Description Logics & OWL

Hideaki Takeda

takeda@nii.ac.jp

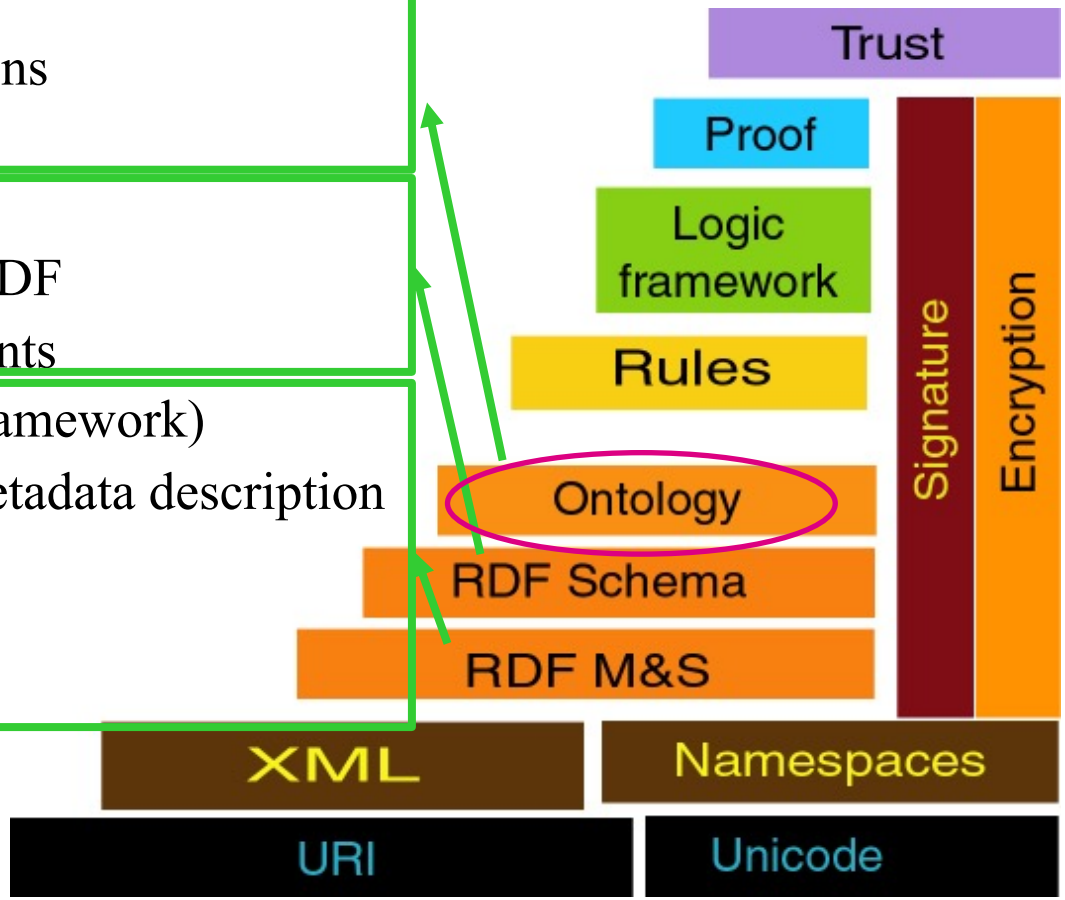
National Institute of Informatics

Types of Ontologies

- Upper (top-level) ontology vs. Domain ontology
  - Upper Ontology: A common ontology throughout all domains
  - Domain Ontology: An ontology which is meaningful in a specific domain
- Object ontology vs. Task ontology
  - Object Ontology: An ontology on “things” and “events”
  - Task Ontology: An ontology on “doing”

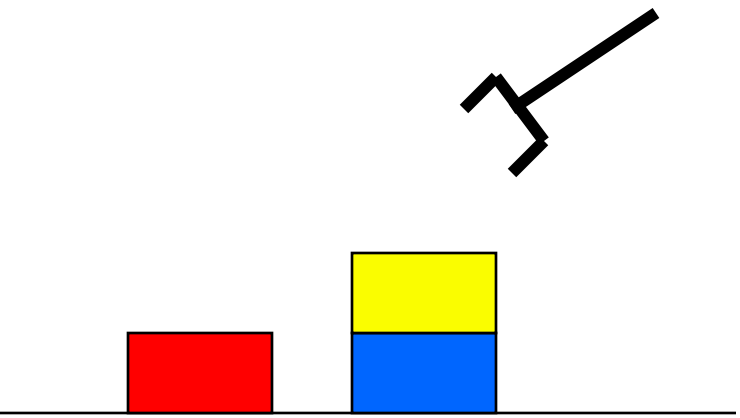
A Layer model for Semantic Web

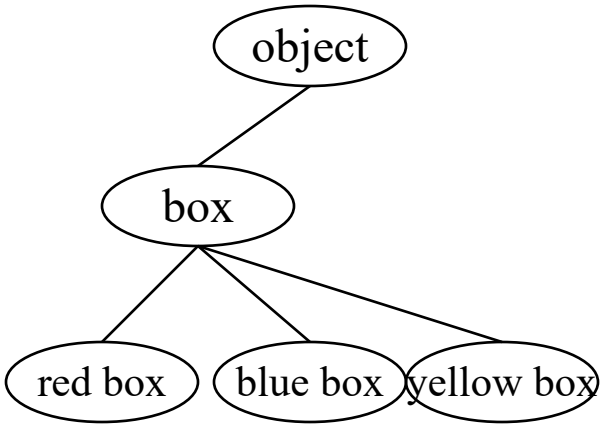
- OWL
  - General concept description language
    - Logical consistency
    - Various class expressions
    - Various constraints
- RDF Schema
  - Addition of "concept" to RDF
    - class-subclass, constraints
- RDF (Resource Description Framework)
  - The primitive model for metadata description
    - SVO model
    - Entity-Relation Model
    - Semantic net



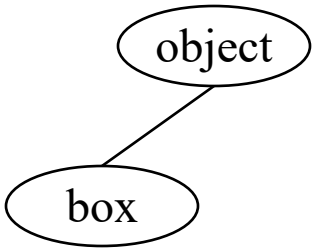
Tim Berners-Lee http://www.w3.org/2002/Talks/09-lcs-sweb-tbl/

