{rdf:type}, \texttt{rdf:Property}) \supset \mathit{triple}(P, \texttt{rdfs:subPropertyOf}, P))
\end{array}
$$

plus the axiomatic triples from [Hayes, 2004, Sections 3.1 and 4.1].

The Semantics of the RDFS vocabulary:

The formal semantics of RDF(S) [Hayes, 2004] is accompanied by a set of (informative) entailment rules ... can be written down as the following first-order formulas:

$$\forall S,P,O\,(\mathit{triple}(S,P,O) \supset \mathit{triple}(S,\texttt{rdf:type},\texttt{rdfs:Resource}))$$
$$\forall S,P,O\,(\mathit{triple}(S,P,O) \supset \mathit{triple}(P,\texttt{rdf:type},\texttt{rdf:Property}))$$
$$\forall S,P,O\,(\mathit{triple}(S,P,O) \supset \mathit{triple}(O,\texttt{rdf:type},\texttt{rdfs:Resource}))$$
$$\forall S,P,O\,(\mathit{triple}(S,P,O) \wedge \mathit{triple}(P,\texttt{rdfs:domain},C) \supset \mathit{triple}(S,\texttt{rdf:type},C))$$
$$\forall S,P,O,C\,(\mathit{triple}(S,P,O) \wedge \mathit{triple}(P,\texttt{rdfs:range},C) \supset \mathit{triple}(O,\texttt{rdf:type},C))$$
$$\forall C\,(\mathit{triple}(C,\texttt{rdf:type},\texttt{rdfs:Class}) \supset \mathit{triple}(C,\texttt{rdfs:subClassOf},\texttt{rdfs:Resource}))$$
$$\forall C_1,C_2,C_3\,(\mathit{triple}(C_1,\texttt{rdfs:subClassOf},C_2) \wedge \mathit{triple}(C_2,\texttt{rdfs:subClassOf},C_3) \supset \mathit{triple}(C_1,\texttt{rdfs:subClassOf},C_3))$$
$$\forall S,C_1,C_2\,(\mathit{triple}(S,\texttt{rdf:type},C_1) \wedge \mathit{triple}(C_1,\texttt{rdfs:subClassOf},C_2) \supset \mathit{triple}(S,\texttt{rdf:type},C_2))$$
$$\forall S,C\,(\mathit{triple}(S,\texttt{rdf:type},C) \supset \mathit{triple}(C,\texttt{rdf:type},\texttt{rdfs:Class}))$$
$$\forall C\,(\mathit{triple}(C,\texttt{rdf:type},\texttt{rdfs:Class}) \supset \mathit{triple}(C,\texttt{rdfs:subClassOf},C))$$
$$\forall P_1,P_2,P_3\,(\mathit{triple}(P_1,\texttt{rdfs:subPropertyOf},P_2) \wedge \mathit{triple}(P_2,\texttt{rdfs:subPropertyOf},P_3) \supset \mathit{triple}(P_1,\texttt{rdfs:subPropertyOf},P_3))$$
$$\forall S,P_1,P_2,O\,(\mathit{triple}(S,P_1,O) \wedge \mathit{triple}(P_1,\texttt{rdfs:subPropertyOf},P_2) \supset \mathit{triple}(S,P_2,O))$$
$$\forall P\,(\mathit{triple}(P,\texttt{rdf:type},\texttt{rdf:Property}) \supset \mathit{triple}(P,\texttt{rdfs:subPropertyOf},P))$$

plus the axiomatic triples from [Hayes, 2004, Sections 3.1 and 4.1].

The Semantics of the RDFS vocabulary:

Note:
All those rules were Datalog expressible, i.e. no negation, no function symbols.

RDFS Semantics Example: The FOAF ontology

FOAF Ontology:

- *Each* `Person` *is a* `Agent` (subclass)
- *The* `img` *property is more specific than* `depiction` (subproperty)
- `img` *is a relation between* `Persons` *and* `Imgages` (domain/range)
- `knows` *is a relation between two* `Persons` (domain/range)
- `homepage` *denotes* ***unique*** *homepage of an* `Agent` (uniquely identifying property)

⋮

RDFS: Semantics

⋮

$$\forall S, C_1, C_2\ (\mathit{triple}(S, \texttt{rdf:type}, C_1)\ \wedge \mathit{triple}(C_1, \texttt{rdfs:subClassOf}, C_2) \supset \mathit{triple}(S, \texttt{rdf:type}, C_2))$$

⋮

Data:

```
:me rdf:type foaf:Person .
```

RDFS Semantics Example: The FOAF ontology

FOAF Ontology in RDF:

- `foaf:Person rdfs:subClassOf foaf:Agent .`
- `foaf:img rdfs:subPropertyOf foaf:depiction .`
- `foaf:img rdfs:domain foaf:Person ; rdfs:range foaf:Image .`
- `foaf:knows rdfs:domain foaf:Person ; rdfs:range foaf:Person .`
- ???

$\vdots$

RDFS: Semantics

$\vdots$

$$\forall S, C_1, C_2\ (triple(S, \texttt{rdf:type}, C_1)\ \land triple(C_1, \texttt{rdfs:subClassOf}, C_2) \supset triple(S, \texttt{rdf:type}, C_2))$$

$\vdots$

Data:

```
:me rdf:type foaf:Person .
:me rdf:type foaf:Agent .
```

RDFS Semantics Example: The FOAF ontology

FOAF Ontology in RDF:

- `foaf:Person rdfs:subClassOf foaf:Agent .`
- `foaf:img rdfs:subPropertyOf foaf:depiction .`
- `foaf:img rdfs:domain foaf:Person ; rdfs:range foaf:Image .`
- `foaf:knows rdfs:domain foaf:Person ; rdfs:range foaf:Person .`
- *homepage denotes unique homepage of an Agent* ???

$\vdots$

RDFS: Semantics

$$\vdots$$

$$\forall S, C_1, C_2\ (triple(S, \texttt{rdf:type}, C_1)\ \wedge triple(C_1, \texttt{rdfs:subClassOf}, C_2) \supset triple(S, \texttt{rdf:type}, C_2))$$

$$\vdots$$

Data:

```
:me rdf:type foaf:Person .
:me rdf:type foaf:Agent .
```

Summary

- We should all have a rough idea about where to find RDF now.
- We should all have a rough idea about how to read RDF now.
- We should all have a rough idea of how to query RDF (SPARQL).
- We should all have an idea of how the semantics of RDF vocabularies and data can be described (RDF Schema ... we'll keep for OWL later)

Details to come!

Unit Outline

1. What does that data mean? Ontologies described in RDFS + OWL

2. RDF Graph – Formal Definitions

3. RDF Interpretations and Simple Entailment

4. APPENDIX: Simple RDF Entailment is NP-complete

RDF Graph – Formal Definitions

Let U be the set of URIs, B be the set of blank nodes (or "variables"), $L = L_t \cup L_p \cup L_{lang}$ be the set of literals (i.e., typed, plain, and plain lang-tagged)

An RDF graph, or simply a graph, is a set of RDF triples from $UB \times U \times UBL$.[2]

A vocabulary of a graph V_G is the subset of UL mentioned in the graph.

A graph or triple without blank nodes is also called ground

[2]We write short e.g. UBL for $U \cup B \cup L$.

RDF Graph – Example 1

Node: "edge labels" may appear as nodes and vice versa, e.g.

G_1 :

```
ex:alice foaf:knows ex:bob.
ex:alice foaf:name "Alice".
foaf:knows rdfs:domain foaf:Person.
```

G_2 :

```
ex:alice rdf:type foaf:Person.
```

G_3 :

```
_:alice foaf:knows ex:bob.
_:alice foaf:name _:name.
```

G_4 :

```
_:alice foaf:knows ex:bob.
_:alice foaf:name _:alice.
```

RDF Graph – Example 1

Node: "edge labels" may appear as nodes and vice versa, e.g.

G_1 :

```
ex:alice foaf:knows ex:bob.
ex:alice foaf:name "Alice".
foaf:knows rdfs:domain foaf:Person.
```

G_2 :

```
ex:alice rdf:type foaf:Person.
```

G_3 :

$Alice$ `foaf:knows ex:bob.`
$Alice$ `foaf:name` $Name$.

G_4 :

$Alice$ `foaf:knows ex:bob.`
$Alice$ `foaf:name` $Alice$.

Again, we will occasionally write blank nodes as like this Var, to make clearer that actually they ammount to existentially quantified variables.

