

Part 3: The Web Ontology Language (OWL)



Andrew W. Crapo
Natural Semantics, LLC

The Web Ontology Language (OWL)

- OWL builds on the Resource Description Framework (RDF)
 - And also upon RDF Schema (RDFS)
 - Its predecessor is DAML (DARPA Agent Markup Language) + OIL (Ontology Inference Layer)
- Identity In OWL is established by Uniform Resource Identifier (**URI**)
 - URI syntax must conform to a [standard](#) (required parts, no spaces, etc.)
 - A **namespace** is an abstract domain containing a collection of names
 - The namespace is identified by a URI
 - The names (fragment identifiers or local names) are identified by a URI
 - Namespace URI + “#” + name
 - Example: predicate class is `http://www.w3.org/1999/02/22-rdf-syntax-ns#Property`

`http://www.w3.org/1999/02/22-rdf-syntax-ns#`


`Property`
 - A namespace can be given an **alias** or **prefix** to make URIs more readable as **QNames**
 - `rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"`

`rdf:Property`

Predicates in OWL

- Multiple OWL predicates, all subclasses of *rdf:Property*
 - *owl:ObjectProperty*: predicates whose objects are individuals
 - Relationships between individuals
 - *owl:DatatypeProperty*: predicates whose objects are data values: number, strings, ...
 - Characteristics of an individual
 - *owl:AnnotationProperty*: predicates used to provide human-readable annotations on the model
 - Not generally meant to be part of the formal model, not used for inference
 - Examples: *rdfs:label* “(alias “...””, *rdfs:comment* “(note “...””
- Domain and Range
 - If *p* is a binary predicate, e.g., *p(x,y)*, then the set of all possible values of *x* is the **domain** of *p*
 - If *p* is a binary predicate, e.g., *p(x,y)*, then the set of all possible values of *y* is the **range** of *p*
 - SADL Syntax: any of
 - “*x* is described by *p* with values of type *y*.”
 - “*p* describes *x* with values of type *y*.”
 - “*p* is the relationship between *x* and *y*.”

Predicates in OWL (cont.)

- Domain and Range (cont.)
 - All domains are classes (subclasses of owl:Thing)
 - Properties of type owl:ObjectProperty have a range which is a class (subclass of owl:Thing)
 - Properties of type owl:DatatypeProperty have a range which is an [XML Schema data type](#)
 - A primitive data type, e.g., int, float, string, date, boolean, etc.
 - A [user-defined data type](#)
 - Properties of type owl:AnnotationProperty do not have a domain or range (in OWL 1.0)
- Enhance your previous model
 -  Add an rdfs:label to each instance of *Person* with their name as normally written.
 - Give the properties in the model a domain and range.
 - Open and examine the translated OWL model found in the project's *OwlModels* folder.

Solution: SADL

```
uri "http://sadl.org/Owl1b.sadl" alias owl1b.
```

```
Person is a class described by dateOfBirth with values of type date,  
described by wife with values of type Person .
```

```
PresidentOfUSA is a type of Person.
```

```
GeorgeWashington (alias "George Washington") is a PresidentOfUSA,  
with dateOfBirth "02/22/1732",  
with wife MarthaDandridge.
```

```
MarthaDandridge (alias "Martha Dandridge") is a Person.
```

Solution: OWL (RDF/XML)

```

<rdf:RDF xml:base="http://sabl.org/Owllb.sabl">
  <owl:Ontology rdf:about="">
    <owl:imports rdf:resource="builtinfunctions"/>
    <owl:imports rdf:resource="sablimplicitmodel"/>
    <owl:imports rdf:resource="sablbasemodel"/>
    <rdfs:comment xml:lang="en">
      This ontology was created from a SABL file 'Owllb.sabl' and should not be directly edited.
    </rdfs:comment>
  </owl:Ontology>
  <owl:Class rdf:ID="PresidentOfUSA">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Person"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:ObjectProperty rdf:ID="wife">
    <rdfs:domain rdf:resource="#Person"/>
    <rdfs:range rdf:resource="#Person"/>
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:ID="dateOfBirth">
    <rdfs:domain rdf:resource="#Person"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#date"/>
  </owl:DatatypeProperty>
  <owllb:PresidentOfUSA rdf:ID="GeorgeWashington">
    <owllb:wife>
      <owllb:Person rdf:ID="MarthaDandridge">
        <rdfs:label xml:lang="en">Martha Dandridge</rdfs:label>
      </owllb:Person>
    </owllb:wife>
    <owllb:dateOfBirth rdf:datatype="http://www.w3.org/2001/XMLSchema#date">1732-02-22</owllb:dateOfBirth>
    <rdfs:label xml:lang="en">George Washington</rdfs:label>
  </owllb:PresidentOfUSA>
</rdf:RDF>