Conceptualization



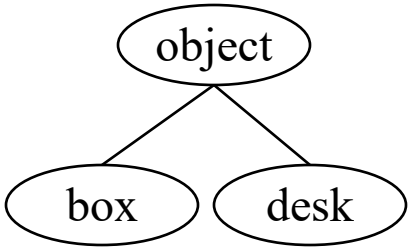


on_desk(A)
on(A, B)
put(A,B)



on_desk(A)
on(A, B)
put(A,B)

box
color: {red, blue, yellow}



on(A/box, B/object)
put(A/box,B/object)

box
color: {red, blue, yellow}

There are many possible ways to conceptualize the target world

Trade off between generality and efficiency

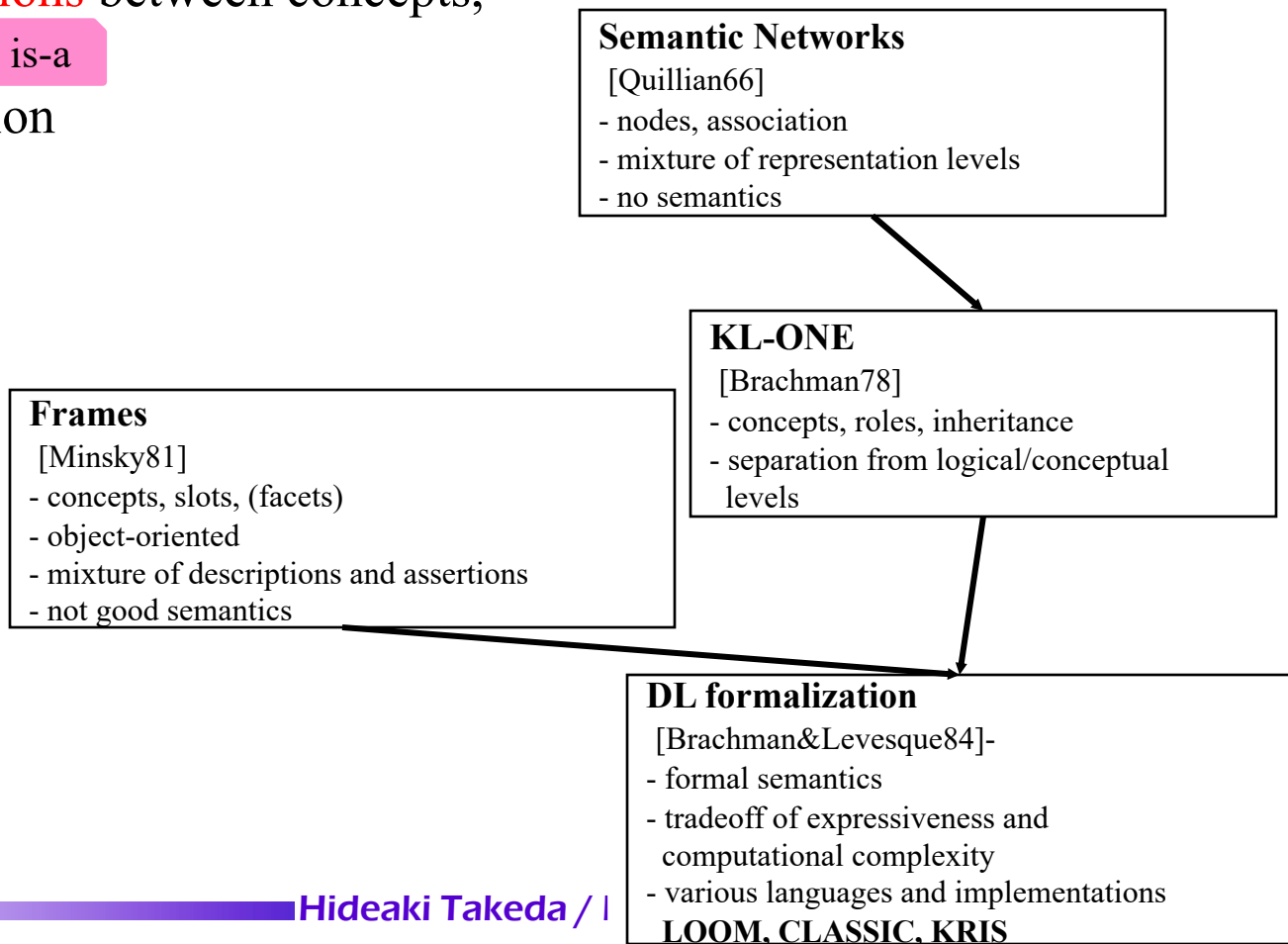
Ontology in Information System

An ontology is an explicit specification of a conceptualization
[Gruber]

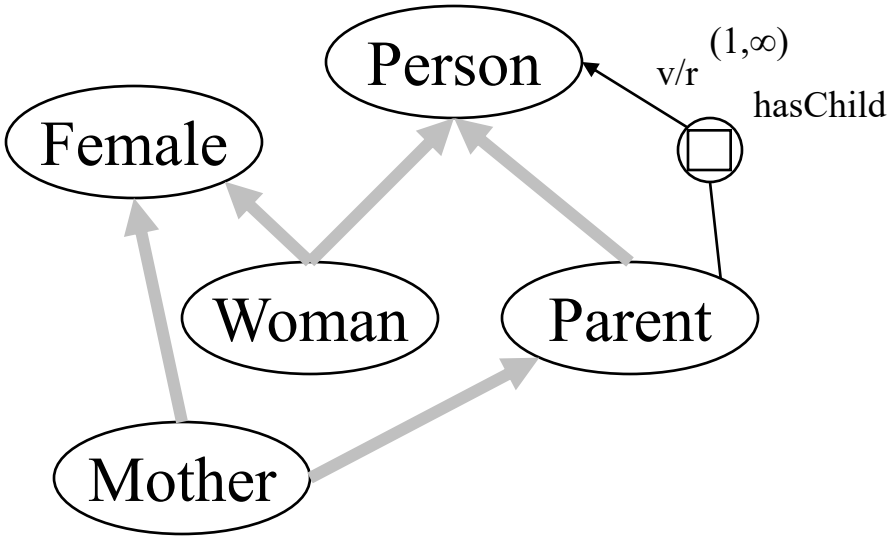
- An **ontology** is an explicit specification of a conceptualization. The term is borrowed from philosophy, where an Ontology is a systematic account of Existence. For AI systems, what "exists" is that which can be represented. When the knowledge of a domain is represented in a declarative formalism, the set of objects that can be represented is called the universe of discourse. This set of objects, and the describable relationships among them, are reflected in the representational vocabulary with which a knowledge-based program represents knowledge. Thus, in the context of AI, we can describe the ontology of a program by defining a set of representational terms. In such an ontology, definitions associate the names of entities in the universe of discourse (e.g., classes, relations, functions, or other objects) with human-readable text describing what the names mean, and formal axioms that constrain the interpretation and well-formed use of these terms. Formally, an ontology is the statement of a logical theory.

