Outline

1. Motivation: Graph Data Model
2. Syntax and Serialization Format
3. Datatypes
4. Multi-valued/n-ary relationships
5. Blank nodes
6. Dataset
Acknowledgements

• Most of the slides in this presentation are adapted from:
1. Motivation: Graph Data Model
2. Syntax and Serialization Format
3. Datatypes
4. Multi-valued/n-ary relationships
5. Blank nodes
6. Dataset
Recap: From Net to Web to Graph

• “It’s not the wires – it’s the computers!”
• “It’s not the computers – it’s the documents!”
• “It’s not the documents – it’s the things!”

RDF in the SW Stack

https://www.w3.org/2007/03/layerCake.png
Encode the following sentence in XML:
“*The book ‘Foundations of Semantic Web Technologies’ was published by CRC Press*”

```xml
<publication>
  <publisher>CRC Press</publisher>
</publication>

<publisher name="CRC Press">
  <publication book="Foundations of Semantic Web Technologies"/>
</publisher>

<book>
  <title>Foundations of Semantic Web Technologies</title>
  <publisher>CRC Press</publisher>
</book>

And many more alternatives ....
<publication>
  <publisher>CRC Press</publisher>
</publication>
<book>
  <title>Foundations of Semantic Web Technologies</title>
  <publisher>CRC Press</publisher>
</book>
• Earlier XML trees essentially represent the same relationship.
• Combining trees are cumbersome
  – and the result isn’t always clear (may not even be a tree!).
• Solution: use (directed) graph model.
• And since we’re on the Web, we use URI, instead of strings, to represent entities.

http://example.org/fost

http://example.org/publishedBy

http://crcpress.com/uri
• RDF = Resource Description Framework
  – W3C Recommendation 2004 (RDF 1.0)
  – W3C Recommendation 2014 (RDF 1.1)

• RDF is a data model:
  – Initially intended for describing metadata of Web resources, but found more general use later.
  – Represents structured information
  – A universal, machine readable data exchange format (with several types of standard serialization).
1. Motivation: Graph Data Model
2. Syntax and Serialization Format
3. Datatypes
4. Multi-valued/n-ary relationships
5. Blank nodes
6. Dataset
• Resource/entity: anything in the world
  – physical things, documents, abstract concepts, numbers, strings, ...
• RDF components correspond to resources:
  – *IRI/URI*: identifier of a resource
  – *Literal*: data value
  – *Blank node*: denote a resource without giving it a name (i.e., the resource exists but doesn’t have a name or the name is unknown)
• Graph model enables simple composition of distributed data
  – True w.r.t. the structure, but not necessarily w.r.t. the content
• Problem 1: the same resource may be labeled with two different identifiers.
  – e.g., no globally agreed identifier for the book “Foundations of Semantic Web Technologies”
• Problem 2: the same identifier may be used for two different resources.
  – e.g., “CRC” may refer to the publishing house, or the Cincinnati Recreation Commission
• Solution: XML and RDF use URI
• URI = Uniform Resource Identifier
• String of characters for identifying a resource.
• Specified in RFC 3986
• Generalized from Uniform Resource Locator (URL), i.e., Web address.
  – Every URL is a URI.
  – Some URIs are not URL – sometimes called Uniform Resource Name (URN).
• Now generalized to Internationalized Resource Identifier (IRI) – specified in RFC 3987 – by allowing non-ASCII characters.
• A single URI cannot refer to two distinct objects
  – If a URI occurs in two distinct data sources, it always refer to the same resource.
scheme : [//authority] path [?query][#fragment]

- Scheme (mandatory): classifies type of URI
  - http, ftp, mailto, file, irc
- Authority (optional): typically domain name, possibly with user and port details
  - example.org:8080, google.com, john@example.com
- Path (mandatory): main part of URI, possibly empty, e.g., in email addresses. May be organized as hierarchy with slash ('/') separator.
  - /, /etc/passwd/, ~/../krisnadhi
  - paths must have initial /, unless no authority is given
- Query (optional): preceded by ?, provides additional non-hierarchical information.
- Fragment (optional): preceded by #, provides second level of identifying resources.
- All parts of URI, except scheme and authority/host, are case-sensitive, unless specified otherwise by the scheme.
  - There are detailed exceptions (read the RFC!)
Literals

- for representing data value
- denoted as strings
- may optionally be suffixed with either a datatype URI or language tag (but not both)
- interpreted by associated datatype
- literals without an associated datatype or language tag are treated as strings

http://example.org/fost

http://example.org/publishedBy

http://example.org/title

“Foundations of Semantic Web Technologies”

http://example.org/name

“CRC Press”

http://crcpress.com/uri
• Discussed later 😊
• Graph can be represented in different ways.
• RDF chooses triples, i.e., an RDF graph is a set of (node-edge-node) triples.
• How many triples are there below?
• Permitted component:
  – subject: URI or blank node
  – predicate: URI
  – object: URI, blank node, or literal
• Node and edge labels are unique, so the graph can always be reconstructed from the set of triples.
RDF serialization format: concrete syntax for writing RDF files or transferring RDF data over network.

- N-Triples
- Turtle
- RDF/XML [The only serialization format of RDF 1.0]
- Trig
- JSON-LD
- N-Quads
- RDFa

RDF Document: a document (e.g., a file) that encodes an RDF graph or RDF dataset in a particular concrete syntax.
N-Triples

- Simple enumeration of triples; suitable for line-based parsing.
- A simplified version of Tim Berners-Lee’s Notation 3 (N3).
- N-Triples document: each line is either a triple or a comment.
- A triple: a sequence of (subject, predicate, object) terms, separated by whitespace and terminated with a full stop (‘.’)
- A comment: parts of a line beginning with a pound sign (‘#’) until the end of line, provided the pound sign occurs outside any URI or literal.
- URIs: enclosed with ‘<‘ and ‘>’
- Blank nodes: prefixed with underscore and colon (‘_:’)
- Literals: enclosed with double quotes; may optionally be suffixed with a datatype indicator (double carets followed by datatype URI) or a language tag (ampersand followed by a language codestring).