```

Solution: OWL (Turtle, subset of N3)

```

@prefix : <http://sabl.org/Owl1b.sabl> .
...
<http://sabl.org/Owl1b.sabl#dateOfBirth>
  a owl:DatatypeProperty ;
  rdfs:domain <http://sabl.org/Owl1b.sabl#Person> ;
  rdfs:range xsd:date .

<http://sabl.org/Owl1b.sabl#Person>
  a owl:Class .

<http://sabl.org/Owl1b.sabl#wife>
  a owl:ObjectProperty ;
  rdfs:domain <http://sabl.org/Owl1b.sabl#Person> ;
  rdfs:range <http://sabl.org/Owl1b.sabl#Person> .

<http://sabl.org/Owl1b.sabl#MarthaDandridge>
  a <http://sabl.org/Owl1b.sabl#Person> ;
  rdfs:label "Martha Dandridge"@en .

: a owl:Ontology ;
  rdfs:comment "This ontology was created from a SABL file 'Owl1b.sabl' and should not be directly edited."@en ;
  owl:imports <http://sabl.org/builtinfunctions> , <http://sabl.org/sablimplicitmodel> , <http://sabl.org/sablbasemodel> .

<http://sabl.org/Owl1b.sabl#PresidentOfUSA>
  a owl:Class ;
  rdfs:subClassOf <http://sabl.org/Owl1b.sabl#Person> .

<http://sabl.org/Owl1b.sabl#GeorgeWashington>
  a <http://sabl.org/Owl1b.sabl#PresidentOfUSA> ;
  rdfs:label "George Washington"@en ;
  <http://sabl.org/Owl1b.sabl#dateOfBirth>
    "1732-02-22"^^xsd:date ;
  <http://sabl.org/Owl1b.sabl#wife>
    <http://sabl.org/Owl1b.sabl#MarthaDandridge> .

```

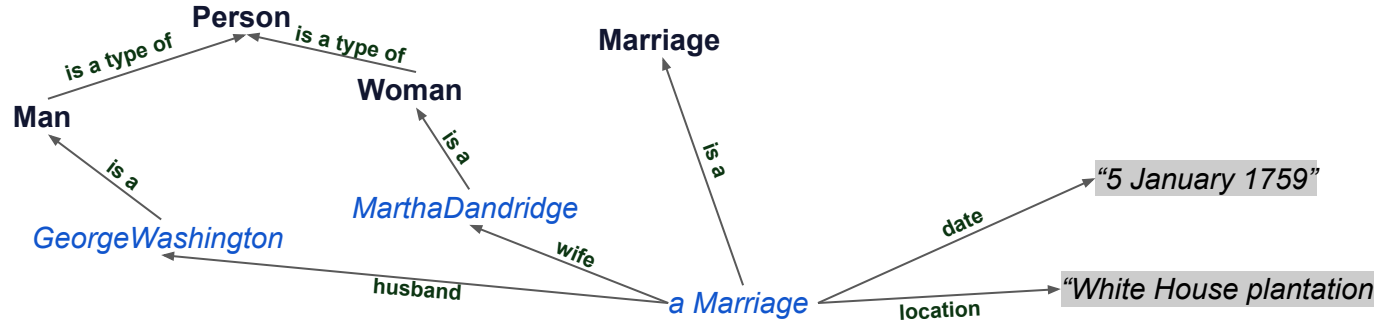
Representing N-ary Relationships

- George Washington and Martha Dandridge Custis were married on January 6, 1759 at the White House plantation in New Kent County, VA.
- One might represent this with an arity-4 predicate
 - marriage(husband, wife, date, location)
 - marriage(GeorgeWashington, MarthaDandridge, “January 5, 1759”, “White House plantation”)
- How can this be represented with binary predicates in OWL?



Solution Visualized As a Graph

- marriage(GeorgeWashington, MarthaDandridge, “January 5, 1759”, “White House plantation”)
- How can this be represented with binary predicates in OWL?



Solution in SADL

- marriage(GeorgeWashington, MarthaDandridge, “January 5, 1759”, “White House plantation”)
- How can this be represented with binary predicates in OWL?

```
uri "http://sadl.org/Owl2b.sadl" alias owl2b.
```

```
Person is a class.
```

```
{Man, Woman} are types of Person.
```

```
Marriage is a class described by husband with values of type Man,  
described by wife with values of type Woman  
described by ^date with values of type date,  
described by location with values of type string.
```

```
GeorgeWashington is a Man.
```

```
MarthaDandridge is a Woman.
```

```
A Marriage with husband GeorgeWashington,  
with wife MarthaDandridge,  
with ^date "5 January 1759",  
with location "White House plantation".
```

Things to Note

- marriage(GeorgeWashington, MarthaDandridge, "January 5, 1759", "White House plantation")
- How can this be represented with binary predicates in OWL?

```
uri "http://sad1.org/Owl2b.sad1" alias owl2b.
```

```
Person is a class.
{Man, Woman} are types of Person.
```

```
Marriage is a class described by husband with values of type Man,
described by wife with values of type Woman
described by ^date with values of type date,
described by location with values of type string.
GeorgeWashington is a Man.
MarthaDandridge is a Woman.
```

```
A Marriage with husband GeorgeWashington,
with wife MarthaDandridge,
with ^date "5 January 1759",
with location "White House plantation".
```

This is an unnamed instance of the Marriage class, identified by association with other named entities and data type values.

