Towards Ontologies

12.1 Reasoning with Ontologies
12.2 Semantics in RDF
12.3 SPARQL

RDF

- RDF (Resource Description Framework) is a markup language to encode knowledge
  - Knowledge is represented by triples
    - Triples are simple propositions consisting of subject, predicate, and object
    - Each triple part is either a literal or a resource
    - Each resources is given by an URI references
    - Each URI represents a given entity or concept
  - e.g.:
    - mdb:BruceWillis mdb:played_in mdb:DieHard

RDF/S

- RDF/S is used to define the vocabulary used in RDF graphs
  - Can define classes and properties, etc
  - Typically, an RDF graph only states propositions on instances of classes defined by the RDF/S it uses
  - RDF/S introduces already some predefined relationships (predicates) on classes and thus instances
    - Sub-classes, sub-properties, ranges, domains, etc…

12.1 Ontologies

- RDF statements represent formal knowledge in a certain domain and thus the abstracted reality
  - New information can be gathered by combining RDF statements in a suitable way
  - Example:
    - St. Bernard is a dog
    - St. Bernard makes sound "wuff"

- The network of relationships created by those statements is called an ontology
12.1 Ontologies

- An ontology is a formal representation of...
  - A set of concepts within a domain
  - And the relationships between those concepts
- It can be used...
  - To define a domain
  - And to reason about the properties of that domain
- Lies on top of the basic RDF representation layer of the Semantic Web stack

12.1 Early Ontologies

- Science and philosophy always strived to explain the world and the nature of being
  - First formal school of studies: Aristotle’s metaphysics (‘beyond the physical’, ca. 360 BC)
  - Traditional branches of metaphysics
    - Ontology (λόγος; word, science and ὄντος; of being)
      - Study of being and existence
    - Natural theology
      - Study of God, nature and creation
    - Universal science
      - “First Principles”, logic

12.1 Early Ontologies

- Ontology tries to describe everything which is (exists), and its relation and categorization with respect to other things in existence
  - What is existence? Which things exist?
  - Which entities are fundamental?
  - What is a physical object?
  - How do the properties of an object relate to the object itself? What features are the essence?
  - What does it mean when a physical object exists?
  - What constitutes the identity of an object?
  - When does an object go out of existence, as opposed to merely change?

12.1 Early Ontologies

- Taxonomies (τάξις: arrangement, νόμος: law) are part of ontology
  - Groups things with similar properties into taxa
  - Taxa are put into a hierarchical structure
    - Hierarchy represents supertype-subtype relationships
    - Represents a specialization of taxa, starting with the most general one
12.1 Early Ontologies

- Example: Linnaean Taxonomy
  - Classification of all living things by Carl von Linné in 1738
  - Classification into multiple hierarchy layers
    - Kingdom, Phylum, Subphylum, Class, Cohort, Order, Suborder, Infraorder, Superfamily, Family, Genus, Species
  - Each layer adds additional properties and restrictions

- Example: Red Squirrel
  (Binomial Name: Tamiasciurus hudsonicus)
  - Kingdom: Animals
  - Phylum: Chordata (with backbone)
  - Class: Mammalia (with backbone, nursing its young)
  - Order: Rodentia (backbone, nursing its young, sharp front teeth)
  - Suborder: Sciuromorpha (backbone, nursing its young, sharp front teeth, like squirrel)
  - Family: Sciuridae (backbone, nursing its young, sharp front teeth, like squirrel, bushy tail & lives on trees (i.e. real squirrel))
  - Genus: Tamiasciurus (backbone, nursing its young, sharp front teeth, like squirrel, bushy tail & trees, from N-America)
  - Species: Hudsonicus (backbone, nursing its young, sharp front teeth, like squirrel, bushy tail & trees, from N-America, brown fur with white belly)

- Example: Edible Dormouse
  (Binomial Name: Glis Glis)
  - Kingdom: Animals
  - Phylum: Chordata (with backbone)
  - Class: Mammalia (with backbone, nursing its young)
  - Order: Rodentia (backbone, nursing its young, sharp front teeth)
  - Suborder: Sciuromorpha (backbone, nursing its young, sharp front teeth, like squirrel)
  - Family: Gliridae (backbone, nursing its young, sharp front teeth, like squirrel, sleeps long)
  - Genus: Glis (backbone, nursing its young, sharp front teeth, bushy tail, like squirrel, eaten by Romans)
  - Species: Glis (backbone, nursing its young, sharp front teeth, bushy tail, climbs trees, nothing more to classify)

