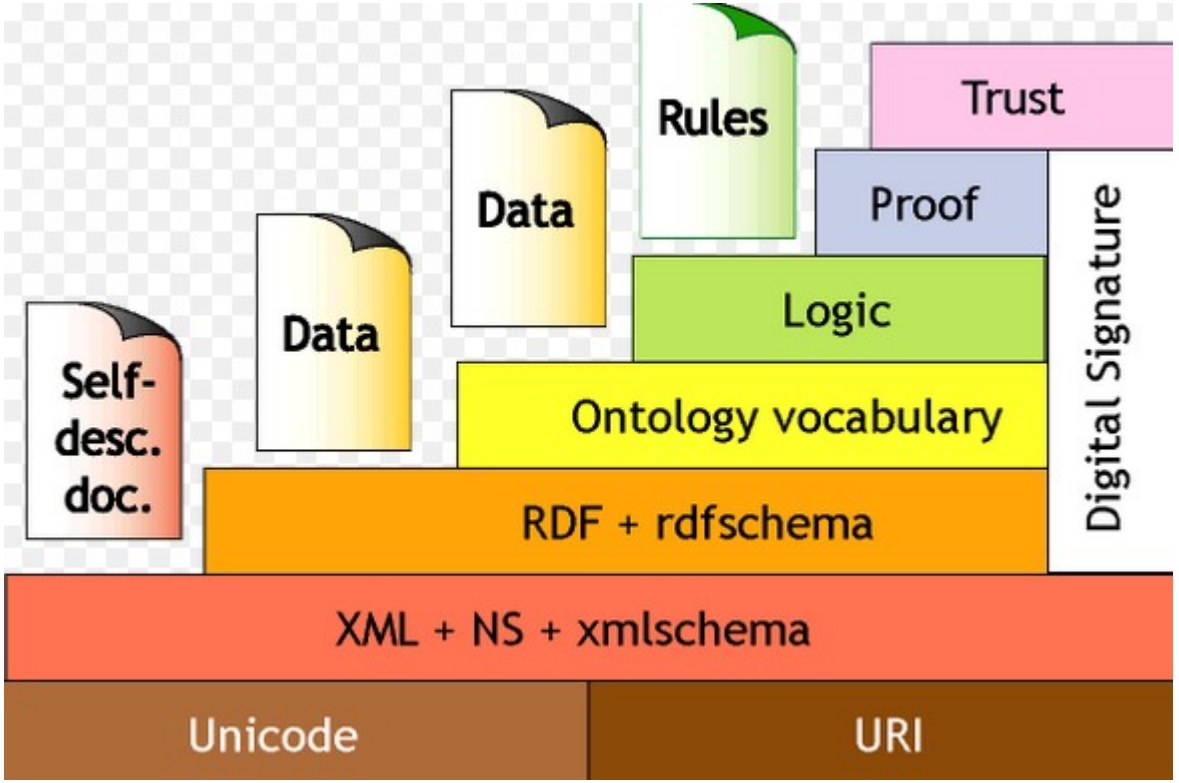
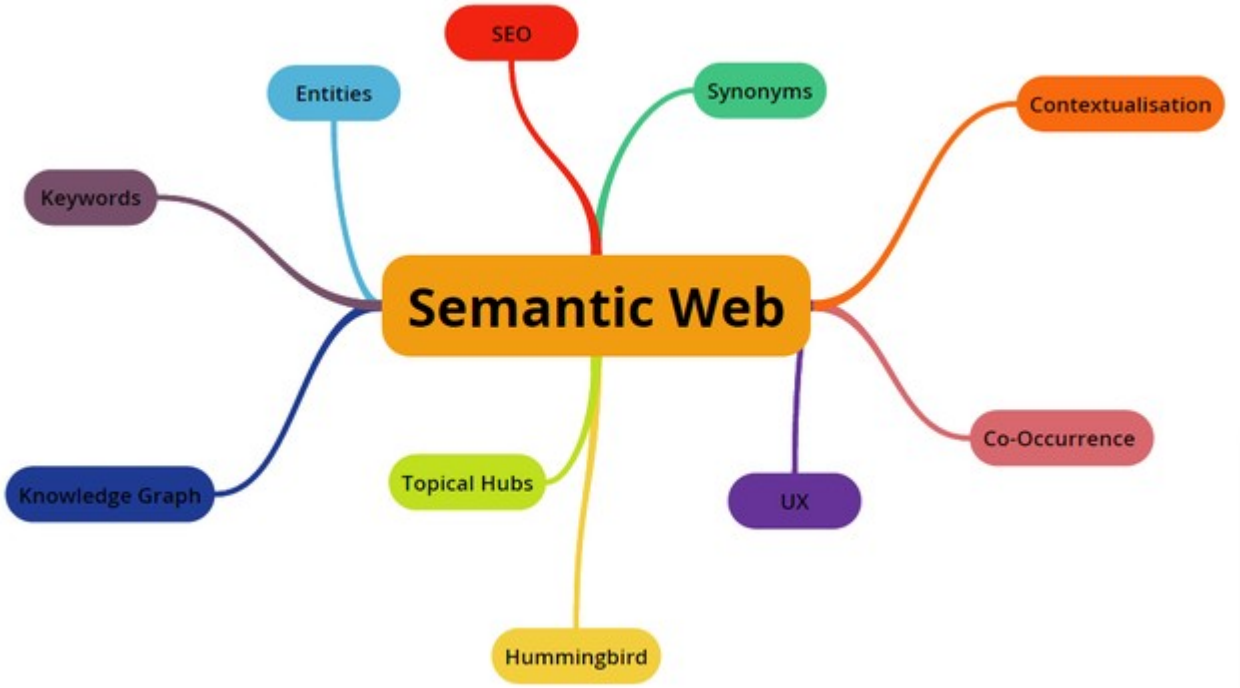
OWL