The keyword "date" can be used as a concept name by escaping it with a prefixed "^". (The concept name is "date"; the prefix just tells the grammar parser that this is not the keyword.)

OWL: More about Classes, Properties, and Property Restrictions



Necessary and Sufficient Conditions Revisited

- Two Classes Are Equivalent If They Have the Same Members
- The Combination of Property Restrictions on a Class and Class Equivalence Enables Expression of Necessary and Sufficient Conditions
 - $\forall x: \text{Man}(x) \leftrightarrow \text{Person}(x) \wedge \text{gender}(x, \text{Male})$
 - $\text{Person} \text{ is a } \text{Man} \text{ if and only if } \text{gender} \text{ always has value } \text{Male}.$ ¹
John is a Man with gender Person with gender Man.
 - $\forall x: \text{Parent}(x) \leftrightarrow \text{Person}(x) \wedge \exists y: \text{Person}(y) \wedge \text{child}(x, y)$
 - $\text{Person} \text{ is a } \text{Parent} \text{ if } \text{child} \text{ has at least one value of type } \text{Person}.$ ¹
Mary is a Parent with child John with child Person.

¹Logicians say “if and only if”. The SADL grammar allows this to be shortened to “only if”. Either phrase generates the same OWL.

Subproperties

- Like classes, properties can be partially ordered
 - Expressed in OWL with *rdfs:subPropertyOf*
 - All pairs related by the subproperty can be inferred to also be related by the property
- Example
 - BenjaminFranklin has ~~son~~ **child** WilliamFranklin. → BenjaminFranklin has **child** WilliamFranklin.

Types of Properties

- Transitive
 - `locatedIn` is transitive. Georgia `locatedIn` USA. \rightarrow Atlanta `locatedIn` USA.
- Symmetrical
 - `spouse` has symmetrical. \rightarrow Jane has `spouse` Tarzan.
- Functional: a Given Subject Can Have Only One Value of the Property
 - `biologicalFather` has a single value.
- Inverse Functional: Only One Subject for a Given Value of the Property
 - `isBirthMotherOf` has a single subject.
- Inverse
 - `ownedBy` is the inverse of `owns`. \rightarrow MountVernon `ownedBy` GeorgeWashington.
 - Can an owl:DatatypeProperty have an inverse property?



Answer

- Can an owl:DatatypeProperty have an inverse property?
 - No, the value of an owl:DatatypeProperty is a data value (a number, date, string, etc.) or a value of a user-defined data type and so is not an individual and cannot be the subject of a statement with the inverse property as predicate.

Classes (Sets) Revisited

- Two Ways to Define a Class (Set)

1. Extensionally: the members of the set are enumerated

- `Season` is a class, can only be one of {Spring, Summer, Fall, Winter}.

```
<owl:Class rdf:type="owl:Class" id="Season">
  <owl:disjointClasses>
    <owl:Class rdf:type="owl:Class" id="Spring"/>
    <owl:Class rdf:type="owl:Class" id="Summer"/>
    <owl:Class rdf:type="owl:Class" id="Fall"/>
    <owl:Class rdf:type="owl:Class" id="Winter"/>
  </owl:disjointClasses>
</owl:Class>
```

2. Intensionally: the conditions of set membership are defined and those things meeting the conditions can be inferred to be members

- How to define the conditions of intensional set membership?

1. By establishing necessary and sufficient conditions on the class through axioms
2. By explicit rules concluding class membership (see Rules in a latter section)

Necessary vs. Necessary and Sufficient Conditions

- Necessary Conditions: What Properties Members of a Class Should Have
 - Implemented as property restriction conditions on the class
 - The class is a subclass of all things meeting the conditions
- Necessary & Sufficient Conditions: When Met, also Imply Class Membership
 - Implemented as owl:equivalentClass relationship between conditions and the class
 - The class is an equivalent class¹ to all things meeting the conditions

¹Equivalent classes have exactly the same members. Therefore, if an instance belongs to one class it must belong to the other.

Property Restrictions on a Class

- Four Main Types of Property Restrictions on a Class
 - Cardinality
 - Some Values From
 - All Values From
 - Has Value

Cardinality Property Restrictions on a Class

- Cardinality Restrictions Can Be
 - [Exact] cardinality: any subject from the class has that number of values of the property
 - spouse of Person has exactly 1 value.
 - Minimum cardinality: any subject from the class has at least that many values of the property
 - child of Parent has at least 1 value.
 - Maximum cardinality: any subject from the class has at most that many values of the property
 - loan of LibraryPatron has at most 50 values.
- Qualified Cardinality Specifies the Cardinality for Each Type of Value
 - digit of Hand has exactly 4 values of type Finger.
 loan of LibraryPatron has at most 10 values of type DVD.