```
# comment here
```
every N-Triples document is a Turtle document
Triples end with a full stop. Groupings are possible (see example).
Whitespaces outside URIs and literals are only as token separator. Newlines are not important, unlike N-Triples.

URIs:
- Full URIs: enclosed by ‘<‘ and ‘>’
- Abbreviated URIs: use a previously declared namespace prefix (no enclosing angled brackets).

Literals:
- Quoted literals (like N-Triples, but datatype URI may be abbreviated)
- Numeric (integer, decimal, floating point) literals, e.g., 5 is shorthand for “5”^^xsd:integer
- Boolean literals: ‘true’ or ‘false’ (without quotes, case sensitive).

Blank nodes:
- Prefixed with ‘_:’ like N-Triples, i.e., labeled blank nodes; or
- Use square brackets (see example later).
# comment here


Abbreviated URIs using prefix

@prefix ex: <http://example.org/> .
@prefix crc: <http://crcpress.com/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ex:fost ex:publishedBy crc:uri .
crc:uri ex:name “CRC Press”^^xsd:string .
Turtle

@prefix ex: <http://example.org/> .
@prefix crc: <http://crcpress.com/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ex:fost ex:publishedBy crc:uri .
crc:uri ex:name "CRC Press"^^xsd:string .
crc:uri ex:name "CRC" .

• Grouping triples with the same subject
• Grouping triples with the same subject and predicate.

@prefix ex: <http://example.org/> .
@prefix crc: <http://crcpress.com/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ex:fost ex:publishedBy crc:uri ;
crc:uri ex:name "CRC Press"^^xsd:string , "CRC" .
• Namespaces used to disambiguate tags (like in XML)
• RDF-specific tags have predefined namespace, by convention, abbreviated as `rdf`

```xml
<?xml version="1.0" encoding="utf-8"?>
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
    <rdf:Description rdf:about="http://example.org/fost">
    </rdf:Description>
    <rdf:Description rdf:about="http://example.org/fost">
    </rdf:Description>
    <rdf:Description rdf:about="http://crcpress.com/uri">
        <ex:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">CRC Press</ex:name>
    </rdf:Description>
    <rdf:Description rdf:about="http://crcpress.com/uri">
        <ex:name>CRC</ex:name>
    </rdf:Description>
</rdf:RDF>
```
RDF/XML

- rdf:Description element encodes the subject
- all direct children of the rdf:Description element encoding a subject are predicates
- predicate elements contain the object either through rdf:resource attribute or another rdf:Description element (see two examples below).

```xml
<rdf:Description rdf:about="http://example.org/fost">
</rdf:Description>
```

```xml
<rdf:Description rdf:about="http://example.org/fost">
  <ex:publishedBy>
    <rdf:Description rdf:about="http://crcpress.com/uri"/>
    <ex:publishedBy/>
  </ex:publishedBy>
</rdf:Description>
```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:ex="http://example.org/"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <rdf:Description rdf:about="http://example.org/fost">
        <ex:publishedBy>
            <rdf:Description rdf:about="http://crcpress.com/uri"/>
        </ex:publishedBy>
    </rdf:Description>
</rdf:RDF>
• Untyped literals can be included as a free text into the predicate element.
• Condensed form:
  – One subject with several predicate elements.
  – One object description serves as subject of another triple.