Iztok Savnik

Kje smo?


Rules
Trust
Data
Proof
Data
Logic
Self-
desc.
doc.
Ontology vocabulary
Digital Signature
RDF + rdfschema
XML + NS + xmlschema
Unicode
URI

Kje smo?


SEO
Entities
Synonyms
Contextualisation
Keywords
Semantic Web
Co-Occurrence
Knowledge Graph
Topical Hubs
UX
Hummingbird

Sklad jezikov

- XML
  - Sintaksa, ni semantike
- XML shema
  - Opisuje strukture XML dokumenti
- RDF
  - Podatkovni model za “relacije” med “objekti”
- RDF shema
  - RDF jezik za definicijo slovarjev
  - Podatkovni model
- OWL
  - Logika!

Pregled RDF sheme

- RDFS omogoča
  - razredi
  - hierarhija razredov
  - lastnosti
  - hierarhija lastnosti
  - omejitve domene in zaloge vrednosti
- RDFS ne omogoča
  - značilnosti lastnosti (inverz, tranzitivnost, ...)
  - lokalne omejitve zaloge vrednosti
  - definicija kompleksnih konceptov
  - kardinalnost
  - logični aksiomi in
  - logične definicije razredov

Načrtovalski cilji OWL

- Na razpolago (za Web)
- Predpostavka odprtega sveta
- Vsebina se spreminja skozi čas
- Interoperabilnost
- Odkrivanje nekonsistence
- Usklajevanje med izrazno močjo in učinkovitostjo
- Enostaven za uporabo
- Kompatibilno z obstoječimi standardi
- Internacionalizacija

Zahteve OWL

- Ontologije so objekti na Web
  - Meta-podatki, verzije
- Ontologije so razširljive
  - Razredi, lastnosti, podatkovni tipi, domene/zaloge vrednosti, objekti, enakost, razredi kot instance, kardinalnost
- Sklepanje
  - Odločljivo, fragmenti predikatnega računa, nivoji jezika, učinkovito za realne KB
- Sintaksa
  - XML in RDF sinatksa

Razširitev RDF sheme

- OWL razširi RDF shemo na kompleten jezik za predstavitev znanja in podatkov na Web
  - logični izrazi (and, or, not)
  - (ne)enakost
  - lokalne lastnosti
  - obvezne/neobvezne lastnosti
  - obvezne vrednosti
  - naštevni razredi
  - simetrija, inverzi

Kaj OWL ne vsebuje

- Privzete vrednosti
- Predpostavka o zaprtem svetu
- Veriženje lastnosti
- Aritmetika
- Operacije nad nizi
- Delno uvažanje
- Definicija oken
- Postopkovno
- Priponke (attachments)

What is OWL?

- OWL 2 is a language for expressing ontologies
- An ontology is a set of precise descriptive statements about some part of the world
- A terminology consists of:
  - set of central terms–called vocabulary, and,
  - the meaning of terms characterized by stating how terms are interrelated to the other terms

A terminology

What is OWL?

- OWL is a knowledge representation language, designed to formulate, exchange and reason with knowledge about a domain of interest
- Axioms:
  - the basic statements that an OWL ontology expresses
- Entities:
  - elements used to refer to real-world objects
- Expressions:
  - combinations of entities to form complex descriptions from basic ones

What is OWL?

- OWL is a powerful general-purpose modeling language that can capture parts of human knowledge
  - The results of the modeling processes are called ontologies
  - Knowledge base is composed of sentences
  - Statements like “it is raining” or “every man is mortal”
- OWL is used to draw consequences from knowledge
  - Deductive reasoner can derive new statements from existing statement

What is OWL?

- What does it mean that a statement is a consequence of other statements?
  - it means that this statement is true whenever the other statements are
- OWL can also check if statements in ontology are consistent, or if a newly inserted statement is consistent with regards to ontology

What is OWL?