Other Property Restrictions on a Class

- Some Values From: Members of the Restricted Class Must Have One or More Values of the Property from the Value Class
 - An expression of existential quantification in OWL¹
 - `pet` of `CatLover` has at least one value of type `Cat`.¹
 - Existential quantification: $\forall x: \text{CatLover}(x) \rightarrow \exists y: \text{Cat}(y) \wedge \text{pet}(x,y)$ ²
- All Values From: Members of the Restricted Class Can Only Have Values of the Property from the Value Class (but Do Not Necessarily Have Any Values)
 - An expression of universal quantification in OWL
 - `loves` of `RichMan` only has values of type `Money`.
 - Universal quantification: $\forall x \forall y: \text{RichMan}(x) \wedge \text{loves}(x,y) \rightarrow \text{Money}(y)$ ²
- Has Value: Members of the Restricted Class Must Have the Specified Value for the Property
 - `gender` of `Woman` always has value `Female`.

¹A subtle distinction in SADL grammar: “at least 1 value of type” is qualified cardinality, “at least one value of type” is some values from.

²The first quantifier, $\forall x$, ranges over the members of the restricted class. The second quantifier, $\exists y$ or $\forall y$, is the one called out here.

Model Building Exercise

- Build a genealogy model capturing the following features.
 - Class: Person
 - Defined subclasses: Man, Woman, Husband, Wife, Parent
 - Properties: parent, child, spouse, sibling, gender, age, dateOfBirth
 - Additional restrictions: age ≥ 0 , no more than 1 spouse
 - Class: Location
 - Properties: description, latitude (-90 to 90, inclusive), longitude (-180 to 180, inclusive)
 - Class: Birth
 - Properties: mother, baby, date, location
 - Restrictions: exactly 1 mother, ≥ 1 baby



One Plausible Solution

```
uri "http://sadl.org/Genealogy1.sadl" alias gen1.
```

```
Gender is a class, must be one of {Male, Female}.
```

```
Person is a class described by gender with values of type Gender,
  described by dateOfBirth with values of type date,
  described by age with values of type nonNegativeInteger,
  described by parent with values of type Person.
```

```
A Person is a Woman only if gender always has value Female.
```

```
A Person is a Man only if gender always has value Male.
```

```
child describes Person with values of type Person.
```

```
spouse describes Person with values of type Person.
```

```
spouse of Person has at most 1 value.
```

```
sibling describes Person with values of type Person.
```

```
A Man is a Husband only if spouse has at least 1 value.
```

```
A Woman is a Wife only if spouse has at least 1 value.
```

```
A Person is a Parent only if child has at least 1 value.
```

```
latValue is a type of double [-90.0,90.0].
```

```
longValue is a type of double [-180.0, 180.0].
```

```
Location is a class described by latitude with values of type latValue
  described by longitude with values of type longValue,
  described by description with values of type string.
```

```
Birth is a class,
```

```
  described by mother with a single value of type Woman,
```


```
  described by baby with values of type Person,
```

```
  described by ^date with values of type date,
```

```
  described by location with values of type Location.
```

```
baby of Birth has at least 1 value.
```

How is each underlined definition realized?

- Build a genealogy model capturing the following features.
 -  Class: Person
 - Defined subclasses: Man, Woman, Husband, Wife, Parent
 - Properties: parent, child, spouse, sibling, gender, age, dateOfBirth
 - Additional restrictions: age >= 0, no more than 1 spouse
 - Class: Location
 - Properties: description, latitude (-90 to 90, inclusive), longitude (-180 to 180, inclusive)¹
 - Class: Birth
 - Properties: mother, baby, date, location
 - Restrictions: exactly 1 mother, >= 1 baby