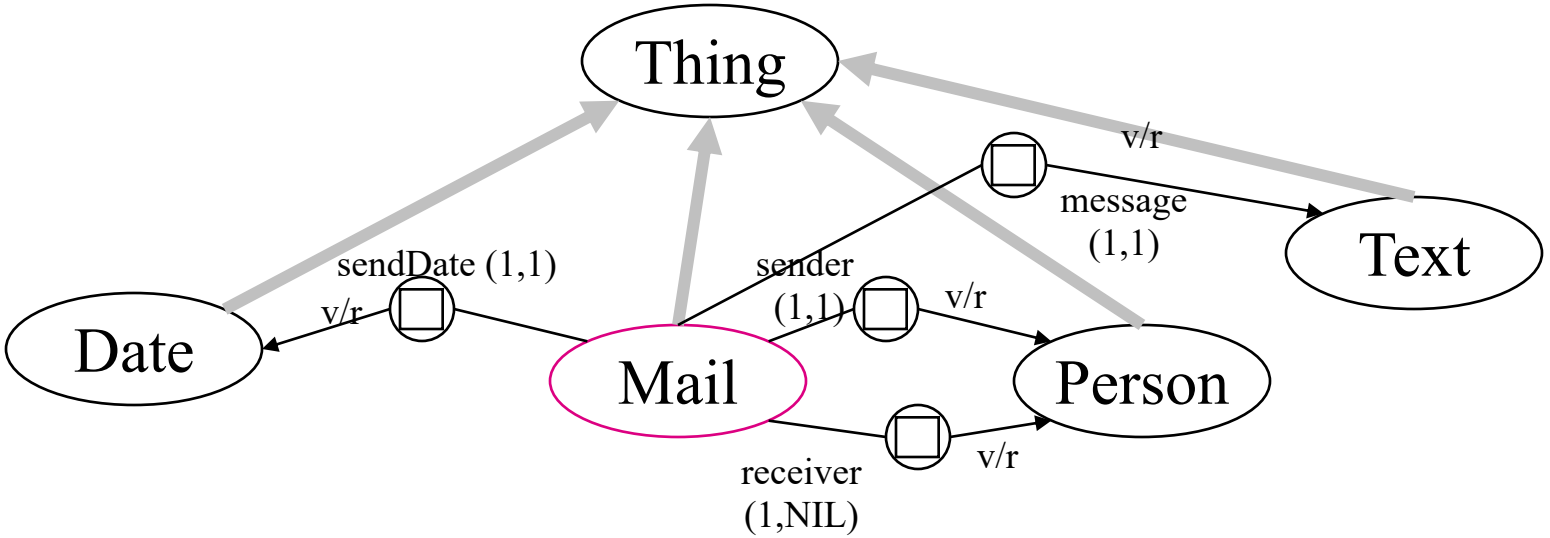
Description Logic

- What is Description Logic?
  - Representation for structured knowledge level
    - Concepts, relations between concepts, inheritance is-a
  - Logical formalization
- History