- Semantic Web KR language based on description logics (DLs)
  - OWL DL is essentially DL SROIQ(D)
  - KR for web resources, using URIs.
  - Using web-enabled syntaxes, e.g. XML or RDF

Osnova OWL je opisna logika

- OWL ima več nivojev
  - Nivoji zgrajeni glede na izrazno moč gradnikov
  - OWL Lite, OWL DL, OWL Full

Jezikovni nivoji OWL

- OWL Lite
  - Klasifikacijska hirearhija
  - Enostavne omejitve
- OWL DL
  - Maksimalna izraznost
  - Ohranitev izračunljivosti
  - Standardna formalizacija z DL
- OWL Full
  - Zelo velika izraznost
  - Ni izračunljiv
  - Vsa sintaktična svoboda RDF

Značilnosti OWL nivojev

OWL Lite

- (sub)razredi, objekti
- (sub)lastnosti, domena, zaloga vrednosti
- konjunkcija
- (ne)enakost
- kardinalnost 0/1
- podatkovni tipi
- inverzi, tranzitivnost, simetrične lastnosti
- someValuesFrom
- allValuesFrom

OWL DL

- negacija
- disjunkcija
- kompletna kardinalnost
- naštevni tipi
- hasValue

OWL Full

- Meta-razredi
- spreminjanje podatkov

OWL Lite

- Ni eksplicitne negacije ali unije
- Omejena kardinalnost (0/1)
- Ni imenskih tipov (oneOf)
- Semantika osnovana na DL
  - sklepanje na osnovi DL sistemih (+podatkovni tipi)
- Semantično samo majhna omejitev OWL DL
  - ni imenskih tipov
  - ni poljubne kardinalnosti

OWL DL

- Uporaba slovarja je omejena
  - Ne more biti uporabljena za “grde trike” (npr. za spremembo OWL)
  - Razredi ne morejo biti instance
- Uporabljena teorija modelov osnovana za DL
  - Direktna preslikava iz DL
  - Sklepanje z DL sistemi

OWL Full

- Ni omejitev glede uporabe slovarja (dokler je legalno znotraj RDF)
  - Razredi in objekti (in več…)
- Teorija modelov za RDF
  - Sklepanje z uporabo FOL sistema
  - Semantika naj ustreza OWL DL za omejene baze znanja

OWL konstrukti

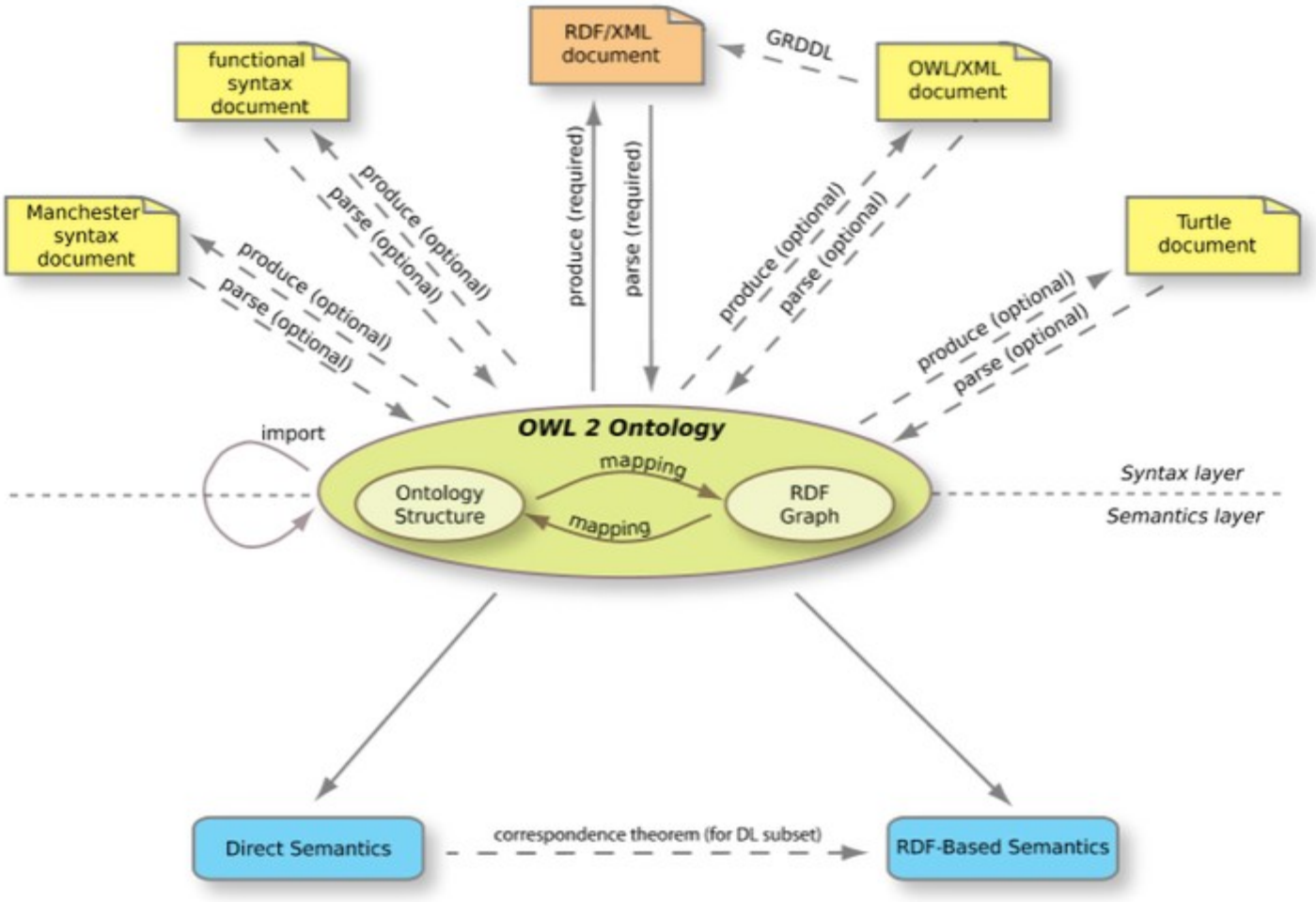
OWL konstrukt	DL	Primer
intersectionOf	$C_1 \sqcap \ ... \sqcap C_n$	Human⊓Male
unionOf	$C_1 \sqcup \ ... \ \sqcup$ Cn	Doctor⊔Lawyer
complementOf	¬C	¬Male
oneOf	$\{o_1, ..., o_n\}$	{john, mary}
allValuesFrom	∀P.C	∀hasChild.Doctor
someValuesFrom	∃P.C	∃hasChild.Lawyer
value	∃P.{o}	∃citizenOf .USA
minCardinality	≥nP.C	≥2hasChild.Lawyer
maxCardinality	≤nP.C	≤1hasChild.Male
cardinality	=nP.C	=1hasParent.Female