RDF Graph – Example 2

That is also a valid RDF graph:

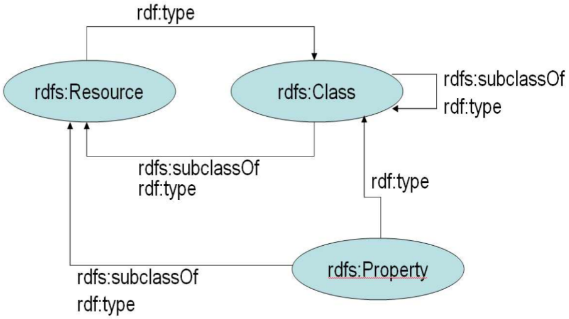
G_5 :

```
rdfs:Resource rdf:type rdfs:Class.
rdf:Property rdf:type rdfs:Resource.
rdf:Property rdfs:subclassOf rdfs:Resource.
rdf:Property rdf:type rdfs:Class.
rdfs:Class rdf:type rdfs:Resource.
rdfs:Class rdf:type rdfs:Class.
rdfs:Class rdfs:subclassOf rdfs:Resource.
rdfs:Class rdfs:subclassOf rdfs:Class.
```

RDF Graph – Example 2

That is also a valid RDF graph:

G_5 :



RDF Graph – Example 3

Or that:

G_6 :

```
rdfs:subClassOf rdfs:subPropertyOf rdfs:Resource.
rdfs:subClassOf rdfs:subPropertyOf rdfs:subPropertyOf.
rdf:type rdfs:subPropertyOf rdfs:subClassOf.
rdfs:subClassOf rdf:type owl:SymmetricProperty.
```

Definitions

Assume a blank node mapping $\mu : B \to UBL$.

Definitions

Assume a blank node mapping $\mu : B \rightarrow UBL$.

By $\mu(G)$ we denote the graph obtained from G by replacing each blank node x with $\mu(x)$.

Definitions

Assume a blank node mapping $\mu : B \to UBL$.

By $\mu(G)$ we denote the graph obtained from G by replacing each blank node x with $\mu(x)$.

We call $\mu(G)$ an *instance* of G.

Definitions

Assume a blank node mapping $\mu : B \rightarrow UBL$.

By $\mu(G)$ we denote the graph obtained from G by replacing each blank node x with $\mu(x)$.

We call $\mu(G)$ an *instance* of G.

A *proper instance* of a graph is an instance in which a blank node has been replaced by a constant (form U or L), or two blank nodes in the graph have been mapped into the same node in the instance.

Definitions

Assume a blank node mapping $\mu : B \rightarrow UBL$.

By $\mu(G)$ we denote the graph obtained from G by replacing each blank node x with $\mu(x)$.

We call $\mu(G)$ an *instance* of G.

A *proper instance* of a graph is an instance in which a blank node has been replaced by a constant (form U or L), or two blank nodes in the graph have been mapped into the same node in the instance.

An RDF graph is *lean* if it has no instance which is a proper subgraph of the graph. Non-lean graphs have internal redundancy and express the same content as their lean subgraphs.

Definitions

Assume a blank node mapping $\mu : B \to UBL$.

By $\mu(G)$ we denote the graph obtained from G by replacing each blank node x with $\mu(x)$.

We call $\mu(G)$ an *instance* of G.

A *proper instance* of a graph is an instance in which a blank node has been replaced by a constant (form U or L), or two blank nodes in the graph have been mapped into the same node in the instance.

An RDF graph is *lean* if it has no instance which is a proper subgraph of the graph. Non-lean graphs have internal redundancy and express the same content as their lean subgraphs.

Two graphs which differ only in the identity of their blank nodes, are considered to be *equivalent*.

Definitions

Assume a blank node mapping $\mu : B \to UBL$.

By $\mu(G)$ we denote the graph obtained from G by replacing each blank node x with $\mu(x)$.

We call $\mu(G)$ an *instance* of G.

A *proper instance* of a graph is an instance in which a blank node has been replaced by a constant (form U or L), or two blank nodes in the graph have been mapped into the same node in the instance.

An RDF graph is *lean* if it has no instance which is a proper subgraph of the graph. Non-lean graphs have internal redundancy and express the same content as their lean subgraphs.

Two graphs which differ only in the identity of their blank nodes, are considered to be *equivalent*.

The *merge* of a set of graphs is obtained by renaming (“standardize apart”) blank nodes in each graph such that no blank nodes between any two graphs are in common and then taking the union of all triples, we write $G1 \uplus G2$ for the graph merge between two graphs $G1, G2$.

Definitions

Assume a blank node mapping $\mu : B \rightarrow UBL$.

By $\mu(G)$ we denote the graph obtained from G by replacing each blank node x with $\mu(x)$.

We call $\mu(G)$ an *instance* of G.

A *proper instance* of a graph is an instance in which a blank node has been replaced by a constant (form U or L), or two blank nodes in the graph have been mapped into the same node in the instance.

An RDF graph is *lean* if it has no instance which is a proper subgraph of the graph. Non-lean graphs have internal redundancy and express the same content as their lean subgraphs.

Two graphs which differ only in the identity of their blank nodes, are considered to be *equivalent*.

The *merge* of a set of graphs is obtained by renaming ("standardize apart") blank nodes in each graph such that no blank nodes between any two graphs are in common and then taking the union of all triples, we write $G1 \uplus G2$ for the graph merge between two graphs $G1, G2$.

Lean and non-lean graphs: Examples

G_7 : non-lean

```
_:x foaf:knows ex:bob.
_:x foaf:knows _:y.
```

G_8 : lean

```
_:x foaf:knows ex:bob.
_:x foaf:knows _:x.
```

Why?

Lean and non-lean graphs: Examples

G_7 : non-lean

$\exists x, y.triple(x, \texttt{knows}, \texttt{bob}) \wedge triple(x, \texttt{knows}, y)$

G_8 : lean

$\exists x, y.triple(x, \texttt{knows}, \texttt{bob}) \wedge triple(x, \texttt{knows}, x)$

Becomes clear if we look at first-order "reading" of the RDF graph, where we treat blank nodes as existential variables and triples in a predicate $triple$.With this reading, one could say: $G'_7 = \{\texttt{\_:x foaf:knows ex:bob.}\} \models G_7$

Lean and non-lean graphs: Examples

G_7 : non-lean

$\exists x, y.triple(x, \texttt{knows}, \texttt{bob}) \models$
$\exists x, y.triple(x, \texttt{knows}, \texttt{bob}) \wedge triple(x, \texttt{knows}, y)$

G_8 : lean

$\exists x, y.triple(x, \texttt{knows}, \texttt{bob}) \not\models$
$\exists x, y.triple(x, \texttt{knows}, \texttt{bob}) \wedge triple(x, \texttt{knows}, x)$

Becomes clear if we look at first-order "reading" of the RDF graph, where we treat blank nodes as existential variables and triples in a predicate $triple$.With this reading, one could say: $G'_7 = \{\texttt{\_:x foaf:knows ex:bob.}\} \models G_7$

We use first-order *entailment* here. Entailment is typically defined in terms of a model theory (interpretation, satisfaction, models)...
RDF has its own model theory!

Unit Outline

1. What does that data mean? Ontologies described in RDFS + OWL

2. RDF Graph – Formal Definitions

3. RDF Interpretations and Simple Entailment

4. APPENDIX: Simple RDF Entailment is NP-complete

Model theoretic semantics – in general

A model theory is usually defined using the following “components”:

- Defining a notion of an interpretation I, consisting of separate interpretation functions
    - i.e., defining how are constants, variables and logical conectives, formulas being “interpreted” in a possible real world.
- A satisfaction relation between interpretations and theories (in our case graphs), written $I \models G$, which says:
    - I is an interpretation satisfying G, or I is a model of G
- An entailment relation between theories (in our case graphs), written $G \models G'$, which says
    - all models of G are also models of G'

Simple Interpretations1/4

"*interpretation I: . . . i.e. how are constants, variables, predicates, formulas being "interpreted" in a possible real world.*"

What does that mean for RDF?

- RDF "constants" . . . subjects, objects, i.e. UL
- RDF "variables" . . . blank nodes, i.e. B
- RDF "predicates" . . . predicates, i.e. U
- RDF "formulas" . . . triples, graphs.

Simple Interpretations1/4

"*interpretation* I*: . . . i.e. how are constants, variables, predicates, formulas being "interpreted" in a possible real world.*"

What does that mean for RDF?

- RDF "constants" . . . subjects, objects, i.e. UL
- RDF "variables" . . . blank nodes, i.e. B
- RDF "predicates" . . . predicates, i.e. U
- RDF "formulas" . . . triples, graphs.

Now here we have something unlike classical logic... URIs can actually need to be interpreted "as predicates" or "as constants" depending on where they appear in the graph.

To cater for that, RDF defines a very general notion of interpretation.