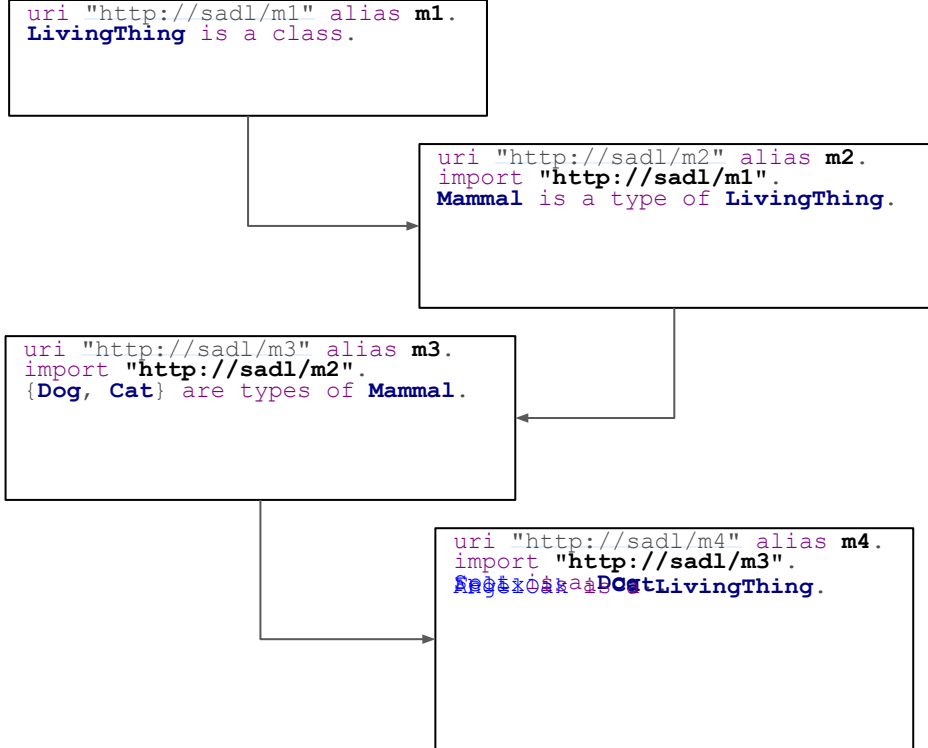
¹See [User-Defined Data Types](#).

OWL: Modularity



Modularity in OWL Knowledge Bases

- Modularity enables reusability, makes kbases more manageable.
- A hierarchy of kbase modules are possible using the *owl:imports* property.
 - All of the concepts known to an imported model are known to the importing model
 - *owl:imports* is transitive--if *m4* imports *m3* and *m3* imports *m2* and *m2* imports *m1*, then it is as if *m4* imports *m1* (see *LivingThing*)



Modularity Exercise

- Create an Instance Data Model Importing Your Genealogy Model.
 - Create a new model (the instance data model)
 - Import the genealogy model previously created (the meta-model)
 - Add instances of each of the classes in the meta-model and relate them using meta-model properties to create a meaningful knowledge graph



Instance Data Example: The Adams Family

```

uri "http://sabl.org/AdamsFamily.sabl" alias adamsfamily.
import "http://sabl.org/Genealogy1.sabl".

JohnAdams (alias "John Adams") is a Person
  with gender Male,
  with dateOfBirth "Oct 30, 1735",
  with parent (a Man JohnAdamsSr (alias "John Adams Sr")),
  with parent (a Woman SusannaBoylston (alias "Susanna Boylston")),
  with spouse (a Woman AbigailSmith (alias "Abigail Smith", "Abigail Adams")
    with dateOfBirth "22 Nov 1744").

A Birth has baby AbigailSmith,
  has location (a Location with description "Weymouth, MA")
  has mother (a Woman ElizabethQuincy (alias "Elizabeth Smith")).

A Birth has baby JohnAdams,
  has location (a Location with description "Braintree, MA")
  has mother SusannaBoylston.

JohnAdamsSr has spouse SusannaBoylston.
ElizabethQuincy has spouse (a Man WilliamSmith (alias "William Smith")).
JohnQuincyAdams (alias "John Quincy Adams") is a Man
  with parent JohnAdams,
  with parent AbigailSmith,
  with dateOfBirth "Jul 11, 1767".

```

OWL: Queries



Graph Patterns and Queries

- Triple Pattern: *<subject> <predicate> <object>*¹
 - A triple pattern can be used to query a knowledge graph
 - The triple can be completely specified
 - *<Mary> <child> <John>*
 - A proposition; is either *true* (in the kbase) or *false* (not in the kbase)
 - The triple can be partially specified
 - *<Mary> <child> ?o*
 - Find the object values of triples with subject *Mary*, predicate *child*.
 - *<Mary> ?p ?o* Find all predicates and object values of triples with subject *Mary*.

¹Here and in the following slides we adopt the syntax of the W3C [SPARQL](#) graph query language except where SADL syntax is used.

Graph Patterns and Queries (cont.)