+ XML Schema tipi: int, string, real, etc...

Več sintaks OWL

- Abstraktna sintaksa
  - enostavneje pisati in brati
  - bližje opisni logiki in okvirjem
- RDF grafi
  - OWL je del Semantičnega spleta !
  - OWL naj bo razširitev RDF (N3, Turtle, RDF/XML, OWL/XML)
  - http://www.w3.org/TR/2012/REC-owl2-mapping-to-rdf-20121211/

Pregled


RDF/XML document
GRDDL
OWL/XML document
functional syntax document
Manchester syntax document
Turtle document
produce (optional)
parse (optional)
produce (required)
parse (required)
produce (optional)
parse (optional)
produce (optional)
parse (optional)
produce (optional)
parse (optional)
import
OWL 2 Ontology
mapping
mapping
Ontology Structure
RDF Graph
Syntax layer
Semantics layer
Direct Semantics
correspondence theorem (for DL subset)
RDF-Based Semantics

Osnovni gradniki OWL

- Razredi in instance
- Hierarhije razredov
- Tuji razredi
- Lastnosti razredov
- Hierarhije lastnosti
- Domena in zaloga vrednosti lastnosti

Razredi in instance

- Primeri: `person,pet,cat`
- Kolekcija individualnih objektov (stvari, ...)

```
ClassAssertion( :Person :Mary )
:Mary rdf:type :Person .

ClassAssertion( :Woman :Mary )
:Mary rdf:type :Woman .
```

Hierarhije razredov (1)

- Relacije pod-razred, ISA, specializacija, dedovanje, ...

```
SubClassOf( :Woman :Person )
:Woman rdfs:subClassOf :Person .
```

- OWL sklepalnik lahko zaključi iz aksioma, da je :Mary instanca :Person, če je instanca :Woman.

Hierarhije razredov (2)

- Še en podrazred :Woman.

```
SubClassOf( :Mother :Woman )
:Mother rdfs:subClassOf :Woman .
```

- Zdaj lahko sklepamo, da je :Mary, ki je instanca :Mother tudi instanca :Woman in :Person.
- Interpretacije: I(:Mother) ⊆ I(:Woman) ⊆ I(:Person)

Enakost in različnost razredov

- Povemo lahko, da sta dva razreda enaka:

```
EquivalentClasses( :Person :Human )
 :Person owl:equivalentClass :Human .
```

- Dva razreda sta različna:

```
DisjointClasses( :Woman :Man )
[]  rdf:type      owl:AllDisjointClasses ;
    owl:members   ( :Woman  :Man ) .
```

Lastnosti objektov

- Kako opisujemo lastnosti objektov:

```
ObjectPropertyAssertion( :hasWife :John :Mary )
:John :hasWife :Mary .
```