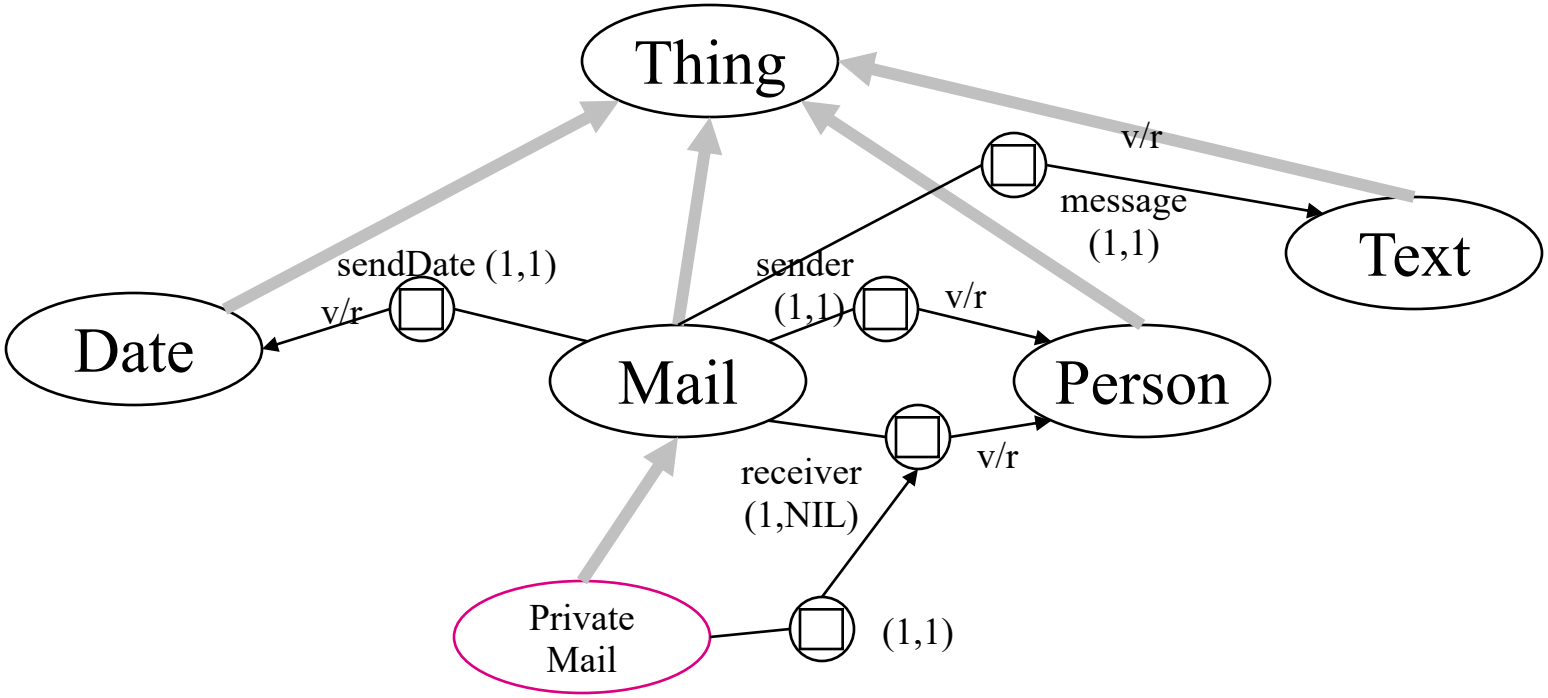


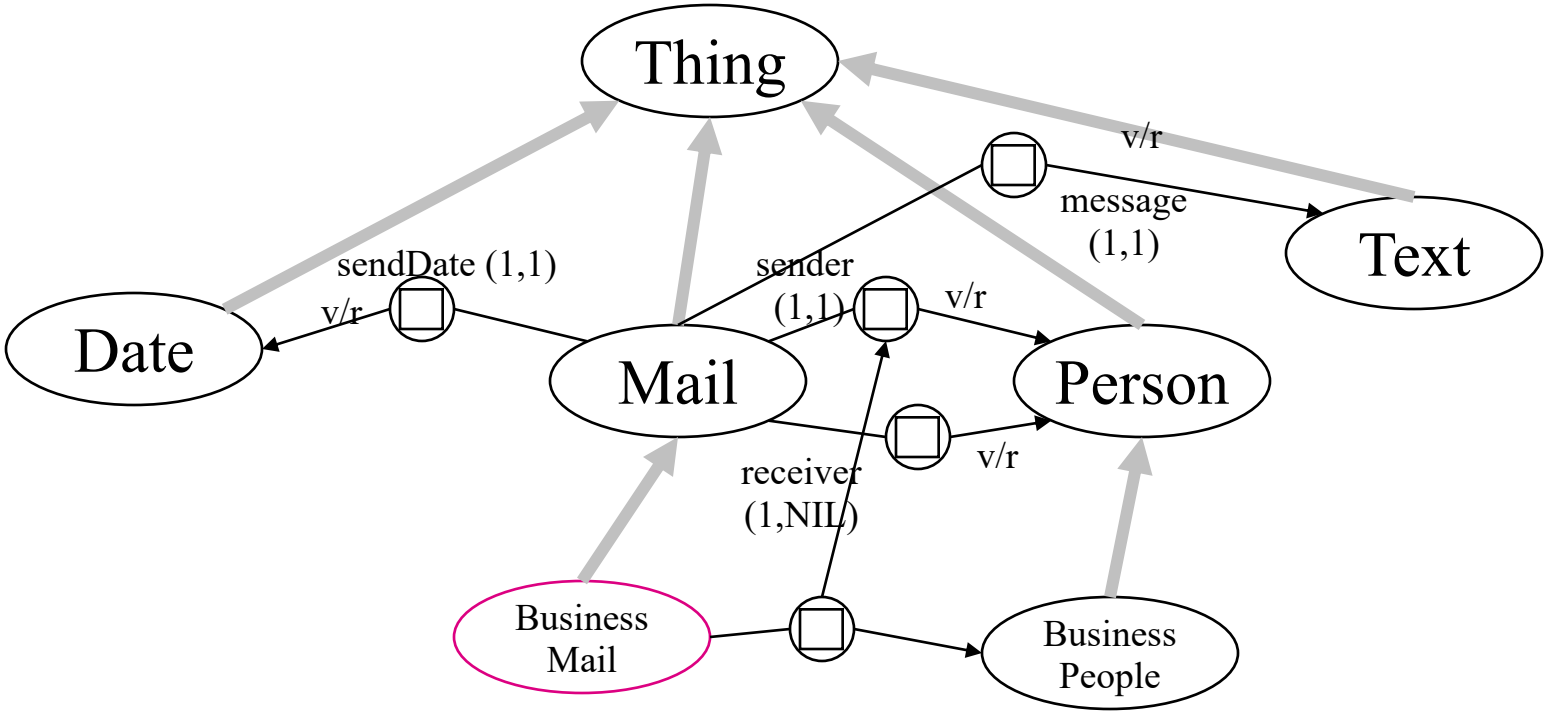
Concept

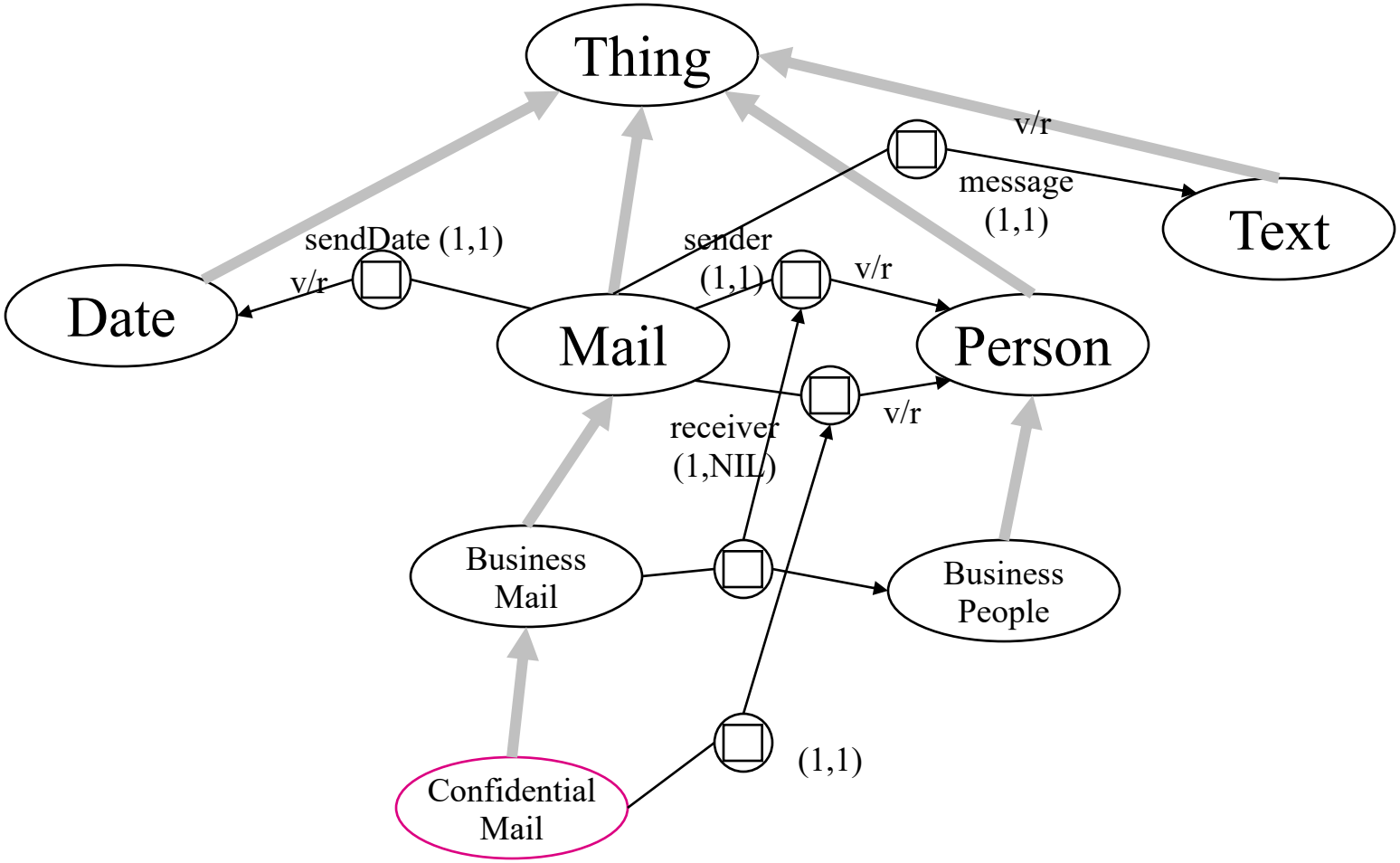




$$\mathrm{Mail} \sqsubseteq \mathrm{Thing} \sqcap \forall \mathrm{sendDate}.\mathrm{Date} \sqcap \forall \mathrm{sender}.\mathrm{Person} \sqcap \forall \mathrm{receiver}.\mathrm{Person} \sqcap \forall \mathrm{message}.\mathrm{Text}$$


Thing
Date
Mail
Person
Text
Private Mail
sendDate (1,1)
v/r
sender
(1,1)
v/r
v/r
message
(1,1)
receiver
(1,NIL)
v/r
(1,1)


Thing
Date
Mail
Person
Text
Business Mail
Business People
sendDate (1,1)
v/r
sender
(1,1)
v/r
v/r
message
(1,1)
v/r
receiver
(1,NIL)


Thing
v/r
message
(1,1)
Text
sendDate (1,1)
v/r
Date
sender
(1,1)
v/r
Mail
Person
receiver
(1,NIL)
v/r
Business
Mail
Business
People
(1,1)
Confidential
Mail

Elements of Description Logic

- Concepts: entities and classes
  - Person
  - *Unary predicates in FOL*
    - {x | Person(x)}, λx.Person(x)
- Roles: properties and relations
  - haschild
  - *2-ary predicates in FOL*
    - {x, y | hasChild(x, y)}
- Constructors for concept expression: conjunction(⊓), union(⊔)
  - Person ⊓ ∃hasChild.Female
  - {x | Person(x) ∧ ∃y.haschild(x, y) ∧ Female(y)}