Simple Interpretations 2/4

A simple interpretation I over vocabulary V is a 6-tuple $I = \langle IR, IP, IEXT, IS, IL, LV \rangle$, s.t.

1. A non-empty set IR of resources.
2. A set IP, called the set of properties of I,
3. A mapping $IEXT : IP \rightarrow 2^{(IR \times IR)}$, i.e. assigns a set of pairs $\langle x, y \rangle$ with $x, y \in IR$.
4. A mapping $IS : U \cap V \rightarrow IR \cup IP$
5. A mapping $IL : L_t \cap V$ into IR.
6. A distinguished subset $LV \subset IR$, called the set of literal values, which contains all the plain literals in V, i.e. $LV \subseteq L_p \cup L_{lang}$.

Simple Interpretations 2/4

A simple interpretation I over vocabulary V is a 6-tuple $I = \langle IR, IP, IEXT, IS, IL, LV \rangle$, s.t.

1. A non-empty set IR of resources.
   - *called the domain or universe of I*
2. A set IP, called the set of properties of I,
   - *not necessaily disjoint of IR!*
3. A mapping $IEXT : IP \rightarrow 2^{(IR \times IR)}$, i.e. assigns a set of pairs $\langle x, y \rangle$ with $x, y \in IR$.
   - *intuitivelty, assigns a binary relation between subjects and objects to properties.*
4. A mapping $IS : U \cap V \rightarrow IR \cup IP$
   - *this basically says, URIs can be both constants and predicates*
5. A mapping $IL : L_t \cap V$ into IR.
   - *typed literals are constants.*
6. A distinguished subset $LV \subset IR$, called the set of literal values, which contains all the plain literals in V, i.e. $LV \subseteq L_p \cup L_{lang}$.
   - *plain literals in RDF are special, they are always interpreted as themselves*

Simple Interpretations 2/4

A simple interpretation I over vocabulary V is a 6-tuple $I = \langle IR, IP, IEXT, IS, IL, LV \rangle$, s.t.

1. A non-empty set IR of resources.
   - *called the domain or universe of I*
2. A set IP, called the set of properties of I,
   - *not necessaily disjoint of IR!*
3. A mapping $IEXT : IP \rightarrow 2^{(IR \times IR)}$, i.e. assigns a set of pairs $\langle x, y \rangle$ with $x, y \in IR$.
   - *intuitivelty, assigns a binary relation between subjects and objects to properties.*
4. A mapping $IS : U \cap V \rightarrow IR \cup IP$
   - *this basically says, URIs can be both constants and predicates*
5. A mapping $IL : L_t \cap V$ into IR.
   - *typed literals are constants.*
6. A distinguished subset $LV \subset IR$, called the set of literal values, which contains all the plain literals in V, i.e. $LV \subseteq L_p \cup L_{lang}$.
   - *plain literals in RDF are special, they are always interpreted as themselves*

Simple Interpretations 2/4

A simple interpretation I over vocabulary V is a 6-tuple $I = \langle IR, IP, IEXT, IS, IL, LV \rangle$, s.t.

1. A non-empty set IR of resources.
   - *called the domain or universe of I*
2. A set IP, called the set of properties of I,
   - *not necessaily disjoint of IR!*
3. A mapping $IEXT : IP \rightarrow 2^{(IR \times IR)}$, i.e. assigns a set of pairs $\langle x, y \rangle$ with $x, y \in IR$.
   - *intuitivelty, assigns a binary relation between subjects and objects to properties.*
4. A mapping $IS : U \cap V \rightarrow IR \cup IP$
   - *this basically says, URIs can be both constants and predicates*
5. A mapping $IL : L_t \cap V$ into IR.
   - *typed literals are constants.*
6. A distinguished subset $LV \subset IR$, called the set of literal values, which contains all the plain literals in V, i.e. $LV \subseteq L_p \cup L_{lang}$.
   - *plain literals in RDF are special, they are always interpreted as themselves*

Simple Interpretations 2/4

A simple interpretation I over vocabulary V is a 6-tuple $I = \langle IR, IP, IEXT, IS, IL, LV \rangle$, s.t.

1. A non-empty set IR of resources.
   - *called the domain or universe of I*
2. A set IP, called the set of properties of I,
   - *not necessaily disjoint of IR!*
3. A mapping $IEXT : IP \rightarrow 2^{(IR \times IR)}$, i.e. assigns a set of pairs $\langle x, y \rangle$ with $x, y \in IR$.
   - *intuitivelty, assigns a binary relation between subjects and objects to properties.*
4. A mapping $IS : U \cap V \rightarrow IR \cup IP$
   - *this basically says, URIs can be both constants and predicates*
5. A mapping $IL : L_t \cap V$ into IR.
   - *typed literals are constants.*
6. A distinguished subset $LV \subset IR$, called the set of literal values, which contains all the plain literals in V, i.e. $LV \subseteq L_p \cup L_{lang}$.
   - *plain literals in RDF are special, they are always interpreted as themselves*

Simple Interpretations 2/4

A simple interpretation I over vocabulary V is a 6-tuple $I = \langle IR, IP, IEXT, IS, IL, LV \rangle$, s.t.

1. A non-empty set IR of resources.
   - *called the domain or universe of I*
2. A set IP, called the set of properties of I,
   - *not necessaily disjoint of IR!*
3. A mapping $IEXT : IP \rightarrow 2^{(IR \times IR)}$, i.e. assigns a set of pairs $\langle x, y \rangle$ with $x, y \in IR$.
   - *intuitivelty, assigns a binary relation between subjects and objects to properties.*
4. A mapping $IS : U \cap V \rightarrow IR \cup IP$
   - *this basically says, URIs can be both constants and predicates*
5. A mapping $IL : L_t \cap V$ into IR.
   - *typed literals are constants.*
6. A distinguished subset $LV \subset IR$, called the set of literal values, which contains all the plain literals in V, i.e. $LV \subseteq L_p \cup L_{lang}$.
   - *plain literals in RDF are special, they are always interpreted as themselves*

Simple Interpretations 2/4

A simple interpretation I over vocabulary V is a 6-tuple $I = \langle IR, IP, IEXT, IS, IL, LV \rangle$, s.t.

1. A non-empty set IR of resources.
   - *called the domain or universe of I*
2. A set IP, called the set of properties of I,
   - *not necessaily disjoint of IR!*
3. A mapping $IEXT : IP \rightarrow 2^{(IR \times IR)}$, i.e. assigns a set of pairs $\langle x, y \rangle$ with $x, y \in IR$.
   - *intuitivelty, assigns a binary relation between subjects and objects to properties.*
4. A mapping $IS : U \cap V \rightarrow IR \cup IP$
   - *this basically says, URIs can be both constants and predicates*
5. A mapping $IL : L_t \cap V$ into IR.
   - *typed literals are constants.*
6. A distinguished subset $LV \subset IR$, called the set of literal values, which contains all the plain literals in V, i.e. $LV \subseteq L_p \cup L_{lang}$.
   - *plain literals in RDF are special, they are always interpreted as themselves*

Simple Interpretations 3/4

Interpreting ground graphs (i.e. without blank nodes):

- Interpreting constants:
  - if $e = \text{"}\texttt{aaa}\text{"} \in V \cap L_p$, then $I(e) = aaa \in LV$
  - if $e = \text{"}\texttt{aaa}\text{"}@\texttt{ttt} \in V \cap L_{lang}$, then $I(e) =< aaa, ttt >\in LV$
  - if $e \in V \cap L_t$, then $I(e) = IL(e)$
  - if $e \in V \cap U$, then $I(e) = IS(e)$

Simple Interpretations 3/4

Interpreting ground graphs (i.e. without blank nodes):

- Interpreting constants:
  - if $e =$ "aaa" $\in V \cap L_p$, then $I(e) = aaa \in LV$
  - if $e =$ "aaa"@ttt $\in V \cap L_{lang}$, then $I(e) =< aaa, ttt >\in LV$
  - if $e \in V \cap L_t$, then $I(e) = IL(e)$
  - if $e \in V \cap U$, then $I(e) = IS(e)$
- Interpreting ground triples:
  - if $t =$ s p o., is a ground triple, then
    - $I(t) = true$ if $\mathtt{s}, \mathtt{p}, \mathtt{o} \in V \wedge I(\mathtt{p}) \in IP \wedge \langle I(\mathtt{s}), I(\mathtt{o})\rangle \in IEXT(I(\mathtt{p}))$
    - $I(t) = false$, otherwise

Simple Interpretations 3/4

Interpreting ground graphs (i.e. without blank nodes):

- Interpreting constants:
  - if $e =$ "aaa" $\in V \cap L_p$, then $I(e) = aaa \in LV$
  - if $e =$ "aaa"@ttt $\in V \cap L_{lang}$, then $I(e) =< aaa, ttt >\in LV$
  - if $e \in V \cap L_t$, then $I(e) = IL(e)$
  - if $e \in V \cap U$, then $I(e) = IS(e)$
- Interpreting ground triples:
  - if $t =$ s p o., is a ground triple, then
    - $I(t) = true$ if $\mathtt{s}, \mathtt{p}, \mathtt{o} \in V \wedge I(\mathtt{p}) \in IP \wedge \langle I(\mathtt{s}), I(\mathtt{o})\rangle \in IEXT(I(\mathtt{p}))$
    - $I(t) = false$, otherwise
- Interpreting ground graphs:
  - if G is a ground RDF graph then $I(G) = true$ if and only if $I(t) = true$ for all triples $t \in G$, .