- Opišemo lahko negativne lastnosti:

```
NegativeObjectPropertyAssertion( :hasWife :Bill :Mary )
[]  rdf:type
owl:NegativeObjectPropertyAssertion ;
     owl:sourceIndividual   :Bill ;
     owl:assertionProperty  :hasWife ;
     owl:targetIndividual   :Mary .
```

Hierarhija lastnosti

- Enako kot razredi imajo lastnosti lahko podlastnosti

```
SubObjectPropertyOf( :hasWife :hasSpouse )
:hasWife rdfs:subPropertyOf :hasSpouse .
```

- Hirearhija lastnosti je značilna za jezike za predstavitev znanja

Omejitve domene in zaloge vrednosti lastnosti

- Enostavne omejitve domene in zaloge vrednosti.

```
ObjectPropertyDomain( :hasWife :Man )
ObjectPropertyRange( :hasWife :Woman )

:hasWife rdfs:domain :Man ;
         rdfs:range  :Woman .
```

Enakost in neenakost objektov

- Lahko definiramo, da sta dva objekta različna.

```
DifferentIndividuals( :John :Bill )
:John owl:differentFrom :Bill .
```

- Lahko definiramo, da sta dva objekta enaka.

```
SameIndividual( :James :Jim )
:James owl:sameAs :Jim.
```

Podatkovni tipi

- Lastnosti objektov se opiše z enostavno vrednostjo

```
DataPropertyAssertion( :hasAge :John "51"^^xsd:integer )
:John  :hasAge  51 .

NegativeDataPropertyAssertion(:hasAge :Jack
"53"^^xsd:integer )
[]  rdf:type     owl:NegativePropertyAssertion ;
    owl:sourceIndividual   :Jack ;
    owl:assertionProperty  :hasAge ;
    owl:targetValue        53 .
```

Kompleksni gradniki OWL

- Kompleksni razredi
- Omejitve lastnosti
- Vrednostne omejitve
- Naštevni tipi (razredi)

Kompleksni razredi (1)

- Logični konstruktorji za definicijo razredov
  - konstruktorji: unija, presek, negacija

```
EquivalentClasses(
   :Mother
   ObjectIntersectionOf( :Woman :Parent )
 )
:Mother  owl:equivalentClass  [
   rdf:type             owl:Class ;
   owl:intersectionOf  ( :Woman :Parent )
 ] .
```

Kompleksni razredi (2)

- Primer definicije z unijo:

```
EquivalentClasses(
   :Parent
   ObjectUnionOf( :Mother :Father )
 )
:Parent  owl:equivalentClass  [
   rdf:type     owl:Class ;
   owl:unionOf  ( :Mother :Father )
 ] .
```

Kompleksni razredi (3)

- Primer definicije s komplementom:

```
EquivalentClasses(
  :ChildlessPerson
  ObjectIntersectionOf(
    :Person
    ObjectComplementOf( :Parent )
  )
)
```

Opisna logika:

$:\mathtt{ChildlessPerson} \equiv :\mathtt{Person} \sqcap \neg :\mathtt{Parent}$

Kompleksni razredi (4)

- Primer definicije vsebovanja
- Obvezna lastnost vendar ne zadostna za definicijo.

```
SubClassOf(
   :Grandfather
   ObjectIntersectionOf( :Man :Parent )
 )
```

Opisna logika:

$:\mathrm{Grandfather} \sqsubseteq :\mathrm{Man} \sqcap :\mathrm{Parent}$

Kompleksni razredi (5)

- Kompleksni razredi se lahko pojavijo na mestu razredov
- Primer definicije objekta, ki je oseba brez otrok

```
ClassAssertion(
   ObjectIntersectionOf(
     :Person
     ObjectComplementOf( :Parent )
   )
   :Jack
 )
```

Omejitve lastnosti (1)

- Uporaba eksistenčne kvantifikacije za definicijo starša

```
EquivalentClasses(
   :Parent
   ObjectSomeValuesFrom( :hasChild :Person )
 )
:Parent  owl:equivalentClass  [
   rdf:type              owl:Restriction ;
   owl:onProperty        :hasChild ;
   owl:someValuesFrom    :Person
 ] .
```

- Nepopolno znanje: Nina pravi, da je Janez starš.

Omejitve lastnosti (2)

- Omejitev lastnosti z univerzalno kvantifikacijo

```
EquivalentClasses(
    :HappyPerson
    ObjectAllValuesFrom( :hasChild :HappyPerson )
 )
```

- Samo-referenčni stavek ali rekurzivna definicija
- Eksistenčna kvantifikacija je bolj pogosta od univerzalne v naravnem jeziku
- Vsi, ki nimajo otrok, so srečni?