```xml
<rdf:Description rdf:about="http://example.org/fost">
  <ex:title>Foundations of Semantic Web Technologies</ex:title>
  <ex:publishedBy>
    <rdf:Description rdf:about="http://crcpress.com/uri"/>
    <ex:name>CRC Press</ex:name>
  </ex:publishedBy>
</rdf:Description>
```
• Alternative: literal as attribute of the subject with the corresponding predicate as the attribute name.
• Object URI as value of the attribute rdf:resource inside predicate tag.

```
<rdf:Description rdf:about="http://example.org/fost"
    ex:title="Foundations of Semantic Web Technologies">
</rdf:Description>
<rdf:Description rdf:about="http://crcpress.com/uri"/>
    <ex:name>CRC Press</ex:name>
</rdf:Description>
```
Namespace is essential in XML serialization, since colon (`:`) in XML attributes is not allowed unless used with a namespace.

Problem: namespace cannot be used in values of XML attributes, e.g., `rdf:about="ex:fost"` is **wrong** since `ex` would be interpreted as URI scheme.

Solution: use XML ENTITY

```xml
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:ex="http://example.org/" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!DOCTYPE rdf:RDF[ <!ENTITY ex 'http://example.org/' ]>

  <rdf:Description rdf:about="&ex;fost">
  </rdf:Description>
</rdf:RDF>
```
• Use of base namespace

```xml
<?xml version="1.0" encoding="utf-8"?>
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#" xmlns:base="http://example.org/">
  <rdf:Description rdf:about="fost">
    <ex:publishedBy rdf:resource="http://crcpress.com/uri" />
  </rdf:Description>
  <rdf:Description rdf:about="fost">
  </rdf:Description>
  <rdf:Description rdf:about="http://crcpress.com/uri">
    <ex:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">CRC Press</ex:name>
  </rdf:Description>
  <rdf:Description rdf:about="http://crcpress.com/uri">
    <ex:name>CRC</ex:name>
  </rdf:Description>
</rdf:RDF>
```
• Turtle is MUCH easier to read and write.
• Tool and programming support for XML are MUCH more widely available.
  – That’s why the normative syntax for RDF 1.0 was XML only.
• There are tools to convert different RDF serialization format.
Other serialization formats?

- Trig
- JSON-LD
- N-Quads
- RDFa

See W3C standard documents 😊
1. Motivation: Graph Data Model
2. Syntax and Serialization Format
3. Datatypes
4. Multi-valued/n-ary relationships
5. Blank nodes
6. Dataset
• Each datatype is identified by a URI.
  – http://www.w3.org/2001/XMLSchema#string is for string.
• Serialization: see previous examples where we use typed literal: “CRC Press”^^xsd:string.
• Many XML Schema datatypes are RDF-compatible.
• rdf:HTML datatype for HTML content as literal value
• rdf:XMLLiteral datatype for XML content as literal value
Datatypes

- Each datatype has a lexical space, a value space, and a lexical-to-value mapping, e.g.,
- Simple literals (no explicit type and language tag) belong to xsd:string
- Language-tagged literals belong to rdf:langString
Datatypes

**xsd:boolean**

- Lexical space: “true”, “false”, “1”, “0”
- Value space: true, false

**xsd:integer**

- Lexical space: “1”, “01”, “+1”, “-1”, “05”, ...
- Value space: 1, -1, 1, -1, 5, ...

- For xsd:string: “02” < “2”
- For xsd:integer: “02” = “2”
1. Motivation: Graph Data Model
2. Syntax and Serialization Format
3. Datatypes
4. Multi-valued/n-ary relationships
5. Blank nodes
6. Dataset
N-ary relationship

• “For preparation of Chutney, you need 1 lb green mango, a teaspoon Cayenne pepper, ...”

• What’s wrong with this modeling attempt? (Try visualizing it).

@prefix ex: <http://example.org/> .
ex:Chutney ex:hasIngredient "1 lb green mango",
"1 tsp. Cayenne pepper",
...
N-ary relationship

• How about this one? (Try visualizing it)

```prefix ex: <http://example.org/> .
ex:Chutney   ex:hasIngredient  ex:greenMango ;
ex:amount     "1 lb" ;
ex:hasIngredient  ex:CayennePepper ;
ex:amount       "1 tsp." ;
...```
N-ary relationship

• The problem: we have a proper ternary relationship (like in relational databases).