Simple Interpretations 3/4

Interpreting ground graphs (i.e. without blank nodes):

- Interpreting constants:
  - if $e = $ "aaa" $\in V \cap L_p$, then $I(e) = aaa \in LV$
  - if $e = $ "aaa"@ttt $\in V \cap L_{lang}$, then $I(e) =< aaa, ttt > \in LV$
  - if $e \in V \cap L_t$, then $I(e) = IL(e)$
  - if $e \in V \cap U$, then $I(e) = IS(e)$

- Interpreting ground triples:
  - if $t = $ s p o., is a ground triple, then
    - $I(t) = true$ if $\mathtt{s}, \mathtt{p}, \mathtt{o} \in V \wedge I(\mathtt{p}) \in IP \wedge \langle I(\mathtt{s}), I(\mathtt{o})\rangle \in IEXT(I(\mathtt{p}))$
    - $I(t) = false$, otherwise

- Interpreting ground graphs:
  - if G is a ground RDF graph then $I(G) = true$ if and only if $I(t) = true$ for all triples $t \in G$, .

Satisfaction

If $I(G) = true$ we also say I satisfies G, written $I \models G$

Simple Interpretation – Example ground graphs

Take the following artificial vocabulary:
$\{\texttt{ex:a}, \texttt{ex:b}, \texttt{ex:c}, "\texttt{whatever}", "\texttt{whatever}"^{\wedge\wedge}\texttt{ex:b}\}$

$IR = LV \cup \{1, 2\}$

$IP = \{1\}$

$IEXT(1) = \{< 1, 2 >, < 2, 1 >\}$

$IS(\texttt{ex:a}) = IS(\texttt{ex:b}) = 1, IS(\texttt{ex:c}) = 2$

$IL("\texttt{whatever}"^{\wedge\wedge}\texttt{ex:b}) = 2$

Simple Interpretation – Example ground graphs

Take the following artificial vocabulary:
$\{\texttt{ex:a}, \texttt{ex:b}, \texttt{ex:c}, \texttt{"whatever"}, \texttt{"whatever"}^{\wedge\wedge}\texttt{ex:b}\}$
$IR = LV \cup \{1, 2\}$

$IP = \{1\}$

$IEXT(1) = \{<1, 2>, <2, 1>\}$

$IS(\texttt{ex:a}) = IS(\texttt{ex:b}) = 1, IS(\texttt{ex:c}) = 2$

$IL(\texttt{"whatever"}^{\wedge\wedge}\texttt{ex:b}) = 2$

G_9 :

```
ex:a ex:b ex:c .
ex:c ex:a ex:a .
ex:c ex:b ex:a .
ex:a ex:b "whatever"^^ex:b .
```

$I(G_9) = true$, i.e., $I \models G_9$:

Simple Interpretation – Example ground graphs

Take the following artificial vocabulary:
$\{\texttt{ex:a}, \texttt{ex:b}, \texttt{ex:c}, \texttt{"whatever"}, \texttt{"whatever"}\hat{}\hat{}\texttt{ex:b}\}$

$IR = LV \cup \{1, 2\}$

$IP = \{1\}$

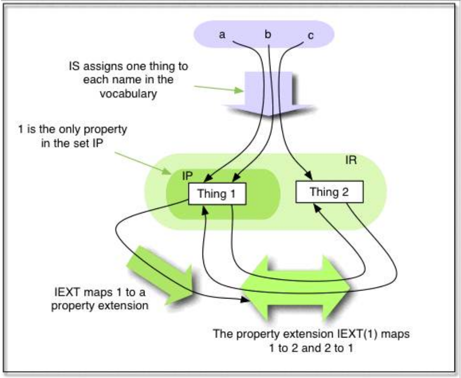
$IEXT(1) = \{<1, 2>, <2, 1>\}$

$IS(\texttt{ex:a}) = IS(\texttt{ex:b}) = 1, IS(\texttt{ex:c}) = 2$

$IL(\texttt{"whatever"}\hat{}\hat{}\texttt{ex:b}) = 2$

G_9 :

```
ex:a ex:b ex:c .
ex:c ex:a ex:a .
ex:c ex:b ex:a .
ex:a ex:b "whatever"^^ex:b .
```


a
b
c
IS assigns one thing to each name in the vocabulary
1 is the only property in the set IP
IR
IP
Thing 1
Thing 2
IEXT maps 1 to a property extension
The property extension IEXT(1) maps 1 to 2 and 2 to 1

$I(G_9) = true$, i.e., $I \models G_9$:

Simple Interpretation – Example ground graphs

Take the following artificial vocabulary:
$\{\texttt{ex:a}, \texttt{ex:b}, \texttt{ex:c}, \texttt{"whatever"}, \texttt{"whatever"}^{\wedge\wedge}\texttt{ex:b}\}$

$IR = LV \cup \{1, 2\}$

$IP = \{1\}$

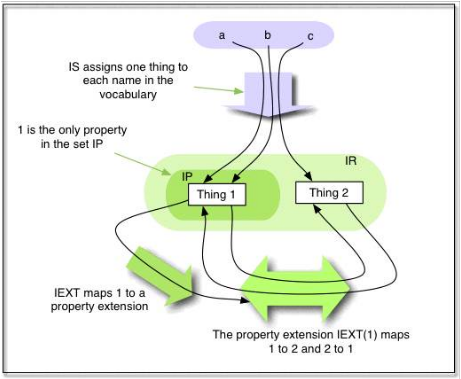
$IEXT(1) = \{<1, 2>, <2, 1>\}$

$IS(\texttt{ex:a}) = IS(\texttt{ex:b}) = 1, IS(\texttt{ex:c}) = 2$

$IL(\texttt{"whatever"}^{\wedge\wedge}\texttt{ex:b}) = 2$

G'_9 :

```
ex:a ex:c ex:b .
ex:a ex:b ex:b .
ex:c ex:b ex:c .
ex:a ex:b "whatever".
```


a
b
c
IS assigns one thing to each name in the vocabulary
1 is the only property in the set IP
IR
IP
Thing 1
Thing 2
IEXT maps 1 to a property extension
The property extension IEXT(1) maps 1 to 2 and 2 to 1

$I(G'_9) = false$, i.e., I doesn't satisfy **any** triple in G'_9:

Simple Interpretation – Example ground graphs

Take the following artificial vocabulary:
$\{\texttt{ex:a}, \texttt{ex:b}, \texttt{ex:c}, \texttt{"whatever"}, \texttt{"whatever"}\hat{}\hat{}\texttt{ex:b}\}$
$IR = LV \cup \{1,2\}$
$IP = \{1\}$
$IEXT(1) = \{<1,2>, <2,1>\}$
$IS(\texttt{ex:a}) = IS(\texttt{ex:b}) = 1, IS(\texttt{ex:c}) = 2$
$IL(\texttt{"whatever"}\hat{}\hat{}\texttt{ex:b}) = 2$

G'_9 :

`ex:a ex:c ex:b .`	$IS(\texttt{ex:c}) = 2 \notin IP$
`ex:a ex:b ex:b .`	$\langle 1,1 \rangle \notin IEXT(IS(\texttt{ex:b}))$
`ex:c ex:b ex:c .`	$\langle 2,2 \rangle \notin IEXT(IS(\texttt{ex:b}))$
`ex:a ex:b "whatever".`	$\langle 1, "whatever" \rangle \notin IEXT(IS(\texttt{ex:b}))$

$I(G'_9) = false$, i.e., I doesn't satisfy **any** triple in G'_9:

Simple Interpretations 4/4

Dealing with blank nodes is analogously to dealing with existential variables in first-order logic:

We call some function $A : B \rightarrow IR$ an assignment.
Given an interpretation I, and an assignment A, $[I + A]$ is defined just like I, except that it uses A to interpret blank nodes.