Omejitve lastnosti (3)

- Popravimo definicijo:
  – Srečni so tisti, ki imajo vsaj enega srečnega otroka

```
EquivalentClasses(
    :HappyPerson
    ObjectIntersectionOf(
      ObjectAllValuesFrom( :hasChild :HappyPerson )
      ObjectSomeValuesFrom( :hasChild :HappyPerson )
    )
 )
```

- Primer prikaže definicijo razreda z omejitvijo lastnosti

Omejitve lastnosti (4)

- Med „vsem“ in „enim“: vrednostne omejitve
- John je oče največ štirih otrok, ki so starši

```
ClassAssertion(
   ObjectMaxCardinality( 4 :hasChild :Parent )
   :John
 )
```

- John ima lahko več otrok, ki niso starši

Omejitve lastnosti (5)

- John je oče vsaj dveh otrok, ki so starši:

```
ClassAssertion(
   ObjectMinCardinality( 2 :hasChild :Parent )
   :John
 )
```

- Če pa vemo natančno koliko otrok ima John:

```
ClassAssertion(
   ObjectExactCardinality( 3 :hasChild :Parent )
   :John
 )
```

Omejitve lastnosti (6)

- Lahko pa tudi omejimo samo število in ne tudi razred zaloge vrednosti
- John ima pet otrok

```
ClassAssertion(
   ObjectExactCardinality( 5 :hasChild )
   :John
 )
```

Naštevni tipi

- Razred je preprosto definiran tako, da naštejemo elemente

```
EquivalentClasses(
    :MyBirthdayGuests
    ObjectOneOf( :Bill :John :Mary)
 )
```

- Izjava pove več kot samo o članstvu elementov. Pove, da so našteti člani edini člani.
- Takšen razred imenujemo zaprti razredi ali naštevni razredi

Napredna uporaba lastnosti

- Značilnosti lastnosti
- Verige lastnosti
- Ključi

Značilnosti lastnosti (1)

- Inverzne lastnosti:
  - Lastnost lahko definiramo kot inverzno lastnost obstoječe lastnosti

```
InverseObjectProperties( :hasParent :hasChild )
```

- Lahko direktno uporabimo inverz lastnosti

```
EquivalentClasses(
    :Orphan
    ObjectAllValuesFrom(
      ObjectInverseOf( :hasChild )
      :Dead
    )
)
```

Značilnosti lastnosti (2)

- V nekaterih primerih smer lastnosti ni pomembna – lastnost je simetrična

```
SymmetricObjectProperty( :hasSpouse )
```

- Nekatere lastnosti ne smejo biti uporabljene v obratni smeri – asimetične lastnosti
  - A P B. => not B P A. !

```
AsymmetricObjectProperty( :hasChild )
```

Značilnosti lastnosti (3)

- Tuje lastnosti:
  - Dve lastnosti sta tuji, če ne nastopata hkrati: ne obstajata takšni dve osebi za kateri veljajo obe lastnosti hkrati

```
DisjointObjectProperties( :hasParent :hasSpouse )
```

Značilnosti lastnosti (4)

- Refleksivne lastnosti:
  - Lastnost objekta s samim seboj: A P A.
  - Vsak je v sorodu s samim sabo

```
ReflexiveObjectProperty( :hasRelative )
```

- Irefleksivne lastnosti:
  - Objekt ne sme biti v relaciji s seboj

```
IrreflexiveObjectProperty( :parentOf )
```

Značilnosti lastnosti (5)

- Funkcijske lastnosti:
  - Relacija definirana z lastnostjo je funkcija

```
FunctionalObjectProperty( :hasHusband )
```

- Lahko izrazimo tudi to, da je inverzna relacija lastnosti funkcija

```
InverseFunctionalObjectProperty( :hasHusband )
```

Značilnosti lastnosti (6)

- Tranzitivne lastnosti:
  - Eksplicitno povemo, da je lastnost tranzitivna
  - Definiramo osnovne relacije z lastnostjo npr. naslednik
  - Tranzitivne relacije izpelje sistem sam

```
TransitiveObjectProperty( :hasAncestor )
```

Verige lastnosti

- Tranzitivne lastnosti so en primer verig lastnosti
- Verigo lahko definiramo eksplicitno
  - Dve lastnosti :hasParent => :hasGrandparent

```
SubObjectPropertyOf(
   ObjectPropertyChain( :hasParent :hasParent )
   :hasGrandparent
 )
```

Ključi

- Kolekcija objektov kot je npr. razred se lahko indeksira

```
HasKey( :Person () ( :hasSSN ) )
```

- Gradnik SUPB

Kompleten primer

- Primeri so shranjeni v eni datoteki
- OWL-Sample.abs

Example: people & ...

- Sean Bechhofer, University of Manchester
  - Origin: Peter F. Patel-Schneider: people & pets ?
  - File: OWL-people-pets.abs
- Owl 1 syntax!
  - Owl 1 - close to DL
  - Owl 2 - KR language
- Main classification of examples:
  - Class inferences
  - Instant inferences

Class inferences

- Bus drivers are drivers
  - A bus driver is a person that drives a bus.
  - A bus is a vehicle.
  - A bus driver drives a vehicle, so must be a driver.

```
Class(a:bus_driver complete intersectionOf(a:person
  restriction(a:drives someValuesFrom (a:bus))))
Class(a:driver complete intersectionOf(a:person
  restriction(a:drives someValuesFrom (a:vehicle))))
Class(a:bus partial a:vehicle)
```

The subclass is inferred due to subclasses being used in existential quantification.

