Description Logic and OWL

- **Description Logic**
  - An important element of the Semantic Web
  - Has a well-defined semantics
    - A Concept is a non-empty set
    - Enables subsumption (subClassOf relations) to be computed
  - Tractable inference algorithms

- **OWL (Web Ontology Language)**
  - An ontology language for the Semantic Web
  - Based on Description Logic
  - RDF/XML syntax

- **OWL 1.1 and 2**
  - Extend OWL
  - Modify syntax

Description Logic

- **Description Logics** allow formal concept definitions that can be reasoned about to be expressed
  - Example Concept definitions:
    - Woman ⊑ Person ∩ Female
    - Man ⊑ Person ∩ ¬Woman
  - Not a single logic, but a family of KR logics originating from KL-One e.g. AL, ALC,…SHIQ,…SHIN(D)
  - Subsets of first-order logic
  - Well-defined model theory
  - Known computational complexity

**FACT inference algorithm**
- Prove subsumption
- Prove disjointness

Further reading (not required reading):
- Horrocks, Ian. (1997) Optimising tableaux decision procedures for Description Logics, and many papers on-line
- Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P. Description Logic Handbook (Chapter 2)

Description Logic Terminology

- **A Box**: Assertions
  - E.g. hasChild(john, mary)
  - This is the knowledge base
  (we will not look at this aspect)

- **T Box**: Terminology
  - The definitions of concepts in the ontology
  - Example axioms for definitions
    - C ⊑ D [C is a subclass of D, D subsumes C]
    - C = D [C is defined by the expression D]
Important terminology:
- Concept: class, category or type (as introduced earlier)
- Role: binary relation
- Attributes are functional roles
- Subsumption:
  - D subsumes C if C is a subclass of D
  - i.e. All Cs are Ds
- Unfoldable terminologies:
  - The defined concept does not occur in the defining expression
  - C \subseteq D where C does not occur in the expression D
- Language families
  - AL: Attributive Language
  - ALC adds full negation to AL

Language elements for concept expressions:
- Bottom: the empty set
- Top: the universal set
- CN: Concept name
- C: Concept expression
- R: Role expressions, limited to RN Role Names
- \neg: 'Not' forms the complement of a concept
- \cup: 'Union' forms the union (OR) of two concepts
- \cap: 'Intersection' forms the intersection (AND) of two concepts
- ∀: 'Value restriction'
- ∃: 'Exists restriction'

Grammar for C:
- \bot | \top | CN | \neg CN | C \cup D | C \cap D | \forall R.C | \exists R.C

Language elements for terminological axioms:
- C \equiv D ‘is defined by’ C is equivalent to D
- C \subseteq D ‘is subsumed by’ C is subsumed by/is a subclass of D

Terminological axioms make assertions about concept expressions.

Grammar for terminological axioms:
- C \equiv D | C \subseteq D

The cases of most interest are where CN is given a
- ‘necessary and sufficient definition’: CN \equiv D
And where CN is given a
- ‘necessary definition’: CN \subseteq D

Terminological axioms: Inclusions and equalities
- Concepts: C \subseteq D and C \equiv D
- Roles: R \subseteq S and R = S
Description Logic ALC

Example concept expressions:

**Parent** = “Persons who have (amongst other things) some children”

\[ \text{Person} \sqcap \exists \text{hasChild}. \top \]

**ParentOfBoys** = “Persons who have some children, and only have children that are male”

\[ \text{Person} \sqcap (\exists \text{hasChild}. \top) \sqcap (\forall \text{hasChild}. \text{Male}) \]

**ScottishParent** = “Persons who only have children that drink (amongst other things) some IrnBru”

\[ \text{Person} \sqcap (\exists \text{hasChild}. (\exists \text{drink.IrnBru})) \]

Each term (atomic or compound) defines a set as given by the right-hand column in the table