- Variables Link Multiple Triple Patterns and Enable More Complex Queries
 - *<Mary> <child> ?x . ?x <friend> ?y*
 - *?x* represents the unbound variable *x*, *?y* the unbound variable *y*
 - *x* will be bound to each *child* of *Mary*
 - For each value of *x*, *y* will be bound to each *friend* of that child
 - The dot (‘.’) between triple patterns separates the patterns, may be read as “and”
 - *?x <child> ?y . ?y <friend> <John>*
 - Find everyone who has a *child* who has *friend John*.
- The *select* Keyword Allows Specification of Which Values to Return
 - *select ?x ?y where {?x <child> ?y . ?y <friend> <John>}*
 - Return the parent and the child for each child that has *friend John*
 - *select ?x where {?x <child> ?y . ?y <friend> <John>}*
 - Return just the parents who have children who have *friend John*

Query Filters, Optional Triple Patterns


- Graph Patterns Are Insufficient as a Query Language
 - Patterns only allow exact matches in the knowledge graph
 - Filters enable refinement of results
- A Filter Is Applied After The Graph Patterns Are Matched to Reduce Results
 - `select ?p ?g where {?p <rdf:type> <Person> . ?p <age> ?g . filter(?g >= 18)}`
 - Find all *Persons* and their *age* who have *age* 18 or older.
- One or More Triple Patterns Can Be Wrapped with *Optional*
 - Allows sparse data to be returned
 - `select ?p ?g ?y where {?p <rdf:type> <Person> . ?p <gender> ?g . OPTIONAL{?p <dateOfBirth> ?y}}`
 - "p", "g", "y"
 - "JohnAdams", "Male", "Sun Oct 30 00:00:00 EST 1735"
 - "JohnAdamsSr", "Male", null

The W3C SPARQL Query Language

- SPARQL Supports Many Other Constructs, see [W3C Docs](#), also [Cheat Sheet](#)
- The Jena Reasoner in SADL Uses the [Jena ARQ SPARQL Query Engine](#)
- SADL Grammar Supports SPARQL Queries as Opaque Strings
 - Ask: "select ?p ?g where {?p <rdf:type> <Person> . ?p <gender> ?g}!".
 - Limited error checking of query before execution
- SADL Grammar Supports Simple Queries as SADL Query Expressions
 - Ask: select **p**, **g** where **p** is a **Person** and **p** has **gender** **g**.
 - Concepts semantically colored, linked to knowledge graph definitions and other references
 - Query translated to and executed as SPARQL when using Jena Reasoner
- Besides *select*, SPARQL supports *ask*, *construct*, *describe*, and *update* where *update* includes *delete data*, *insert data* (see references above).
- SADL query grammar supports simple queries, excluding *describe*.

Query Exercise

Create the Following Queries in Your Genealogy Instance Data Model.

-  1. Find all spouses
2. Find all instances of Person and, if known, their gender, date of birth, location of birth, and spouse.
3. Find, for each person, their parents, if known.

Solution (and Query Results for Adams Family Data)

1. Find all spouses.

Ask: select **s1**, **s2** where **s1** has **spouse s2**.

"s1","s2"

"ElizabethQuincy","WilliamSmith"

"JohnAdamsSr","SusannaBoylston"

"JohnAdams","AbigailSmith"

2. Find all instances of Person and, if known, their gender, date of birth, location of birth, and spouse.

Ask: OPTIONAL (?p <rdf:type> .<Person> ?g ?d ?l ?s)

"p","g","y","ld"¹

"JohnAdams","Male","Sun Oct 30 00:00:00 EST 1735","Braintree, MA"

¹In the instance data of AdamsFamily.sadl, JohnAdams is the only individual explicitly given a value of *gender*.

Solution (cont.)

3. Find, for each person, their parents, if known.

```
Ask: "select ?p ?pnt where {?p <?d?ontype> <Person>
} UNION OPTIONAL{?p <?d?ontype> <Baby> ?p -- ?p <?d?ontype> <mother> ?pnt}
```

```
"p", "pnt"1
```

```
"JohnAdams", "JohnAdamsSr"
```

```
"JohnAdams", "SusannaBoylston"
```

¹As in the previous query result, JohnAdams is the only individual explicitly declared to belong to the Person class so only he matches the first condition.

OWL: Entailments, Rules, and Reasoning



OWL Entailments

- Necessary Conditions (covered above, class \subseteq property restrictions)
 - Example: `child` of `Parent` has at least 1 value of type `Person`.
- Nec. & Sufficient Conditions (covered above, class \equiv property restrictions)
 - Example: A `Person` is a `Parent` if and only if `child` has at least 1 value.
- Domain and Range Implications
 - $rdfs:domain(p, c1) \wedge p(i1, y) \rightarrow c1(i1)$
 - $rdfs:range(p, c2) \wedge p(i1, y) \rightarrow c2(y)$
 - This is sometimes surprising to people who are thinking in terms of constraint languages
 - $rdfs:domain(p, c1) \wedge p(i1, y) \Rightarrow$ if $c1(i1)$ is not known to be true, then type error for $i1$
 - $rdfs:range(p, c2) \wedge p(i1, y) \Rightarrow$ if $c2(y)$ is not known to be true, then type error for y
 - SADL does type checking with results displayed as errors or warning per preference setting
- Actual OWL entailments inferred depend upon reasoner and reasoner settings chosen
- OWL Reasoning Can Be Computationally Intensive

Rules

- If-then Rules Implement Propositional Logic Implication (\rightarrow)

- When conditions (if...) are met, conclusions (then ...) are inferred

- Example Using Genealogy Model Concepts

Rule: **Parent From Birth Rule:** *has baby* **bb** and **b** *has mother* **m**

- Instead of Variables, Indefinite and Definite Articles Can Be Used in SADL

- Preference must be set accordingly
- Indefinite article identifies a free variable to be bound and definite article is a reference to it
 - Similar to English usage of articles
 - Ordinals indicate additional variables of the same type, e.g., “*a second Person*”

Rule: **Parent From Birth Rule?** *Person* **Woman**. *Birth* *has mother* *a Woman*

Rules (cont.)