Class inferences

- Cat Owners like Cats
  - Cat owners have cats as pets.
  - has pet is a subproperty of likes, so anything that has a pet must like that pet.
  - Cat owners must like a cat.

```
Class(a:cat_owner complete intersectionOf(a:person
  restriction(a:has_pet someValuesFrom (a:cat))))

SubPropertyOf(a:has_pet a:likes)

Class(a:cat_liker complete intersectionOf(a:person
  restriction(a:likes someValuesFrom (a:cat))))
```

The subclass is inferred due to a subproperty assertion.

Class inferences

- Drivers are Grown Ups
  - Drivers are defined as persons that drive cars (complete definition)
  - We also know that drivers are adults (partial definition)
  - So all drivers must be adult persons (e.g. grownups)

```
Class(a:driver complete intersectionOf(a:person
  restriction(a:drives someValuesFrom (a:vehicle))))

Class(a:driver partial a:adult)

Class(a:grownup complete intersectionOf(a:adult
                                        a:person)
```

An example of axioms being used to assert additional necessary information about a class. We do not need to know that a driver is an adult in order to recognize one, but once we have recognized a driver, we know that they must be adult.

```
Class(a:sheep partial
  a:animal
  restriction(a:eats allValuesFrom (a:grass)))

Class(a:grass partial a:plant)

DisjointClasses(unionOf(restriction(a:part_of someValuesFrom (a:animal))
                        a:animal)
                unionOf(a:plant restriction(a:part_of someValuesFrom (a:plant))))

Class(a:vegetarian complete intersectionOf(
    restriction(a:eats allValuesFrom
                (complementOf(restriction(a:part_of someValuesFrom (a:animal)))))
    restriction(a:eats allValuesFrom (complementOf(a:animal))) a:animal))
```

- Sheep are Vegetarians
  - Sheep only eat grass.
  - Grass is a plant.
  - Plants and parts of plants are disjoint from animals and parts of animals.
  - Vegetarians only eat things which are not animals or parts of animals.

Note the complete definition, which means that we can recognise when things are vegetarians.

```
Class(a:giraffe partial a:animal
  restriction(a:eats allValuesFrom (a:leaf)))

Class(a:leaf partial restriction(a:part_of someValuesFrom (a:tree)))
Class(a:tree partial a:plant)

DisjointClasses(unionOf(restriction(a:part_of someValuesFrom (a:animal))
                        a:animal)
                unionOf(a:plant
                        restriction(a:part_of someValuesFrom (a:plant))))

Class(a:vegetarian complete intersectionOf(
    restriction(a:eats
                allValuesFrom (complementOf(restriction(a:part_of
                                                someValuesFrom (a:animal)))))
    restriction(a:eats allValuesFrom (complementOf(a:animal))) a:animal))
```

- Giraffes are Vegetarians
  - Giraffes only eat leaves
  - Leaves are parts of trees, which are plants
  - Plants and parts of plants are disjoint from animals and parts of animals
  - Vegetarians only eat things which are not animals or parts of animals

Similar to the previous example with the additional inference provided by the existential restriction in the definition of leaf

Class inferences

```
Class(a:old_lady complete
  intersectionOf(a:person a:female a:elderly))

Class(a:old_lady partial intersectionOf(
  restriction(a:has_pet allValuesFrom (a:cat))
  restriction(a:has_pet someValuesFrom (a:animal))))

Class(a:cat_owner complete intersectionOf(a:person
  restriction(a:has_pet someValuesFrom (a:cat))))
```

- Old Ladies own Cats
  - Old ladies must have a pet.
  - All pets that old ladies have must be cats.
  - An old lady must have a pet that is a cat.

An example of the interaction between an existential quantification (asserting the existence of a pet) and a universal quantification (constraining the types of pet allowed).

Instance inferences

- Walt loves animals

```
Individual(a:Walt type(a:person)
  value(a:has_pet a:Huey)
  value(a:has_pet a:Louie)
  value(a:has_pet a:Dewey))

Individual(a:Huey type(a:duck))
Individual(a:Dewey type(a:duck))
Individual(a:Louie type(a:duck))

DifferentIndividuals(a:Huey a:Dewey a:Louie)

Class(a:animal_lover complete
         intersectionOf(a:person restriction(a:has_pet
                                    minCardinality(3))))
```

  - Walt has pets Huey, Dewey and Louie
  - Huey Dewey and Louie are all distinct individuals
  - Walt has at least three pets and is thus an animal lover.