- Interpreting non-ground graphs:
  - if G is a non-ground RDF graph then $I(G) = \mathit{true}$ if and only if there exists an assignment A such that $[I + A](G) = \mathit{true}$.

Simple Interpretation – Example non-ground graphs

Same interpretation as before, artificial vocabulary:
$\{\texttt{ex : a}, \texttt{ex : b}, \texttt{ex : c}, \texttt{"whatever"}, \texttt{"whatever"}^{\wedge\wedge}\texttt{ex : b}\}$
$IR = LV \cup \{1, 2\}$

$IP = \{1\}$

$IEXT(1) = \{< 1, 2 >, < 2, 1 >\}$

$IS(\texttt{ex : a}) = IS(\texttt{ex : b}) = 1, IS(\texttt{ex : c}) = 2$

$IL(\texttt{"whatever"}^{\wedge\wedge}\texttt{ex : b}) = 2$

Simple Interpretation – Example non-ground graphs

Same interpretation as before, artificial vocabulary:
$\{\texttt{ex:a}, \texttt{ex:b}, \texttt{ex:c}, \texttt{"whatever"}, \texttt{"whatever"}\hat{}\hat{}\texttt{ex:b}\}$
$IR = LV \cup \{1, 2\}$

$IP = \{1\}$

$IEXT(1) = \{<1, 2>, <2, 1>\}$

$IS(\texttt{ex:a}) = IS(\texttt{ex:b}) = 1, IS(\texttt{ex:c}) = 2$

$IL(\texttt{"whatever"}\hat{}\hat{}\texttt{ex:b}) = 2$

G_{10} :

```
_:x <ex:a> <ex:b> .
<ex:c> <ex:b> _:y .
```

$I(G_{10}) = \mathit{true}$, i.e., $I \models G_{10}$:

E.g. take the assignment $A(x) = 2, A(y) = 1$

Simple Interpretation – Example non-ground graphs

Same interpretation as before, artificial vocabulary:
$\{\texttt{ex:a}, \texttt{ex:b}, \texttt{ex:c}, "\texttt{whatever}", "\texttt{whatever}"\hat{}\hat{}\texttt{ex:b}\}$
$IR = LV \cup \{1,2\}$

$IP = \{1\}$

$IEXT(1) = \{<1,2>, <2,1>\}$

$IS(\texttt{ex:a}) = IS(\texttt{ex:b}) = 1, IS(\texttt{ex:c}) = 2$

$IL("\texttt{whatever}"\hat{}\hat{}\texttt{ex:b}) = 2$

G'_{10} :

```
_:x <ex:a> <ex:b> .
<ex:c> <ex:b> _:x .
```

$I(G'_{10}) = false$, i.e., $I \not\models G'_{10}$:

If A maps x to 1 then the first triple is false, and if it maps it to 2 then the second one.

Simple Entailment between RDF Graphs

The usual entailment relation as we know it from first-order theories:

Simple Entailment

An RDF graph G (simply) entails a graph E, written $G \models E$, if every interpretation which satisfies G also satisfies E

"Entailment is the key idea which connects model-theoretic semantics to real-world applications" [Hayes, 2004] . . . indeed, simple entailment is the key for SPARQL graph pattern matching.

Simple Entailment between RDF Graphs

The usual entailment relation as we know it from first-order theories:

Simple Entailment (for sets of graphs)

A set S of RDF graphs (simply) entails a graph E, written $S \models E$, if every interpretation which satisfies **every member of** S also satisfies E

"Entailment is the key idea which connects model-theoretic semantics to real-world applications" [Hayes, 2004] . . . indeed, simple entailment is the key for SPARQL graph pattern matching.

Simple Entailment - Properties

Merging lemma

The merge of a set S of RDF graphs is entailed by S, and entails every member of S, i.e.
$S \models \biguplus_{s \in S} s$ and $\biguplus_{s \in S} s \models s'$, where $s' \in S$.

Simple Entailment - Properties

Merging lemma

The merge of a set S of RDF graphs is entailed by S, and entails every member of S, i.e.
$S \models \biguplus_{s \in S} s$ and $\biguplus_{s \in S} s \models s'$, where $s' \in S$.

Recall the example from before:
G'_{10}:

```
_:x <ex:a> <ex:b> .
<ex:c> <ex:b> _:x .
```

This example shows the difference of union and merge:
The merge of each triple by itself taken as a singleton graph is NOT equivalent to G'_{10}!

(Recall the definition of merge: Obtained by "standardizing apart" blank nodes.)

Simple Entailment - Properties

Main result for simple RDF inference is:

Interpolation Lemma

S entails a graph E if and only if a subgraph of S is an instance of E.

Simple Entailment - Properties

Main result for simple RDF inference is:

Interpolation Lemma

S entails a graph E if and only if a subgraph of S is an instance of E.

What does this mean?
Recall: We call $\mu(G)$ an *instance* of G, where μ maps blank nodes to UBL.
So, you can test entailment $G \models ?G'$ by

1. guessing a mapping μ and
2. test whether $\mu(G') \subseteq G$

Simple Entailment - Properties

Main result for simple RDF inference is:

Interpolation Lemma

S entails a graph E if and only if a subgraph of S is an instance of E.

What does this mean?
Recall: We call $\mu(G)$ an *instance* of G, where μ maps blank nodes to UBL.
So, you can test entailment $G \models ?G'$ by

1. guessing a mapping μ and
2. test whether $\mu(G') \subseteq G$

Complexity

Simple entailment is NP-complete.

(proof in the end of the slides, time allowed)

Simple Entailment - Examples 1/4

G_1 :

```
ex:alice foaf:knows ex:bob.
ex:alice foaf:name "Alice".
foaf:knows rdfs:domain foaf:Person.
```

G_3 :

```
_:alice foaf:knows ex:bob.
_:alice foaf:name _:name.
```

G_4 :

```
_:alice foaf:knows ex:bob.
_:alice foaf:name _:alice.
```

Simple Entailment - Examples 1/4

G_1 :

```
ex:alice foaf:knows ex:bob.
ex:alice foaf:name "Alice".
foaf:knows rdfs:domain foaf:Person.
```

G_3 :

$Alice$ `foaf:knows ex:bob.`
$Alice$ `foaf:name` $Name$.

G_4 :

$Alice$ `foaf:knows ex:bob.`
$Alice$ `foaf:name` $Alice$.

$G_1 \models G_3$:

Simple Entailment - Examples 1/4

G_1 :

```
ex:alice foaf:knows ex:bob.
ex:alice foaf:name "Alice".
foaf:knows rdfs:domain foaf:Person.
```

G_3 :

$Alice$ `foaf:knows ex:bob.`
$Alice$ `foaf:name` $Name$.

G_4 :

$Alice$ `foaf:knows ex:bob.`
$Alice$ `foaf:name` $Alice$.

$G_1 \models G_3$:
$\mu(Alice) = ex : alice, \mu(Name) = \text{"}Alice\text{"} \Rightarrow \mu(G_3) \subseteq G_1$

Simple Entailment - Examples 1/4

G_1 :

```
ex:alice foaf:knows ex:bob.
ex:alice foaf:name "Alice".
foaf:knows rdfs:domain foaf:Person.
```

G_3 :

$Alice$ `foaf:knows ex:bob.`
$Alice$ `foaf:name` $Name$.

G_4 :

$Alice$ `foaf:knows ex:bob.`
$Alice$ `foaf:name` $Alice$.

$G_1 \not\models G_4$:

Simple Entailment - Examples 1/4

G_1 :

```
ex:alice foaf:knows ex:bob.
ex:alice foaf:name "Alice".
foaf:knows rdfs:domain foaf:Person.
```

G_3 :

$Alice$ `foaf:knows ex:bob.`
$Alice$ `foaf:name` $Name$.

G_4 :

$Alice$ `foaf:knows ex:bob.`
$Alice$ `foaf:name` $Alice$.

$G_1 \not\models G_4$:
no blank node mapping μ makes $\mu(G_4)$ a subset of G_1

Simple Entailment - Examples 1/4

G_1 :

```
ex:alice foaf:knows ex:bob.
ex:alice foaf:name "Alice".
foaf:knows rdfs:domain foaf:Person.
```

G_3 :

$Alice$ `foaf:knows ex:bob.`
$Alice$ `foaf:name` $Name$.

G_4 :

$Alice$ `foaf:knows ex:bob.`
$Alice$ `foaf:name` $Alice$.

$G_3 \not\models G_4$:

Simple Entailment - Examples 1/4

G_1 :

```
ex:alice foaf:knows ex:bob.
ex:alice foaf:name "Alice".
foaf:knows rdfs:domain foaf:Person.
```

G_3 :

$Alice$ `foaf:knows ex:bob.`
$Alice$ `foaf:name` $Name$.