- Individuals: instances of concepts, co-reference to objects in the world
  - Ex, Takeda, s1234

Constructors for concept expression

$\mathcal{FL}^-$ $\mathcal{AL}^\star$

Constructors	Syntax	Semantics
Concept	C	$C^I \subseteq \Delta^I$
Role name	R	$R^I \subseteq \Delta^I \times \Delta^I$
Conjunction	$C \sqcap D$	$C^I \cap D^I$
Value restriction	$\forall R.C$	$\{x \in \Delta^I \mid \forall y.(x,y) \in R^I \Rightarrow y \in C^I\}$
Existential quantification	$\exists R$	$\{x \in \Delta^I \mid \exists y.(x,y) \in R^I\}$
Negation	$\neg C$	$\Delta^I \setminus C^I$
Top	$\top$	Δ^I
Bottom	$\bot$	$\varnothing$
Disjunction	$C \sqcup D$	$C^I \cup D^I$
Existential restriction	$\exists R.C$	$\{x \in \Delta^I \mid \exists y.(x,y) \in R^I \wedge y \in C^I\}$
Number restriction	$(\geqslant n\ R)$	$\{x \in \Delta^I \mid \lvert\{y \mid y.(x,y) \in R^I\}\rvert \geq n\}$
Collection of individuals	$\{a_1, a_2, \ldots\}$	$\{a_1^I, a_2^I, \ldots\}$

Variety of Description Logics

- $\mathcal{ALC}$ is the smallest propositionally closed DL
  - – Concept operators: ⋂, ⋃, ¬, ∀, ∃
  - – No role operators (only atomic roles)
  - – Concept axioms: ⊑, ≡
  - – No role axioms

http://www.cs.ox.ac.uk/ian.horrocks/Seminars/download/oracle-seminar-20101210.pdf