Note that in this case, we don't actually need to include person in the definition of animal lover (as the domain restriction will allow us to draw this inference).

Instance inferences

- Pete is a Person, Spike is an Animal
  - Spike is the pet of Pete
  - So Pete has pet Spike
  - Pete must be a Person
  - Spike must be an Animal

```
Individual(a:Spike type(owl:Thing)
  value(a:is_pet_of a:Pete))

Individual(a:Pete
  type(owl:Thing))

ObjectProperty(a:has_pet
                domain(a:person) range(a:animal))

ObjectProperty(a:is_pet_of inverseOf(a:has_pet))
```

Here we see an interaction between an inverse relationship and domain and range constraints on a property.

Instance inferences

- Tom is a Cat

```
Individual(a:Minnie type(a:female) type(a:elderly)
value(a:has_pet a:Tom))

Individual(a:Tom type(owl:Thing))

ObjectProperty(a:has_pet domain(a:person) range(a:animal))

Class(a:old_lady complete
  intersectionOf(a:person a:female a:elderly))

Class(a:old_lady partial intersectionOf(
  restriction(a:has_pet allValuesFrom (a:cat))
  restriction(a:has_pet someValuesFrom (a:animal))))
```

  - Minnie is elderly, female and has a pet, Tom
  - Minnie must be a person
  - Minnie is an old lady
  - All Minnie's pets must be cats

Here the domain restriction gives us additional information which then allows us to infer a more specific type. The universal quantification then allows us to infer information about the role filler.

Instance inferences

```
Individual(a:Daily_Mirror type(owl:Thing))
Individual(a:Mick type(a:male)
 value(a:drives a:Q123_ABC)
 value(a:reads a:Daily_Mirror))
Individual(a:Q123_ABC type(a:van) type(a:white_thing))

Class(a:van partial a:vehicle)
Class(a:driver partial a:adult)
Class(a:driver complete
    intersectionOf(restriction(a:drives
                        someValuesFrom(a:vehicle))
                    a:person))


Class(a:white_van_man complete
 intersectionOf(a:man restriction(a:drives
       someValuesFrom (intersectionOf(a:van a:white_thing)))))
Class(a:white_van_man partial
           restriction(a:reads allValuesFrom (a:tabloid)))
```

- The Daily Mirror is a Tabloid
  - Mick drives a white van
  - Mick must be a person and an adult, so he is a man
  - Mick is a man who drives a white van, so he's a white van man
  - A white van man only reads tabloids, and Mick reads the Daily Mirror, thus the Daily Mirror must be a tabloid

Here we see interaction between complete and partial definitions plus a universal quantification allowing an inference about a role filler.

OWL pogled na življenje

- OWL ni sistem za delo s podatkovnimi bazami
  - Kaj je potem ?
  - Dodamo nove operacije v SQL DBMS ?
- Razlogi za samostojen sistem
  - Ni zahtev po tem, da so edine lastnosti objekta tiste, ki so zapisane.
  - Ni predpostavke o tem, da je vse znano.
  - Razredi in lastnosti imajo lahko več “definicij”.
  - Stavki o posameznikih niso nujno skupaj v istem dokumentu.
  - Stroj za izvajanje mehanizmov sklepanja
  - Novi aspekti podatkovnih modelov

Kreiranje ontologij

- OWL opisuje ontologije
  - Ontologija določa kaj je zanimivo znotraj dane domene in kako so podatki strukturirani
  - OWL ontologija je samo zbirka informacij
    - podatki o razredih in lastnostih
- Ontologija lahko vključi (import) podatke iz drugih ontologij
  - Ontology([name] owl:imports(<name>) ...)

Gradnja ontologij

- Definiraj kako naj domena izgleda
  - Definiraj razrede in lastnosti v dani domeni
  - Definiraj domene in zaloge vrednosti lastnosti
  - Definiraj karakteristike razredov
  - Dodaj individualne objekte in relacije med objekti
  - Iteriraj dokler opis ni spejemljiv
  - Zloži vse to v ontologijo
- Izgradi OWL ontologijo
  - Vprašaj se ali je ontologija konsistentna
  - Vprašaj se če so razredi koherentni

Gradnja ontologij

- Naseli svet (za določeno opravilo)
  - Določi individualne objekte potrebne za delo
  - Definiraj razmerja med individualnimi objekti
  - Določi omejtive individualnih objektov
- Napiši podatke v OWL sintaksi
  - Vprašaj se če so podatki konsistentni
  - Vprašaj se kakšne podatke se da izpeljati

Literatura

- http://www.w3.org/TR/owl2-overview/
- http://www.w3.org/TR/2012/REC-owl2-primer-20121211/
- http://owl.man.ac.uk/2003/why/latest/
- Jos de Bruijn, Semantics Web Technologies, Course at Free University of Bolzano, 2008.
- Peter F. Patel-Schneider, OWL-Tutorial, Bell Labs Research, Lucent Technologies