G_4 :

$Alice$ `foaf:knows ex:bob.`
$Alice$ `foaf:name` $Alice$.

$G_3 \not\models G_4$:
no blank node mapping μ makes $\mu(G_4)$ a subset of G_3

Simple Entailment - Examples 1/4

G_1 :

```
ex:alice foaf:knows ex:bob.
ex:alice foaf:name "Alice".
foaf:knows rdfs:domain foaf:Person.
```

G_3 :

$Alice$ `foaf:knows ex:bob.`
$Alice$ `foaf:name` $Name$.

G_4 :

$Alice$ `foaf:knows ex:bob.`
$Alice$ `foaf:name` $Alice$.

$G_4 \models G_3$:

Simple Entailment - Examples 1/4

G_1 :

```
ex:alice foaf:knows ex:bob.
ex:alice foaf:name "Alice".
foaf:knows rdfs:domain foaf:Person.
```

G_3 :

$Alice$ `foaf:knows ex:bob.`
$Alice$ `foaf:name` $Name$.

G_4 :

$Alice$ `foaf:knows ex:bob.`
$Alice$ `foaf:name` $Alice$.

$G_4 \models G_3$:
$\mu(Alice) = Alice, \mu(Name) = Alice \Rightarrow \mu(G_3) \subseteq G_4$

Simple Entailment - Examples 2/4[4]

G_7 : non-lean

X `foaf:knows ex:bob.`

X `foaf:knows` Y`.`

G_8 : lean

X `foaf:knows ex:bob.`

X `foaf:knows` X`.`

G'_7 : lean

X `foaf:knows ex:bob.`

G'_8 : lean

X `foaf:knows` X`.`

[3] since G'_7 is a subgraph that is a proper instance entailing the whole graph

[4] draw on whiteboard

Simple Entailment - Examples 2/4[4]

G_7 : non-lean
X `foaf:knows ex:bob.`
X `foaf:knows` Y`.`

G_8 : lean
X `foaf:knows ex:bob.`
X `foaf:knows` X`.`

G'_7 : lean
X `foaf:knows ex:bob.`

G'_8 : lean
X `foaf:knows` X`.`

$G_7 \not\models G_8$, $G_7 \not\models G'_8$

[3] since G'_7 is a subgraph that is a proper instance entailing the whole graph
[4] draw on whiteboard

Simple Entailment - Examples 2/4[4]

G_7 : non-lean
X `foaf:knows ex:bob.`
X `foaf:knows` Y`.`

G_8 : lean
X `foaf:knows ex:bob.`
X `foaf:knows` X`.`

G_7' : lean
X `foaf:knows ex:bob.`

G_8' : lean
X `foaf:knows` X`.`

$G_7 \not\models G_8$, $G_7 \not\models G_8'$
$G_8 \models G_7$, $G_7 \models G_7'$

[3]since G_7' is a subgraph that is a proper instance entailing the whole graph
[4]draw on whiteboard

Simple Entailment - Examples 2/4[4]

G_7 : non-lean

X foaf:knows ex:bob.

X foaf:knows Y.

G_8 : lean

X foaf:knows ex:bob.

X foaf:knows X.

G'_7 : lean

X foaf:knows ex:bob.

G'_8 : lean

X foaf:knows X.

$G_7 \not\models G_8$, $G_7 \not\models G'_8$

$G_8 \models G_7$, $G_7 \models G'_7$

Finally: $G'_7 \models G_7$!!!! that confirms non-leanness![3]

[3]since G'_7 is a subgraph that is a proper instance entailing the whole graph

[4]draw on whiteboard

Simple Entailment - Examples 3/4

Now what about G_2?

G_1 :

```
ex:alice foaf:knows ex:bob.
ex:alice foaf:name "Alice".
foaf:knows rdfs:domain foaf:Person.
```

G_2 :

```
ex:alice rdf:type foaf:Person.
```

Obviously, no simple entailment: $G_1 \not\models G_2$!

Simple Entailment - Examples 3/4

Now what about G_2?

G_1 :

```
ex:alice foaf:knows ex:bob.
ex:alice foaf:name "Alice".
foaf:knows rdfs:domain foaf:Person.
```

G_2 :

```
ex:alice rdf:type foaf:Person.
```

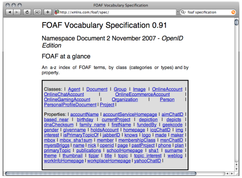
Obviously, no simple entailment: $G_1 \not\models G_2$!

Would need "special" interpretation of the `rdf:` and `rdfs:` vocabulary!

This is needed to interpret *ontologies*. . .

Recall from before – The FOAF ontology:

- **Properties:** `foaf:name`, `foaf:knows`, `foafhomepage`, `foaf:primaryTopic` etc.
- **Classes:** `foaf:Person`, `foaf:Agent`, `foaf:Document`, `foaf:Organisation`, etc.
- **Relations:** e.g.
  - *`Each Person is a Agent`* (subclass)



FOAF Vocabulary Specification 0.91

Namespace Document 2 November 2007 - *OpenID Edition*

FOAF at a glance

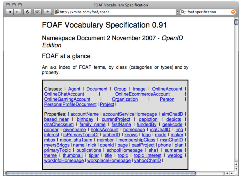
An a-z index of FOAF terms, by class (categories or types) and by property.

Classes: | Agent | Document | Group | Image | OnlineAccount | OnlineChatAccount | OnlineEcommerceAccount | OnlineGamingAccount | Organization | Person | PersonalProfileDocument | Project |

Properties: | accountName | accountServiceHomepage | aimChatID | based_near | birthday | currentProject | depiction | depicts | dnaChecksum | family_name | firstName | fundedBy | geekcode | gender | givenname | holdsAccount | homepage | icqChatID | img | interest | isPrimaryTopicOf | jabberID | knows | logo | made | maker | mbox | mbox_sha1sum | member | membershipClass | msnChatID | myersBriggs | name | nick | openid | page | pastProject | phone | plan | primaryTopic | publications | schoolHomepage | sha1 | surname | theme | thumbnail | tipjar | title | topic | topic_interest | weblog | workInfoHomepage | workplaceHomepage | yahooChatID |

Recall from before – The FOAF ontology:

- **Properties:** `foaf:name`, `foaf:knows`, `foafhomepage`, `foaf:primaryTopic` etc.
- **Classes:** `foaf:Person`, `foaf:Agent`, `foaf:Document`, `foaf:Organisation`, etc.
- **Relations:** e.g.
  - *Each `Person` is a `Agent`* (subclass)
  - *The `img` property is more specific than `depiction`* (subproperty)



FOAF Vocabulary Specification 0.91

Namespace Document 2 November 2007 - *OpenID Edition*

FOAF at a glance

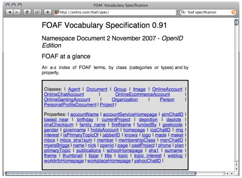
An a-z index of FOAF terms, by class (categories or types) and by property.

Classes: | Agent | Document | Group | Image | OnlineAccount | OnlineChatAccount | OnlineEcommerceAccount | OnlineGamingAccount | Organization | Person | PersonalProfileDocument | Project |

Properties: | accountName | accountServiceHomepage | aimChatID | based_near | birthday | currentProject | depiction | depicts | dnaChecksum | family_name | firstName | fundedBy | geekcode | gender | givenname | holdsAccount | homepage | icqChatID | img | interest | isPrimaryTopicOf | jabberID | knows | logo | made | maker | mbox | mbox_sha1sum | member | membershipClass | msnChatID | myersBriggs | name | nick | openid | page | pastProject | phone | plan | primaryTopic | publications | schoolHomepage | sha1 | surname | theme | thumbnail | tipjar | title | topic | topic_interest | weblog | workInfoHomepage | workplaceHomepage | yahooChatID |

Recall from before – The FOAF ontology:

- **Properties:** `foaf:name`, `foaf:knows`, `foafhomepage`, `foaf:primaryTopic` etc.
- **Classes:** `foaf:Person`, `foaf:Agent`, `foaf:Document`, `foaf:Organisation`, etc.
- **Relations:** e.g.
  - *Each `Person` is a `Agent`* (subclass)
  - *The `img` property is more specific than `depiction`* (subproperty)
  - *`img` is a relation between `Persons` and `Imgages`* (domain/range)



FOAF Vocabulary Specification 0.91

Namespace Document 2 November 2007 - *OpenID Edition*

FOAF at a glance

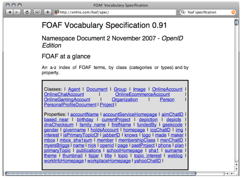
An a-z index of FOAF terms, by class (categories or types) and by property.