Variety of Description Logics

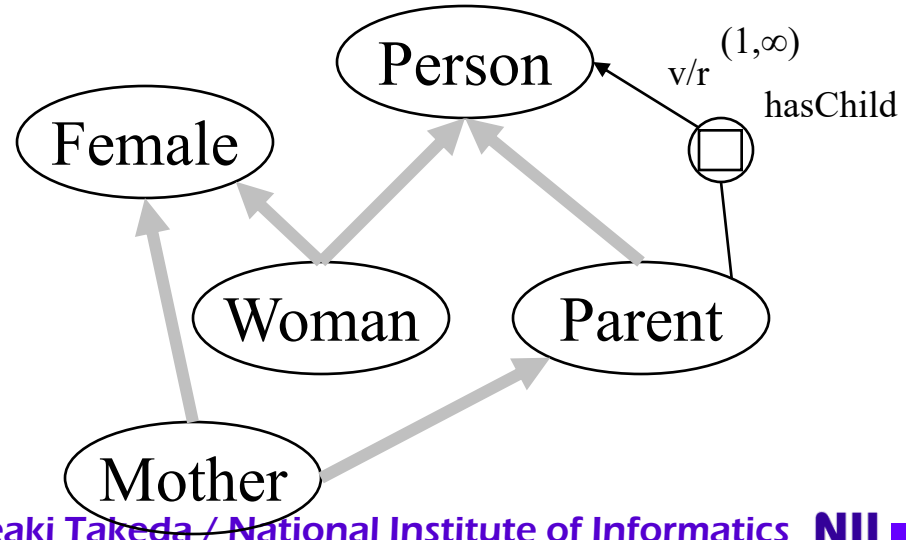
- $\mathcal{S}$ used for $\mathcal{ALC}$ extended with (role) transitivity axioms
- Additional letters indicate various extensions, e.g.:
  - $\mathcal{H}$ for role hierarchy (e.g., hasDaughter $\sqsubseteq$ hasChild)
  - $\mathcal{R}$ for role box (e.g., hasParent $\sqcap$ hasBrother $\sqsubseteq$ hasUncle)
  - $\mathcal{O}$ for nominals/singleton classes (e.g., {Italy})
  - $\mathcal{I}$ for inverse roles (e.g., isChildOf $\equiv$ hasChild$^-$)
  - $\mathcal{N}$ for number restrictions (e.g., $\geqslant$2 hasChild, $\leqslant$3 hasChild)
  - $\mathcal{Q}$ for qualified number restrictions (e.g., $\geqslant$ 2 hasChild.Doctor)
  - $\mathcal{F}$ for functional number restrictions (e.g., $\leqslant$ 1 hasMother)
- E.g., $\mathcal{SHIQ}$ = $\mathcal{S}$ + role hierarchy + inverse roles + QNRs
- $\mathcal{SHOIN}$ = $\mathcal{S}$ + role hierarchy + nominals + inverse roles + NR
- $\mathcal{SHIF}$ = $\mathcal{S}$ + role hierarchy + inverse roles + FNR

Semantics

- Interpretation I consists of the domain of discourse Δ^I (non empty set) and interpretation function ^{I}
  - ^{I} maps
    - Concept C to $C^I \subseteq \Delta^I$
    - Role R to $R^I \subseteq \Delta^I \times \Delta^I$
- A model for C is an interpretation where C^I is not empty
- A concept C is satisfiable if it has a model for C

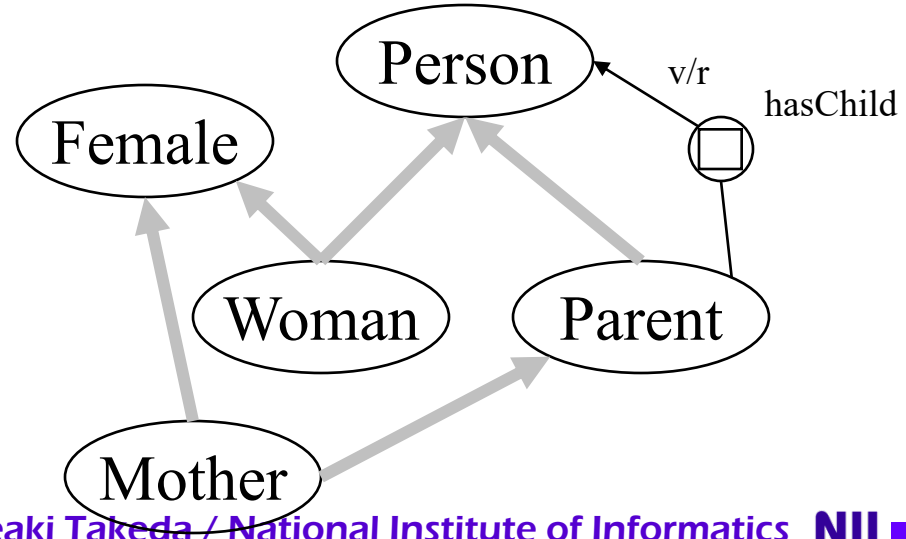
TBox & ABox

- Knowledge Base Σ = <TBox, ABox>
- TBox
  - Conceptual or terminological knowledge
  - Intensional knowledge
  - General knowledge
  - Examples
    - Woman ≡ Person ⊓ Female
    - Parent ≡ Person ⊓ ∃ hasChild.Person ⊓ ∀ hasChild.Person
    - Mother ≡ Female ⊓ Parent
- ABox
  - Instance or assertional knowledge
  - Extensional knowledge
  - Knowledge in a situaion
  - Examples
    - Woman(Sazae)
    - hasChild(Sazae, Tara)



Reasoning

- Subsumption
  - Concept satisfiablity: $\Sigma \not\models C \equiv \bot$
  - Concept Subsumption: $\Sigma \models C \sqsubseteq D$ or $\Sigma \models C \sqcap \neg D \equiv \bot$
  - Inconsistency:
  - Ex.)
    - Mother $\sqsubseteq$ Woman



Structural Subsumption Algorithm

- Normalization (Conjunctive normal form)
  - Mother= Person ⊓ Female ⊓ ∀hasChild.Person
  - SMother= Person ⊓(∃hasChild ⊓ ∀hasChild.Person) ⊓ Female ⊓ ∀ hasChild.Student
    = Person ⊓ Female ⊓ ∃ hasChild ⊓ ∀ hasChild.(Person ⊓ Student)
- Compare each term
  - C subsumes D if each term $C_i \in C$ satisfies:
    - If C_i is atomic or ∃R, then there is D_j with $D_j=C_i$
    - If C_i is ∀R.C', then there is a D_j with D_j= ∀R.D' and C' subsumes D'
- Liner complexity and sound
- Complete only for $\mathcal{FL}^-$

Tableau algorithms

- Check satisfiability of concept descriptions
  - Assume an instance ***b*** which satisfies all the descriptions
  - Then check whether this assumption turns out impossible
    - The assumption is wrong -> not satisfiable
- For NNF (Negation Nomarl Form)
  - Negation appears only just before concepts
- Completion rules
  - Adding constraints by interpreting terms
  - Derive contradiction

Tableau algorithms

- Completion rules
  - $\sqcap$ -rule:
    - Condition: S contains $(C \sqcap D)(x)$ and does not contains both $C(x)$ and $D(x)$
    - Action: $S' = S \sqcup \{C(x), D(x)\}$
  - $\sqcup$ -rule:
    - Condition: S contains $(C \sqcup D)(x)$ and neither $C(x)$ nor $D(x)$
    - Action: $S' = S \sqcup \{C(x)\}$ or $S' = S \sqcup \{D(x)\}$
  - $\exists$-rule:
    - Condition: S contains $(\exists R.C)(x)$ and no individual z such that satisfy $C(z)$ and $R(x,z)$ in S
    - Action: $S' = S \sqcup \{C(y), R(x,y)\}$
  - $\forall$-rule:
    - Condition: S contains $(\forall R.C)(x)$ and $R(x,y)$, and does not contain $C(y)$
    - Action: $S' = S \sqcup \{C(y)\}$