<table>
<thead>
<tr>
<th>dish</th>
<th>Ingredient</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>chutney</td>
<td>green mango</td>
<td>1 lb.</td>
</tr>
<tr>
<td>chutney</td>
<td>Cayenne pepper</td>
<td>1 tsp.</td>
</tr>
</tbody>
</table>

• Representation in RDF needs auxiliary nodes.
N-ary relationships

@prefix ex: <http://example.org/> .
ex:Chutney ex:hasIngredient ex:ingredient1 .
ex:ingredient1 ex:ingredient ex:greenMango ;
ex:amount "1 lb" .

http://example.org/Chutney

http://example.org/hasIngredient

http://example.org/ingredient1

http://example.org/ingredient

http://example.org/greenMango

http://example.org/amount

1 lb
1. Motivation: Graph Data Model
2. Syntax and Serialization Format
3. Datatypes
4. Multi-valued/n-ary relationships
5. Blank nodes
6. Dataset
Blank nodes

- Blank nodes (bnodes) can be used for nodes that need not be named.
- Can be read as an existential statement.
• RDF/XML syntax:

```xml
<rdf:Description rdf:about="http://example.org/Chutney">
    <ex:hasIngredient rdf:nodeID="id1" />
</rdf:Description>

<rdf:Description rdf:nodeID="id1">
    <ex:ingredient rdf:resource="http://example.org/greenMango"/>
    <ex:amount>1 lb</ex:amount>
</rdf:Description>
```

• abbreviated:

```xml
<rdf:Description rdf:about="http://example.org/Chutney">
    <ex:hasIngredient rdf:parseType="Resource">
        <ex:ingredient rdf:resource="http://example.org/greenMango"/>
        <ex:amount>1 lb</ex:amount>
    </ex:hasIngredient>
</rdf:Description>
```
Blank Nodes

• Turtle syntax:

```turtle
@prefix ex: <http://example.org/> .
ex:Chutney ex:hasIngredient _:id1 .
_:id1 ex:ingredient ex:greenMango ;
ex:amount "1 lb" .
```

• abbreviated (using square brackets):

```turtle
@prefix ex: <http://example.org/> .
ex:Chutney ex:hasIngredient
    [ ex:ingredient ex:greenMango ;
      ex:amount "1 lb" ] .
```
Blank Node Identifiers

- Blank node identifiers: local identifiers used in some concrete RDF syntax (i.e., serialization format) or RDF store implementations.
  - Thus, entirely dependent on the implementation of the serialization or RDF store.
  - e.g., in Turtle, we use ‘_:’ prefix to indicate such identifiers.
- Always locally scoped to the file or RDF store.
- NOT persistent and NOT portable.
- Need to be careful when dealing with multiple blank nodes, especially from different RDF data sources.
- During processing, we may need to Skolemize blank nodes: replacing them with globally unique URIs (called Skolem URI/IRI) that are not previously used.
Graph Isomorphism

- Often, we want to know if two RDF graphs are structurally “the same”
- We use the notion of isomorphism.
- Graph $G_1$ and $G_2$ are isomorphic (i.e., have identical form) if there is a mapping/function $M$ from $G_1$ to $G_2$ such that:
  - $M$ is a bijection
  - $M$ maps blank nodes in $G_1$ to blank nodes in $G_2$
  - $M(lit) = lit$, for each literal lit occurring as a node in $G_1$
  - $M(uri) = uri$, for each URI/IRI uri occurring as a node in $G_1$
  - The triple $(s, p, o)$ is in $G_1$ if and only if the triple $(M(s), p, M(o))$ is in $G_2$.

- Note: we use equality definition for literals and URI/IRIs defined in the RDF specs.
Outline

1. Motivation: Graph Data Model
2. Syntax and Serialization Format
3. Datatypes
4. Multi-valued/n-ary relationships
5. Blank nodes
6. Dataset
Datasets

• RDF Dataset = A collection of RDF graphs such that:
  – Exactly one graph is the default graph. The default graph does not have a name and may be empty.
  – Zero or more named graphs. The name of each named graph is either a URI or a blank node.
    • Graph names are unique within a dataset.

• Serialization of RDF datasets are supported by TriG, N-Quads, JSON-LD
  – There is no mention of the notion of RDF Dataset in the RDF 1.1 XML syntax specification.
Other RDF Features

- Container: open collection
- Collection: closed collection
- Reification
- Utility properties

We’ll discuss this along with RDF Schema.