Classes: | Agent | Document | Group | Image | OnlineAccount | OnlineChatAccount | OnlineEcommerceAccount | OnlineGamingAccount | Organization | Person | PersonalProfileDocument | Project |

Properties: | accountName | accountServiceHomepage | aimChatID | based_near | birthday | currentProject | depiction | depicts | dnaChecksum | family_name | firstName | fundedBy | geekcode | gender | givenname | holdsAccount | homepage | icqChatID | img | interest | isPrimaryTopicOf | jabberID | knows | logo | made | maker | mbox | mbox_sha1sum | member | membershipClass | msnChatID | myersBriggs | name | nick | openid | page | pastProject | phone | plan | primaryTopic | publications | schoolHomepage | sha1 | surname | theme | thumbnail | tipjar | title | topic | topic_interest | weblog | workInfoHomepage | workplaceHomepage | yahooChatID |

Recall from before – The FOAF ontology:

- **Properties:** `foaf:name`, `foaf:knows`, `foafhomepage`, `foaf:primaryTopic` etc.
- **Classes:** `foaf:Person`, `foaf:Agent`, `foaf:Document`, `foaf:Organisation`, etc.
- **Relations:** e.g.
  - *Each `Person` is a `Agent`* (subclass)
  - *The `img` property is more specific than `depiction`* (subproperty)
  - *`img` is a relation between `Persons` and `Imgages`* (domain/range)
  - *`knows` is a relation between two `Persons`* (domain/range)



FOAF Vocabulary Specification 0.91

Namespace Document 2 November 2007 - *OpenID Edition*

FOAF at a glance

An a-z index of FOAF terms, by class (categories or types) and by property.

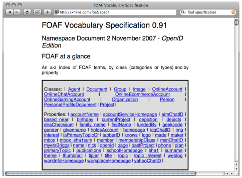
Classes: | Agent | Document | Group | Image | OnlineAccount | OnlineChatAccount | OnlineEcommerceAccount | OnlineGamingAccount | Organization | Person | PersonalProfileDocument | Project |

Properties: | accountName | accountServiceHomepage | aimChatID | based_near | birthday | currentProject | depiction | depicts | dnaChecksum | family_name | firstName | fundedBy | geekcode | gender | givenname | holdsAccount | homepage | icqChatID | img | interest | isPrimaryTopicOf | jabberID | knows | logo | made | maker | mbox | mbox_sha1sum | member | membershipClass | msnChatID | myersBriggs | name | nick | openid | page | pastProject | phone | plan | primaryTopic | publications | schoolHomepage | sha1 | surname | theme | thumbnail | tipjar | title | topic | topic_interest | weblog | workInfoHomepage | workplaceHomepage | yahooChatID |

Recall from before – The FOAF ontology:

- **Properties:** `foaf:name`, `foaf:knows`, `foafhomepage`, `foaf:primaryTopic` etc.
- **Classes:** `foaf:Person`, `foaf:Agent`, `foaf:Document`, `foaf:Organisation`, etc.
- **Relations:** e.g.
  - *Each `Person` is a `Agent`* (subclass)
  - *The `img` property is more specific than `depiction`* (subproperty)
  - *`img` is a relation between `Persons` and `Imgages`* (domain/range)
  - *`knows` is a relation between two `Persons`* (domain/range)
  - *`homepage` denotes **unique** homepage of an `Agent`* (uniquely identifying property)

⋮



FOAF Vocabulary Specification 0.91

Namespace Document 2 November 2007 - *OpenID Edition*

FOAF at a glance

An a-z index of FOAF terms, by class (categories or types) and by property.

Classes: | Agent | Document | Group | Image | OnlineAccount | OnlineChatAccount | OnlineEcommerceAccount | OnlineGamingAccount | Organization | Person | PersonalProfileDocument | Project |

Properties: | accountName | accountServiceHomepage | aimChatID | based_near | birthday | currentProject | depiction | depicts | dnaChecksum | family_name | firstName | fundedBy | geekcode | gender | givenname | holdsAccount | homepage | icqChatID | img | interest | isPrimaryTopicOf | jabberID | knows | logo | made | maker | mbox | mbox_sha1sum | member | membershipClass | msnChatID | myersBriggs | name | nick | openid | page | pastProject | phone | plan | primaryTopic | publications | schoolHomepage | sha1 | surname | theme | thumbnail | tipjar | title | topic | topic_interest | weblog | workInfoHomepage | workplaceHomepage | yahooChatID |

Simple Entailment - Examples 4/4

G'_1 :

```
ex:alice foaf:knows ex:bob.
ex:alice foaf:name "Alice".  ex:alice ex:age "30.0"^^xs:decimal.
```

G_{FOAF}: `<http://xmlns.com/foaf/0.1/>`

```
foaf:knows rdfs:domain foaf:Person.
foaf:knows rdfs:range foaf:Person.
foaf:Person rdfs:subclassOf foaf:Agent.
```

Simple Entailment - Examples 4/4

G'_1 :

```
ex:alice foaf:knows ex:bob.
ex:alice foaf:name "Alice".  ex:alice ex:age "30.0"^^xs:decimal.
```

G_{FOAF}: <http://xmlns.com/foaf/0.1/>

```
foaf:knows rdfs:domain foaf:Person.
foaf:knows rdfs:range foaf:Person.
foaf:Person rdfs:subclassOf foaf:Agent.
```

Intuitively, $G'_1 \uplus G_{FOAF}$ should entail: G'_2 :

```
ex:alice rdf:type foaf:Person.
ex:bob rdf:type foaf:Person.
ex:alice rdf:type foaf:Agent.
ex:bob rdf:type foaf:Agent.
ex:alice ex:age "30"^^xs:integer
```

Simple Entailment - Examples 4/4

G'_1 :

```
ex:alice foaf:knows ex:bob.
ex:alice foaf:name "Alice".  ex:alice ex:age "30.0"^^xs:decimal.
```

G_{FOAF}: <http://xmlns.com/foaf/0.1/>

```
foaf:knows rdfs:domain foaf:Person.
foaf:knows rdfs:range foaf:Person.
foaf:Person rdfs:subclassOf foaf:Agent.
```

Intuitively, $G'_1 \uplus G_{FOAF}$ should entail: G'_2 :

```
ex:alice rdf:type foaf:Person.    ... because the domain of knows is Person
ex:bob rdf:type foaf:Person.
ex:alice rdf:type foaf:Agent.
ex:bob rdf:type foaf:Agent.
ex:alice ex:age "30"^^xs:integer
```

The RDF semantics specification[Hayes, 2004] defines three refinements of simple interpretations and entailment relations which cover these entailments![Hayes, 2004]...

Simple Entailment - Examples 4/4

G'_1 :

```
ex:alice foaf:knows ex:bob.
ex:alice foaf:name "Alice".  ex:alice ex:age "30.0"^^xs:decimal.
```

G_{FOAF}: <http://xmlns.com/foaf/0.1/>

```
foaf:knows rdfs:domain foaf:Person.
foaf:knows rdfs:range foaf:Person.
foaf:Person rdfs:subclassOf foaf:Agent.
```

Intuitively, $G'_1 \uplus G_{FOAF}$ should entail: G'_2 :

```
ex:alice rdf:type foaf:Person.     ... because the domain of knows is Person
ex:bob rdf:type foaf:Person.       ... because the range of knows is Person
ex:alice rdf:type foaf:Agent.
ex:bob rdf:type foaf:Agent.
ex:alice ex:age "30"^^xs:integer
```

The RDF semantics specification[Hayes, 2004] defines three refinements of simple interpretations and entailment relations which cover these entailments![Hayes, 2004]...

Simple Entailment - Examples 4/4

G'_1 :

```
ex:alice foaf:knows ex:bob.
ex:alice foaf:name "Alice".  ex:alice ex:age "30.0"^^xs:decimal.
```

G_{FOAF}: `<http://xmlns.com/foaf/0.1/>`

```
foaf:knows rdfs:domain foaf:Person.
foaf:knows rdfs:range foaf:Person.
foaf:Person rdfs:subclassOf foaf:Agent.
```

Intuitively, $G'_1 \uplus G_{FOAF}$ should entail: G'_2 :

```
ex:alice rdf:type foaf:Person.    ... because the domain of knows is Person
ex:bob rdf:type foaf:Person.      ... because the range of knows is Person
ex:alice rdf:type foaf:Agent.     ... because each Person is an Agent
ex:bob rdf:type foaf:Agent.       ... because each Person is an Agent
ex:alice ex:age "30"^^xs:integer
```

The RDF semantics specification[Hayes, 2004] defines three refinements of simple interpretations and entailment relations which cover these entailments![Hayes, 2004]...