Sumsumption by Tableau algorithms

- Unfold Tbox *T* to *T'*
  - Remove defined concepts by applying their definition in ***T***
    - Pick up B such as A≡B in ***T***
    - Replace A' such as A'≡B' with B' in B recursively
- Remove defined concepts by applying their definition in C ⊓¬D
    - Pick up A such as A≡B in C ⊓¬D
    - Replace A with B
- Transform C ⊑ D to C ⊓¬D
- Transform C ⊓ ¬D into NNF (Negation Normal Form)
- Check C ⊓ ¬D by Tableau algorithm

Tableau algorithms: An exmple

- Dmom ⊑ Mother ?
- T
  - Woman ≡ Person ⊓ Female
  - Parent ≡ Person ⊓ ∃ hasChild.Person ⊓ ∀ hasChild.Person
  - Mother ≡ Female ⊓ Parent
  - Dmom ≡ Woman ⊓ ∃ hasChild.Woman ⊓ ∀ hasChild.Woman
- T'
  - Woman ≡ Person ⊓ Female
  - Parent ≡ Person ⊓ ∃ hasChild.Person ⊓ ∀ hasChild.Person
  - Mother ≡ Female ⊓ Person ⊓ ∃ hasChild.Person ⊓ ∀ hasChild.Person
  - Dmom ≡ Person ⊓ Female ⊓ ∃ hasChild.(Person ⊓ Female) ⊓ ∀ hasChild. (Person ⊓ Female)
- Transform C ⊑ D to C ⊓ ¬D
  - Dmom ⊓ ¬Mother
- Remove defined concepts
  - Person ⊓ Female ⊓ ∃ hasChild. (Person ⊓ Female) ⊓ ∀ hasChild. (Person ⊓ Female) ⊓ ¬(Female ⊓ Person ⊓ ∃ hasChild.Person ⊓ ∀ hasChild.Person)
- NNF
  - Person ⊓ Female ⊓ ∃ hasChild. (Person ⊓ Female) ⊓ ∀ hasChild. (Person ⊓ Female) ⊓ (¬Female ⊔ ¬Person ⊔ ∀¬hasChild.Person ⊔ ∃ ¬hasChild.Person)

An example

- S0={x: Person ⊓ Female ⊓ ∃ hasChild. (Person ⊓ Female) ⊓ ∀ hasChild. (Person ⊓ Female) ⊓ (¬Female ⊔ ¬Person ⊔ ∀¬hasChild.Person ⊔ ∃ ¬hasChild.Person)}
  - ⊔-rule
  - S1=S0 ⊔ {x: ¬Female}= ⊥
  - S1''= S0 ⊔ {x: ¬Person}= ⊥
  - S1'''= S0 ⊔ {x: ∀¬hasChild.Person }$_{(1)}$
    - ⊓-rule
  - S2=S1''' ⊔ {x: Person ⊓ Female ⊓ ∃ hasChild. (Person ⊓ Female), x:∀ hasChild. (Person ⊓ Female)}
    - ∃-rule
  - S3=S2 ⊔ {y: Person ⊓ Female, (x,y): hasChild}
    - ⊓ -rule
  - S4=S3 ⊔{y:Person, y:Female}
    - ∀-rule (for (1))
  - S5= S4 ⊔ {y: ¬Person}= ⊥
  - S1''''= S0 ⊔ {x: ∃ ¬hasChild.Person }
    - ….

OWL(Web Ontology Language)

- More general knowledge representation
- Based on Description Logics
- Features
  - Class
    - Necessary condition / necessary and sufficient condition
    - Class expression:
      - Constraint by property
        - Like slot definition of a class
        - Type constraint (all/some), cardinality, typed cardinality
      - Logical operation of classes: union, intersection, negation
  - Property
    - Multiple ranges and domains
    - Specifying meta-property
  - Import of definitions

Class descriptions

- a class identifier (a URI reference)
- an exhaustive enumeration of individuals that together form the instances of a class
- a property restriction
- the intersection of two or more class descriptions
- the union of two or more class descriptions
- the complement of a class description

OWL: Instances

- Instance
  - Instance for a class or property
  - **`rdf:type`**

```
:Mary rdf:type :Person .
```

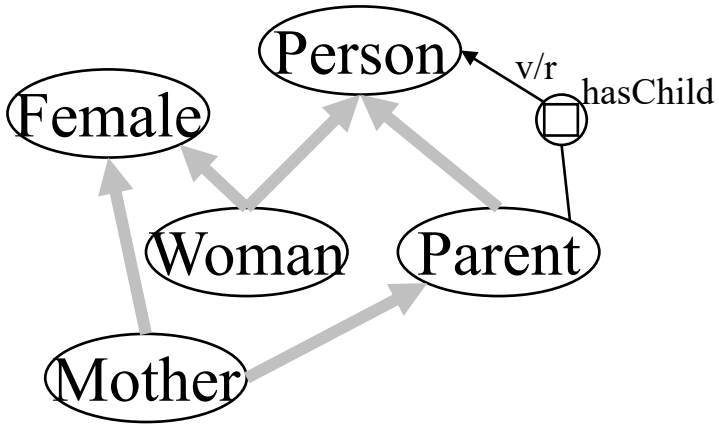
- Equality and Inequality of Individuals

```
:John owl:differentFrom :Bill .
:James owl:sameAs :Jim.
```

OWL

- Class axioms
  - **`rdfs:subClassOf`**
  - **`owl:equivalentClass`**
  - **`owl:intersectionOf`**

```
:Woman rdfs:subClassOf :Female .
:Human owl:equivalentClass :Person .
:Mother rdfs:subClassOf [
  rdf:type owl:Class ;
  owl:unionOf ( :Female :Parent)
```



OWL

- Object Properties

```
:John :hasWife :Mary .
```

- Data properties
  - `:John :hasAge 51 .`
- Property hierarchy

```
:hasWife rdfs:subPropertyOf :hasSpouse .
```