- In computer science ontologies are formal, explicit specifications of a shared conceptualization
  - Basically an ontology provides a shared vocabulary
    - It can be used to define the type of objects and/or concepts that exist and their properties and relations
  - Ontologies are often equated with taxonomic hierarchies of classes, class definitions, and the subsumption relation
    - But this definition is far too narrow
  - Domain ontologies model the real world with respect to a specific domain
    - This also disambiguates most terms
    - But domain ontologies are not compatible with each other
12.1 Upper Ontologies

- An upper ontology is a model of the common objects that are generally applicable across a wide range of domain ontologies
  - It contains a core glossary in whose terms objects in a set of domains can be described
  - There are several standardized upper ontologies available for use, including Dublin Core, GFO, OpenCyc, SUMO, and DOLCE

12.1 Upper Ontologies

- Simple Dublin Core Metadata Element Set (DCMES) defines 15 elements
  1. Title
  2. Creator
  3. Subject
  4. Description
  5. Publisher
  6. Contributor
  7. Date
  8. Type
  9. Format

12.1 Upper Ontologies

- Qualified Dublin Core defines three additional elements
  1. Audience
  2. Provenance
  3. RightsHolder
  - Elements refinements make the meaning of an element narrower or more specific
  - Dumb-Down Principle
    - If an application does not understand a specific element refinement term, it should be able to ignore the qualifier and treat the metadata value as if it were unqualified

12.2 RDF-Semantics

- There are several ways to capture ontologies
  - RDF-S is able to model formal vocabularies
    - Defines available constants and predicate symbols
  - Using those vocabularies, RDF allows to express statements
    - Triples: Subject, Predicate, Object
  - Basically the syntax of RDF with RDF-S vocabularies can be seen as a language for expressing propositions (or assertions)
12.2 RDF-Semantics

- For capturing the semantics of RDF, two approaches can be chosen
  - **Model theoretic approach:** Directly define a model semantic for RDF
  - **Translation approach:** Translate RDF into logics
  - We will showcase and mix both approaches briefly

12.2 RDF-Semantics

- A graph may have **blank nodes** or **blank edges**
  - A graph without any blanks is called **ground graph**
  - To identify blank nodes, we give them domain-free names
    * e.g.: `?x` or `?y` (so they look like variables)
- All **literals** or **URIs** are summarized as names
  - Names can be considered as constant values in logics
  - Thus, names don’t have any inherited semantics
  - Also, URIs don’t have semantics
    * Although the URL might be interpretable—but still, this is no formal semantics, e.g. http://www.moviedb.com/movie/dieHard

12.2 RDF-Semantics

- **A proper instance** is a graph where a blank node is mapped to either a name, or two nodes mapped into the same node

12.2 RDF-Semantics

- **Nomenclature of RDF syntax**
  - **RDF Graph**
    * Set of RDF triples
  - **RDF Subgraph**
    * Subset of the graph triples
    * A proper subgraph is a graph of a proper subset of the triples

12.2 RDF-Semantics

- All names of a graph are called its **vocabulary**
- Given a graph $G$, and a **mapping $M$** of blank nodes to some names or nodes:
  - Any graph obtained by replacing some blank nodes $N$ with $M(N)$ is called an **instance of $G$**

12.2 RDF-Semantics

- A graph is **equivalent**, if a blank node is mapped to a new blank node which was originally not in the graph
  - An equivalent graph has just its “variables” renamed
- A graph is **lean**, if it does not show internal redundancies, e.g. has no instance which is a proper subgraph of the graph
12.2 RDF-Semantics

• Furthermore, graphs may be **merged**
  - If two graphs have no common blank nodes, the merge is just the graph of the unioned triples
  - If they do have common blank nodes, the blank nodes of one are mapped to new blank nodes such that both graph have no blanks in common

Based on this, we can **classify, merge and map** graphs with respect to each other
- Those operations are needed when mapping RDF to a logical representation
- RDF graphs express logical facts which, of course, can be mapped to **logic**
- Thus, we may use known methods for interpretation and reasoning

• Triples can be translated into an **logic formula**

Intuition:

\[
\text{rdf:type}(\text{mdb:dieHard}, \text{mdb:movie}) \land \text{mdb:released_in}(\text{mdb:dieHard}, 1988)
\]