Simple Entailment - Examples 4/4

G'_1 :

```
ex:alice foaf:knows ex:bob.
ex:alice foaf:name "Alice".  ex:alice ex:age "30.0"^^xs:decimal.
```

G_{FOAF}: <http://xmlns.com/foaf/0.1/>

```
foaf:knows rdfs:domain foaf:Person.
foaf:knows rdfs:range foaf:Person.
foaf:Person rdfs:subclassOf foaf:Agent.
```

Intuitively, $G'_1 \uplus G_{FOAF}$ should entail: G'_2 :

`ex:alice rdf:type foaf:Person.`	... because the domain of `knows` is `Person`
`ex:bob rdf:type foaf:Person.`	... because the range of `knows` is `Person`
`ex:alice rdf:type foaf:Agent.`	... because each `Person` is an `Agent`
`ex:bob rdf:type foaf:Agent.`	... because each `Person` is an `Agent`
`ex:alice ex:age "30"^^xs:integer`	... simply because each $30.0 = 30$

The RDF semantics specification[Hayes, 2004] defines three refinements of simple interpretations and entailment relations which cover these entailments![Hayes, 2004]...

RDF Entailment regimes beyond simple Entailment

The RDF semantics specification[Hayes, 2004] defines three refinements of simple interpretations and entailment relations which cover these entailments!:

- RDF-entailment: Interpreting the `rdf:` vocabulary

RDF Entailment regimes beyond simple Entailment

The RDF semantics specification[Hayes, 2004] defines three refinements of simple interpretations and entailment relations which cover these entailments!:

- RDF-entailment: Interpreting the `rdf:` vocabulary
  - e.g. imposes that $\{$`s p o .`$\} \models$ `p rdf:type rdf:Property`

RDF Entailment regimes beyond simple Entailment

The RDF semantics specification[Hayes, 2004] defines three refinements of simple interpretations and entailment relations which cover these entailments!:

- RDF-entailment: Interpreting the `rdf:` vocabulary
  - e.g. imposes that $\{$`s p o .`$\} \models$ `p rdf:type rdf:Property`
- RDFS-entailment: Interpreting the `rdfs:` vocabulary

RDF Entailment regimes beyond simple Entailment

The RDF semantics specification[Hayes, 2004] defines three refinements of simple interpretations and entailment relations which cover these entailments!:

- RDF-entailment: Interpreting the `rdf:` vocabulary
  - e.g. imposes that {`s p o .`} $\models$ `p rdf:type rdf:Property`
- RDFS-entailment: Interpreting the `rdfs:` vocabulary
  - e.g. imposes that $G'_1 \uplus G_{FOAF} \models$ { `ex:alice rdf:type foaf:Person.`}

RDF Entailment regimes beyond simple Entailment

The RDF semantics specification[Hayes, 2004] defines three refinements of simple interpretations and entailment relations which cover these entailments!:

- RDF-entailment: Interpreting the `rdf:` vocabulary
  - e.g. imposes that $\{$`s p o .`$\} \models$ `p rdf:type rdf:Property`
- RDFS-entailment: Interpreting the `rdfs:` vocabulary
  - e.g. imposes that $G'_1 \uplus G_{FOAF} \models \{$ `ex:alice rdf:type foaf:Person.`$\}$
- D-entailment: Interpreting datatypes

RDF Entailment regimes beyond simple Entailment

The RDF semantics specification[Hayes, 2004] defines three refinements of simple interpretations and entailment relations which cover these entailments!:

- RDF-entailment: Interpreting the `rdf:` vocabulary
  - e.g. imposes that {`s p o .`} $\models$ `p rdf:type rdf:Property`
- RDFS-entailment: Interpreting the `rdfs:` vocabulary
  - e.g. imposes that $G_1' \uplus G_{FOAF} \models$ { `ex:alice rdf:type foaf:Person.`}
- D-entailment: Interpreting datatypes
  - e.g. imposing that in all interpretations that `"1"^^xs:integer` is interpreted the same as `"1.0"^^xs:decimal`

Unit Outline

1. What does that data mean? Ontologies described in RDFS + OWL

2. RDF Graph – Formal Definitions

3. RDF Interpretations and Simple Entailment

4. APPENDIX: Simple RDF Entailment is NP-complete

Simple RDF Entailment is NP-complete: Membership

Recall, we had that before already: We can test entailment $G \models ?G'$ by

1. guessing a mapping μ and
2. test whether $\mu(G') \subseteq G$ (this is obviously polynomial)

Membership in NP - done

Simple RDF Entailment is NP-complete: Hardness

To proof hardness we have to reduce another NP-hard problem to RDF entailment (in polynomial time). Let's "adapt" the proof from [Chandra and Merlin, 1977].

Simple RDF Entailment is NP-complete: Hardness

To proof hardness we have to reduce another NP-hard problem to RDF entailment (in polynomial time). Let's "adapt" the proof from [Chandra and Merlin, 1977].

3-colorability: Given an undirected Graph Gr, can all nodes be colored with 3 colors red, green, blue without two adjacent nodes having the same color?

Simple RDF Entailment is NP-complete: Hardness

To proof hardness we have to reduce another NP-hard problem to RDF entailment (in polynomial time). Let's "adapt" the proof from [Chandra and Merlin, 1977].

3-colorability: Given an undirected Graph Gr, can all nodes be colored with 3 colors red, green, blue without two adjacent nodes having the same color?

Reduction (the "trick" is we have to convert an undirected to a directed RDF graph):

- Graph G_1: simply encodes all "allowed" edges:
```
:red :edge :green.  :green :edge :red.
:green :edge :blue.  :blue :edge :green.
:blue :edge :red.  :red :redge :blue.
```
- Graph G_2: for each $(node_1, node_2) \in Gr$ we add two triples:
```
_:n1 :edge _:n2.  _:n2 :edge _:n1.
```
  to the graph G_2, i.e, we model the nodes as blank nodes.

Simple RDF Entailment is NP-complete: Hardness

To proof hardness we have to reduce another NP-hard problem to RDF entailment (in polynomial time). Let's "adapt" the proof from [Chandra and Merlin, 1977].

3-colorability: Given an undirected Graph Gr, can all nodes be colored with 3 colors red, green, blue without two adjacent nodes having the same color?

Reduction (the "trick" is we have to convert an undirected to a directed RDF graph):

- Graph G_1: simply encodes all "allowed" edges:
```
:red :edge :green.   :green :edge :red.
:green :edge :blue.   :blue :edge :green.
:blue :edge :red.   :red :redge :blue.
```
- Graph G_2: for each $(node_1, node_2) \in Gr$ we add two triples:
```
_:n1 :edge _:n2.   _:n2 :edge _:n1.
```
  to the graph G_2, i.e, we model the nodes as blank nodes.

Now, it is easy to see that:

Proposition

Gr is 3-colorably if and only if $G_1 \models G_2$

Recommended Reading

- [Gutiérrez *et al.*, 2004], excellent article on the logical foundations of RDF
- [de Bruijn *et al.*, 2005], relating RDF entailment to normal first-order logic.

A bit more tough reading (specs), but also recommended:

- [Hayes, 2004, Sections 1–2], official RDF semantics specification.
- [Mallea *et al.*, 2011] . . . all you ever wanted to know about blank nodes and never dared to ask.

Dan Brickley and R.V. Guha (eds.).
RDF vocabulary description language 1.0: RDF Schema, February 2004.
W3C Recommendation, available at `http://www.w3.org/TR/rdf-schema/`.

A. K. Chandra and P. M. Merlin.
Optimal Implementation of Conjunctive Queries in Relational Data Bases.
In *Conference Record of the Ninth Annual ACM Symposium on Theory of Computing*, pages 77–90, 1977.

Jos de Bruijn, Enrico Franconi, and Sergio Tessaris.
Logical reconstruction of normative RDF.
In *OWL: Experiences and Directions Workshop (OWLED-2005)*, Galway, Ireland, November 2005.

Claudio Gutiérrez, Carlos A. Hurtado, and Alberto O. Mendelzon.
Foundations of semantic web databases.
In *PODS*, pages 95–106, 2004.

P. Hayes.
RDF semantics, 2004.
`http://www.w3.org/TR/rdf-mt/`.

Alejandro Mallea, Marcelo Arenas, Aidan Hogan, and Axel Polleres.
On Blank Nodes.
In *Proceedings of the 10th International Semantic Web Conference (ISWC 2011)*, volume 7031 of *Lecture Notes in Computer Science* (*LNCS*), Bonn, Germany, October 2011. Springer.

Nominated for best paper award.

📕 Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks.
OWL Web Ontology Language Semantics and Abstract Syntax, February 2004.
W3C Recommendation.