- Rules Are NOT Part of OWL Standard
- Rules ARE Often Used with OWL and Processed by Reasoners
 - Semantic Web Rule Language (SWRL), Jena Rules, and Prolog are rule formats
 - Reasoning must seamlessly integrate OWL entailments and rule inferences
 - OWL entailments can be *selectively* implemented as rules, possibly improving performance
- SADL Currently Offers Two Reasoner Choices, with Compatible Translators
 - [Jena Reasoner](#): an RDF rule engine with it's own rule syntax. OWL entailments is by rules, various possible rule sets are identified by [model specification](#) setting in project properties
 - [SWI-Prolog](#) Reasoner: uses SWI-Prolog as a service to do reasoning. OWL entailments can be accomplished by rules but user must supply
 - Other reasoners can be added by wrapping them in the appropriate Java Interface class and providing a translator, also wrapped in the appropriate Interface class

Built-in Functions in Rules

- Remember the Predicate Logic *Function* Symbol?
 - A symbol that takes 0 or more arguments and returns another non-logical symbol
- Rules use functions--AKA built-ins--to compute non-logical values
 - Math operations: addition, subtraction, multiplication, etc.
 - String operations
 - List operations
 - Example: Rule **AreaOfRect**: if **x** is a **Rectangle** then **area** of **x** is **height** of **x** * **width** of **x**.
 - Jena translation: [AreaOfRect: (?x rdf:type http://sabl.org/Shapes/Rectangle#Rectangle),
 (?x http://sabl.org/Shapes/Rectangle#height ?v0),
 (?x http://sabl.org/Shapes/Rectangle#width ?v1),
product(?v0, ?v1, ?v2)
 -> (?x http://sabl.org/Shapes/Shapes#area ?v2)]
 - Function **product** takes input arguments *v0* and *v1* and binds result to *v2*
 - Users can define and add their own functions, [in Jena](#) and [in SWI-Prolog](#).

Exercise

- Using the genealogy model previously constructed:
 - Add a rule to the model which will infer the *sibling* relationship.
 - Add a query to find all *sibling* relationships.
 - Select SADL Perspective (Window->Perspective->Other Perspective->Other->SADL).
 - Run inference (SADL->Test Model). Add instance data if necessary to obtain results.
 - Set the Jena reasoner's model specification to "OWL_MEM_RDFS"
(Project->Properties->SADL->ReasonerPreferences, click on "Jena", then "Edit", then set model specification).
 - Rerun inference and compare the results of the various queries to the previous results.
 - Optional:
 - Set the Jena reasoner's model specification to "OWL_DL_MEM_RULE".
 - Rerun inference.



Solution

1. or b) Rule `SiblingRule` and `x` has `Parent` `x` and `x` != `z`
~~`if (isPerson(x) && isSibling(x,z)) Person Person .Person`~~
2. Find all *sibling* relationships: Ask: select `s1`, `s2` where `s1` has `sibling` `s2`.
 - a. Inference result is empty (for AdamsFamily.sadl data above)
 - b. Add additional instance data to AdamsFamily.sadl
`AbigailAmeliaAdams (Abigail "Nabby" Amelia Adams Smith) is a Woman`
 - c. Rerunning (with rule 1.a above)¹, sibling query results:
 - "s1","s2"
 - "JohnQuincyAdams","AbigailAmeliaAdams"
 - "AbigailAmeliaAdams","JohnQuincyAdams"

¹If rule 1.b is used, the type constraint in “if a Person...” will prevent the rule from firing as neither JohnQuincyAdams nor AbigailAmeliaAdams are know to be of type Person. Rule 1.a has no such type constraint.

Solution (cont.)

3. With Jena Reasoner's Model Specification Set to OWL_MEM (the Default):

```
Query: select ?p ?g ?y ?ld where {?p <rdf:type> <Person> .  
    OPTIONAL{?p <gender> ?g} .  
    OPTIONAL{?p <dateOfBirth> ?y} .  
    OPTIONAL{?b <baby> ?p . ?b <location> ?l . ?l <description> ?ld}  
}  
"p","g","y","ld"  
"JohnAdams","Male","Sun Oct 30 00:00:00 EST 1735","Braintree, MA"
```

Explanation: This Jena model specification does not do any OWL entailments. JohnAdams is the only instance of the Person class so the only match to the query pattern.