- Domain and rage

```
:hasWife rdfs:domain :Man ;
         rdfs:range  :Woman .
:hasAge rdfs:domain  :Person ;
        rdfs:range xsd:nonNegativeInteger .
```

OWL

- Property restriction
  - A special kind of class description. It describes an anonymous class, namely a class of all individuals that satisfy the restriction

OWL: Value constraints

- A restriction class to either a class description or a data range
- **owl:allValuesFrom**
  - All individuals must satisfy the specific condition (class or data range)

```
:HappyPerson rdf:type owl:Class ;
  owl:equivalentClass [
     rdf:type owl:Restriction ;
     owl:onProperty  :hasChild ;
     owl:allValuesFrom  :Happy
] .
```

- **owl:someValuesFrom**
  - At least one property value must satisfy the specific condition

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

- **owl:hasValue**
  - Constrain the value as the specific individual

```
:JohnsChildren owl:equivalentClass [
   rdf:type owl:Restriction ;
   owl:onProperty  :hasParent ;
   owl:hasValue  :John
] .
```

OWL: Cardinality constraints

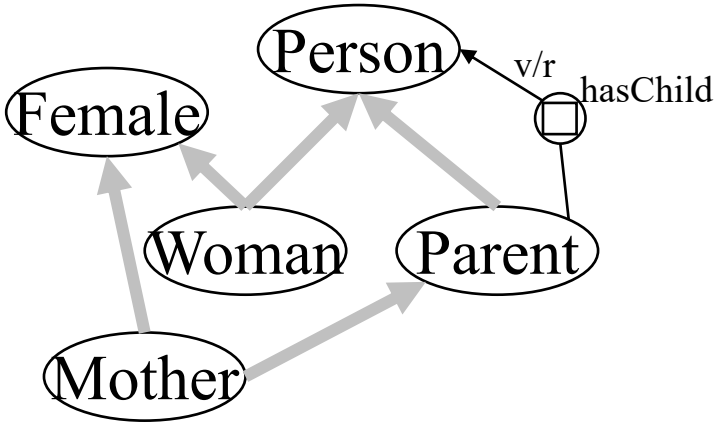
- to allow only a specific number of values
- **owl:maxQualifiedCardinality**
  - A restriction containing an **owl:maxQualifiedCardinality** constraint describes a class of all individuals that have at most N semantically distinct values

```
:John rdf:type [
  rdf:type owl:Restriction ;
  owl:maxQualifiedCardinality "4"^^xsd:nonNegativeInteger ;
  owl:onProperty  :hasChild ;
  owl:onClass  :Parent
] .
```

- **owl:minQualifiedCardinality**
- **owl:qualifiedCardinality**

OWL

- Cardinality Constraint




```
:Parent rdfs:quivalentClass [
  rdf:type owl:Restriction ;
  owl:onProperty  :hasChild ;
  ow:allValuesFrom :person ;
  owl:minQualifiedCardinality ”1"^^xsd:nonNegativeInteger ;
  owl:onClass  :person
] .
```

OWL Families

- OWL lite
  - $\mathcal{SHIF}(\mathcal{D})$ = $\mathcal{SHIF}$ + Datatype
- OWL DL
  - $\mathcal{SHOIN}(\mathcal{D})$ = $\mathcal{SHOIN}$ + Datatype
- OWL Full
  - OWL + RDFS
  - Eg.,
    - **Owl:class vs rdfs:class**
    - **Owl:ObjectTypeProperty and rdf:Property**

http://ragrawal.wordpress.com/2007/02/20/difference-between-owl-lite-dl-and-full/

Difference among Lite, DL, and Full

	Lite	DL	Full
Compatibility with RDF	Theoretically, no rdf document can be assumed to be compatible with OWL Lite	Theoretically,no rdf document can be assumed to be compatible with OWL DL	All valid rdf documents are OWL full
Restrictions on class definition	Requires separation of classes, instances, properties, and data values	Requires separation of classes, instances, properties, and data values	Classes can be instances or properties at the same time. For example, it is perfectly legal in OWL Full to have a "Fokker-100″ identifier which acts both as a class name (denoting the set of Fokker-100 airplanes flying around the world) and as an individual name (*e.g., an instance of the class AirplaneType).*
RDF Mixing	Restricts mixing of rdf and owl constructs	Restricts mixing of RDF and OWL constructs	Freely allows mixing of RDF and OWL constructs
Classes Descriptions	The only class description available in OWL lite is IntersectionOf	Classes can be described as UnionOf , ComplementOf, IntersectionOf, and enumeration Eg: class can be exhaustively defined by its instances. For example defining a class DaysOfWeek exhaustively by Sun, Mon, Tue, Wed, Thurs, Fri, Sat	Classes can be UnionOf, ComplementOf, IntersectionOf, and enumeration Eg: class can be exhaustively defined by its instances. For example defining a class DaysOfWeek exhaustively by Sun, Mon, Tue, Wed, Thurs, Fri, Sat
Cardinality	Cardinality: 0/1 MinCardinality:	Cardinality>= 0 MaxCardinality >= 0	

OWL Lite

- **RDF Schema Features:**
  - Class (Thing, Nothing)
  - rdfs:subClassOf
  - rdf:Property
  - rdfs:subPropertyOf
  - rdfs:domain
  - rdfs:range
  - Individual
- **(In)Equality:**
  - equivalentClass
  - equivalentProperty
  - sameAs
  - differentFrom
  - AllDifferent
  - distinctMembers
- **Property Characteristics:**
  - ObjectProperty
  - DatatypeProperty
  - inverseOf
  - TransitiveProperty
  - SymmetricProperty
  - FunctionalProperty
  - InverseFunctionalProperty
- **Property Restrictions:**
  - Restriction
  - onProperty
  - allValuesFrom
  - someValuesFrom
- **Restricted Cardinality:**
  - minCardinality (only 0 or 1)
  - maxCardinality (only 0 or 1)
  - cardinality (only 0 or 1)
- **Header Information:**
  - *Ontology*
  - *imports*
- **Class Intersection:**
  - IntersectionOf
- **Versioning:**
  - versionInfo
  - priorVersion
  - backwardCompatibleWith
  - incompatibleWith
  - DeprecatedClass
  - DeprecatedProperty
- **Annotation Properties:**
  - rdfs:label
  - rdfs:comment
  - rdfs:seeAlso
  - rdfs:isDefinedBy
  - AnnotationProperty
  - OntologyProperty
- **Datatypes**
  - xsd datatypes