• Thus, we can build a formal language \( \mathcal{L} = (\Gamma, \Omega, \Pi, \chi) \) for a given RDF graph
  - **Predicate** symbols \( \Pi \) are given by all names used as predicates in the graph
  - Additionally, \( \Pi \) contains RDF's special names:
    - \text{rdfs:Type}: Indicates that a given name is of a given type
    - \text{rdfs:Resource}: Indicates that a certain name is an resource
    - \text{rdfs:Class}: Defines a class membership
    - \text{rdfs:SubClassOf}: Defines a subclass relation
    - \text{rdfs:Property}: Defines a property
    - \text{rdfs:range}: Defines the value range of a property
    - \text{rdfs:domain}: Defines the value domain of a property
    - etc.

  - **Function** symbols \( \Omega \) are given by any used RDF functions
  - **Variable** symbols \( \chi \) are given by the names of the blank nodes
  - **Constant** symbols \( \Gamma \) are given by all other names used in the graph
    - Constants can be separated into
      - Literals (Strings, Numerals, etc)
      - URIs

Thus, we can build a formal language \( \mathcal{L} = (\Gamma, \Omega, \Pi, \chi) \) for a given RDF graph
- **Predicate** symbols \( \Pi \) are given by all names used as predicates in the graph
- Additionally, \( \Pi \) contains RDF's special names:
  - \text{rdfs:Type}: Indicates that a given name is of a given type
  - \text{rdfs:Resource}: Indicates that a certain name is an resource
  - \text{rdfs:Class}: Defines a class membership
  - \text{rdfs:SubClassOf}: Defines a subclass relation
  - \text{rdfs:Property}: Defines a property
  - \text{rdfs:range}: Defines the value range of a property
  - \text{rdfs:domain}: Defines the value domain of a property
  - etc.

Based on this, we can **classify, merge and map** graphs with respect to each other
- Those operations are needed when mapping RDF to a logical representation
- RDF graphs express logical facts which, of course, can be mapped to **logic**
- Thus, we may use known methods for interpretation and reasoning

Triples can be translated into an **logic formula**

Intuition:

\[
\text{rdf:type}(\text{mdb:dieHard}, \text{mdb:movie}) \land \text{mdb:released_in}(\text{mdb:dieHard}, 1988)
\]
12.2 RDF-Semantics

- Overview

<table>
<thead>
<tr>
<th>Literals</th>
<th>URIs</th>
</tr>
</thead>
<tbody>
<tr>
<td>untyped</td>
<td>typed</td>
</tr>
</tbody>
</table>

Constant symbols

Vocabulary $V$

Constant symbols & Predicate Symbols

12.2 RDF-Semantics

- Besides the syntax, we will need a formal interpretation $I = (U, I_U, I_P, I_L)$
  - Unfortunately, this will be a little bit dirty...
  - The universe of discourse $U$ consisting of
    - $U_t$ all resources
      - Resources especially contain alpha-numeric string literals
    - $U_p$ all properties
      - Yes, predicates are part of the universe! (in contrast to “classic” logics)
  - The mapping $I_P$ of property URIs to properties
  - The mapping $I_L$ consisting of $I_{CL}, I_{CR}$
    - $I_{CL}$ mapping typed literals to resources
    - $I_{CR}$ mapping untyped literals to literal values
    - $I_{UR}$ mapping URIs to resources

12.2 RDF-Semantics

- All those mappings use Herbrand style interpretations
  - i.e. constants are just mapped to their syntactical representation, but special treatment for classes and propositions on memberships
- Additionally, we need an extension interpretation $I_{EXT}$ for the properties in $U_p$
  - The extension will provide the actual semantics of the property, e.g., in which cases the property evaluated to true and false
  - $I_{EXT} : U_p \rightarrow \{0,1\}$
  - Compares to Herbrand interpretation
    - Remember: Different Herbrand interpretations correspond to different subsets of the Herbrand base

12.2 RDF-Semantics

- Model theoretic observation: When is an interpretation $I$ a model of a RDF graph $G$?
  - $I$ is model of $G$, if it is a model of each triple in $G$
- When is an interpretation $I$ a model of a triple $t$?
  - $I$ is model of $t$, if it evaluates the triple to true
  - Each triple consists of a subject $s_t$, predicate $p_t$ and object $o_t$
  - Subject, predicate and object need to be element of the vocabulary
  - Furthermore, the predicate must hold i.e. $(I(s_t), I(o_t)) \in I_{EXT}(p_t)$