Solution (cont.)

4. With Jena Reasoner's Model Specification Set to OWL_MEM (the Default): (cont.)

```
Query: select ?p ?pnt where {?p <rdf:type> <Person> .
  {
    optional{?p <parent> ?pnt}
  }
  UNION
  {
    optional{?b <baby> ?p . ?b <mother> ?pnt}
  }
}
```

```
"p","pnt"
"JohnAdams","JohnAdamsSr"
"JohnAdams","SusannaBoylston"
```

Explanation: This Jena model specification does not do any OWL entailments. JohnAdams is the only instance of the Person class so the only match to the query pattern.

Solution (cont.)

5. With Jena Reasoner's Model Specification Set to OWL_MEM_RDFS, differing results are:

```

Query: select ?p ?g ?y ?ld where {?p <rdf:type> <Person> .
      OPTIONAL{?p <gender> ?g} .
      OPTIONAL{?p <dateOfBirth> ?y} .
      OPTIONAL{?b <baby> ?p . ?b <location> ?l . ?l <description> ?ld}
}

```

"p","g","y","ld"
 "JohnAdams","Male","Sun Oct 30 00:00:00 EST 1735","Braintree, MA"
 "AbigailSmith",null,"Sun Nov 22 00:00:00 EST 1744","Weymouth, MA"
 "JohnAdamsSr",null,null,null
 "SusannaBoylston",null,null,null
 "WilliamSmith",null,null,null
 "AbigailAmeliaAdams",null,"Sun Jul 14 00:00:00 EST 1765",null
 "JohnQuincyAdams",null,"Sat Jul 11 00:00:00 EST 1767",null
 "ElizabethQuincy",null,null,null

Explanation: This Jena model specification does transitive closure over class hierarchies (infers instances of a class are also instances of any superclass). This expands membership in the Person class to include many more individuals who now match the query pattern.

Solution (cont.)

6. With Jena Reasoner's Model Specification Set to OWL_MEM_RDFS, differing results are: (cont.)

```

Query: select ?p ?pnt where {?p <rdf:type> <Person> .
    {
        optional{?p <parent> ?pnt}
    }
    UNION
    {
        optional{?b <baby> ?p . ?b <mother> ?pnt}
    }
}
"p","pnt"
"JohnAdams","JohnAdamsSr"
"JohnAdams","SusannaBoylston"
"AbigailSmith","ElizabethQuincy"
"JohnAdamsSr",null
"SusannaBoylston",null
"WilliamSmith",null
"AbigailAmeliaAdams","JohnAdams"
"AbigailAmeliaAdams","AbigailSmith"
"JohnQuincyAdams","JohnAdams"
"JohnQuincyAdams","AbigailSmith"
"ElizabethQuincy",null

```

Explanation: This Jena model specification does transitive closure over class hierarchies (infers instances of a class are also instances of any superclass). This expands membership in the Person class to include many more individuals who now match the query pattern.

Solution (cont.)

7. With Jena Reasoner's Model Specification Set to OWL_MEM_RDFS:

Rule 1.b

```
Rule SiblingRuleAlt:
if a Person hasSibling and is Person . Person
```

will provide sibling query results:

```
Query: select ?s1 ?s2 where {?s1 <http://sadl.org/Genealogy1.sadl#sibling> ?s2}
"s1","s2"
"JohnQuincyAdams","AbigailAmeliaAdams"
"AbigailAmeliaAdams","JohnQuincyAdams"
```

because all instances of Men and Women will be inferred to be instances of Person.

Solution (cont.)

8. With Jena Reasoner's Model Specification Set to OWL_DL_MEM_RULE

Depending on the specific instance data provided, running inference with this model specification may be very slow or may never return, in which case the Eclipse instance must be killed forcefully and Eclipse restarted. It is often sufficient to use an RDFS model specification and add very specific rules to provide desired OWL entailments with better performance. For example:

Rule **SpouseSymmetry**: if **x** has **spouse y** then **y** has **spouse x**.

Query: select ?s1 ?s2 where {?s1 <http://sabl.org/Genealogy1.sabl#spouse> ?s2}

"s1","s2"

"WilliamSmith","ElizabethQuincy"

"SusannaBoylston","JohnAdamsSr"

"AbigailSmith","JohnAdams"

"ElizabethQuincy","WilliamSmith"

"JohnAdamsSr","SusannaBoylston"

"JohnAdams","AbigailSmith"

Additional Information

- For More Information on OWL:
 - <https://www.w3.org/OWL/>
- For More Information on SADL:
 - <http://semanticapplicationdesignlanguage.github.io/sadl/>
- To Access This Tutorial as Video or Slides:
 - http://semanticapplicationdesignlanguage.github.io/sadl/Tutorial/FCSM_Introduction.pdf