12.2 RDF-Semantics

- Blank nodes in $G$ need to be mapped to literal or URI nodes until $G$ is a ground graph
  - This step accords to a variable substitution in logical models
- Example:
  - $U_p = \{a, b, p, q, r, 1998\}$
  - $U_p = \{p, q, r\}$
  - $I_p = \{1998\}$
  - $I_L = \{ mdb:bruceWillis = a, mdb:action = b \}$
  - $I_L = \{ mdb:released_in = p, mdb:genre = q, mdb:played_in = r \}$
  - $I_{EXT} = \{ p := \{(c, b), r := \{(c, a)\}\} \}$
  - Is the graph a model?
12.2 RDF-Semantics

• Provide a node substitution $\text{?x} \mapsto c$
• Check for each triple if the interpretation is a model
  \[
  (I(?x), I(mdb:bruceWillis)) = (c,a) \in I_{\text{EXT}}(r)
  \]
  \[
  (I(?x), I(mdb:action)) = (c,b) \in I_{\text{EXT}}(q)
  \]
  \[
  (I(?x), I(1988)) = (c,1988) \in I_{\text{EXT}}(p)
  \]
• Yes... $I$ is a model for $G$

Orthogonal approach: Translate RDF to logic

– Logic representation has to retain special RDF features like properties and classes
  \* Modeled by special predicate names, e.g. rdf:Property or rdf:Class
– Note: We need to be able to express properties of predicates,
  \* e.g. the predicate mdb:Movie describes a class
    from mdb:Movie rdf:type rdfs:Class
  \* This functionality is usually provided by higher order logics!!!
  \* However, we only use features of second order logics for easy and well
    defined tasks (class membership, sub class relations, etc.), thus not the
    full power of second order logic is necessary.
    – Remember: General second order logics are undecidable…
– Furthermore, this approach will also work for all other kinds of
  knowledge representation
  \* e.g. DAML, OWL, OWL2, etc…

Following rules of thumb $TR[E]$ can be used to transform RDF expression $E$ into logic

– Any URI reference in subject or object
  \* \rightarrow \text{Constant Symbols or Predicate Symbol}
– Blank nodes
  \* \rightarrow \text{Variable Symbols}
– Any triple of the form $a \text{rdf:type} b$
  \* \rightarrow TR(b)(TR(a)) \land rdfs:Class(TR(b))

– Any other triple of the form $a \text{ } b \text{ } c$
  \* \rightarrow TR[b](TR[a],TR[c]) \land rdfs:Property(TR(b))
– An RDF graph
  \* \rightarrow \text{The existential closure of the conjunction of all}
    \text{translations of all triples in the graph}
– A set of RDF graphs
  \* \rightarrow \text{The conjunction of the translation of all graphs}

Example

$\exists x (\text{Movie}(x) \land \text{rdfs:Class}(\text{Movie}) \land \text{mdb:genre}(x, \text{mdb:action}) \land \text{rdfs:Property(mdb:genre)}) \land \text{mdb:released in}(x, 1998) \land \text{rdfs:Property(mdb:released_in)})$

Tricky here = statement on properties of properties

However, we are not done by just translating RDF to logics

– We still have to provide means to interpret and evaluate the RDF and RDFS special constructs
  \* e.g. rdfs:subClassOf, rdf:type, rdfs:Class
– Thus, we need additional axioms and rules which will help to evaluate and interpret the translated tuples
23.06.2009

12.2 RDF-Semantics

- Following axioms can be used to evaluate base RDF
  - Note: The direct model-theoretic approach to RDF semantics will need similar axioms
  - Axioms are used to construct proof trees for evaluating a RDF graphs

\[
\text{rdf:Type}(x, y) \rightarrow y(x) \\
\text{rdf:Property}(\text{rdf:Type}) \\
\text{rdf:Property}(\text{rdf:Subject}) \\
\ldots \text{(for all other rdf special names which are properties & data types)}
\]

12.2 RDF-Semantics

- RDF/s needs some more axioms

\[
\text{rdfs:Resource}(x) \\
\text{rdfs:Class}(y) \rightarrow (y(x) \leftrightarrow \text{rdf:Type}(x, y)) \\
\text{rdfs:range}(x, y) \rightarrow (x(u, v) \rightarrow y(v)) \\
\text{rdfs:domain}(x, y) \rightarrow (x(u, v) \rightarrow y(u)) \\
\text{rdfs:subClassOf}(x, y) \rightarrow \text{rdfs:Class}(x) \land \text{rdfs:Class}(y) \\
\forall u \ (x(u) \rightarrow y(u)) \\
\text{rdfs:Class}(x) \rightarrow \text{rdfs:subClassOf}(x, x) \\
\text{rdfs:Class}(x) \rightarrow \text{rdfs:subClassOf}(y, x) \\
\text{rdfs:Class}(x) \rightarrow \text{rdfs:subClassOf}(x, y) \\
\text{rdfs:Class}(x) \rightarrow \text{rdfs:subClassOf}(y, z) \rightarrow \text{rdfs:subClassOf}(x, z) \\
\text{rdfs:Class}(x) \rightarrow \text{rdfs:subClassOf}(y, z) \\
\ldots \text{(similar axioms for all other resources)} \\
\ldots \text{(similar axioms for properties)}
\]

12.2 RDF-Semantics

- Summary of semantics
  - The semantics of RDF can be established by a direct model-theoretic approach
  - or by mapping RDF to logics
    - This approach can also be used for other frame-based knowledge languages and thus allow a direct comparison of expressiveness
    - Will result in higher order logics if not further restricted
    - Different restrictions will reduce the evaluation complexity
      - Next lecture

12.2 RDF-Semantics

- RDF semantics references
  - http://www.w3.org/TR/rdf-mt/
  - http://www.w3.org/TR/2003/NOTE-lbase-20031010/

12.3 SPARQL

- After we have defined the semantics of RDF graphs, we now need a way to retrieve the information
  - SPARQL Protocol And RDF Query Language
    - SPARQL is a query language and a protocol for accessing RDF designed by the W3C RDF Data Access Working Group
  - SPARQL Query language is used to
    - extract information in the form of URIs, blank nodes, plain and typed literals
    - extract RDF subgraphs
    - construct new RDF graphs based on information in the queried graphs

12.3 SPARQL

- Basic idea is to define a set of Triple Patterns
  - Similar to an RDF Triple (subject, predicate, object), but any component can be a query variable, also literal subjects are allowed

\[
\text{rdf:Type} \ \text{rdf:Type} \ \text{rdf:Property}
\]

- Matching a triple pattern to a graph introduces bindings between variables and RDF Terms
12.3 SPARQL

- SPARQL uses a SQL-style syntax
  - Example with PREFIX dc:
    <http://purl.org/dc/elements/1.1/>
    - SELECT ?title
      FROM http://example.org/library
  - SELECT identifies the variables to be returned
  - FROM gives the name of the graph to be queried
  - WHERE query pattern as a list of triple patterns
  - Plus additional functions like LIMIT, OFFSET, or ORDER BY

12.3 SPARQL Example

- Find the URL of the blog by the person named Jon Foobar
  - SELECT ?url
    FROM <bloggers.rdf>
    WHERE {
      ?contributor foaf:name "Jon Foobar" .
    }

12.3 SPARQL – Query Forms

- SELECT
  - Returns all, or a subset of the variables bound in a query pattern match
- CONSTRUCT
  - Returns an RDF graph constructed by substituting variables in a set of triple templates
- DESCRIBE
  - Returns an RDF graph that describes the resources found
- ASK
  - Returns whether a query pattern matches or not

12.3 SPARQL – Graph Patterns

- Basic graph pattern
  - Set of triple patterns
- Group pattern
  - A set of graph patterns that must all match
- Value constraints
  - Restrict RDF terms in a solution
- Optional graph patterns
  - Additional patterns may extend the solution
- Alternative pattern
  - Two or more possible patterns are tried
- Patterns on named graphs
  - Patterns are matched against named graphs

12.3 SPARQL – Example

- Assume a graph describing people: return the full names of all people in the graph!
  - SELECT ?fullName
    WHERE {?x vCard:FN ?fullName}
  - fullName
  - "John Smith"
  - "Mary Smith"

- Return all people over 30 in the KB
  - SELECT ?x
    WHERE {?x hasAge ?age .
      FILTER(?age > 30)}
  - <http://example.org/#john>
12.3 SPARQL - Example

- Are there any married people in the graph?
  - ASK { ?person marriedTo ?spouse }

  YES

12.3 SPARQL

- For the full SPARQL standard see
  - http://www.w3.org/TR/rdf-sparql-query/

12 Next Lecture

- Ontologies, pt. 2
  - Description Logics
  - DAML+OIL
  - OWL