– The model theory makes this more formal

### ALC Model Theory: (wff) = {…a set…}; R' is a set = {<d,r>,…}

<table>
<thead>
<tr>
<th>CN', DN'</th>
<th>Atomic concepts</th>
<th>Non-empty sets CN', DN' \subseteq \lambda'</th>
</tr>
</thead>
<tbody>
<tr>
<td>\bot'</td>
<td>Bottom</td>
<td>\emptyset</td>
</tr>
<tr>
<td>\top'</td>
<td>Universal concept, Top</td>
<td>\lambda'</td>
</tr>
<tr>
<td>\neg C'</td>
<td>Full Negation</td>
<td>\lambda' \setminus C'</td>
</tr>
<tr>
<td>(C \cup D)'</td>
<td>Union</td>
<td>C' \cup D'</td>
</tr>
<tr>
<td>(C \cap D)'</td>
<td>Intersection</td>
<td>C' \cap D'</td>
</tr>
<tr>
<td>(\forall R,C)'</td>
<td>Value restriction</td>
<td>{x \in \lambda' \mid \forall y &lt;x,y&gt; \in R' \Rightarrow y \in C}</td>
</tr>
<tr>
<td>(\exists R,C)'</td>
<td>Full existential restriction</td>
<td>{x \in \lambda' \mid \exists y &lt;x,y&gt; \in R' \land y \in C}</td>
</tr>
</tbody>
</table>

Terminological axioms: Inclusions and equalities

Concepts: C \subseteq D iff C' \subseteq D'

C = D iff C' = D'

Value and Exists Restrictions

{a,b,c,d,e,f} are instances; Plant and Animal are classes

\[ \text{Plant} \sqcap \text{Animal} \subseteq \bot' \]

(disjointness)  \[ \top' \subseteq \text{Plant} \sqcup \text{Animal} \]

(partition)

\[ \exists \text{eats}. \text{Animal} = \{c,d,e\} \]

\[ \forall \text{eats}. \text{Animal} = \{a,b,c,e,f\} \]

\[ \exists \text{eats}. \text{Animal} \sqcap \forall \text{eats}. \text{Animal} = \{c,e\} \]
Model Theory

\[ \Delta \text{universal domain of individuals, let } \Delta = \{a, b, c, d, e, f\} \]

- set of pairs for the relation eats, let \( \text{eats} = \{ (d, a), (d, e), (e, d), (e, f), (c, f) \} \)

For all concepts C:

i) \( C \subseteq \Delta \)

ii) \( \Diamond C \wedge \phi \)

Let \( \text{Animal} = \{ d, e, f \} \)

\( \neg \text{Animal} = \{ a, b, c \} \)

\( \text{eats. Animal} = \{ a, b, c, e, f \} \)

\( \text{eats. \neg Animal} = \{ c, d, e \} \)

Value and Exists Restrictions

\( \{a, b, c, d, e, f\} \) are instances; Plant and Animal are classes

Inference:

- MeatEater = \( \text{eats. Animal} = \{a, b, c, e, f\} \)
- Vegetarian = \( \text{eats. \neg Animal} = \{a, b, f\} \)
- Omnivore = \( \text{eats. Animal} = \{c, d, e\} \)

Vegetarian = \( \{a, b, f\} \) partition? MeatEater= \( \{a, b, c, e, f\} \)

ALC: Value Restriction

Value restriction: \( \forall R.C \)

- \( R \) is a binary relation, e.g. \( \text{eats}(x, y) \)
- \( C \) is a concept expression, e.g. Animal

Consider:

\[ \forall y \text{ if } \text{eats}(x, y) \text{ then } y \in \text{Animal} \]

defines the set \( x \): Veats. Animal "things that eat only Animal"

In the formal model theory, where the domain is \( \Delta \), \( \text{eats} \) is represented by a set of tuples, e.g.

\( \text{eats} = \{ (d, a), (d, e), (e, d), (e, f), (c, f) \} \) meaning \( \text{eats}(d, a) \text{ eats}(d, e) \ldots \)

\( \text{Animal} = \{ d, e, f \} \)

The set corresponding to \( \forall y \text{ if } \text{eats}(x, y) \text{ then } y \in \text{Animal} \) is:

\[ \{ x \in \Delta | \forall y \text{ if } \text{eats}(x, y) \text{ then } y \in \text{Animal} \} = \{ a, b, c, e, f \} \]

In general, \( \forall R.C \) is interpreted as:

\[ \{ x \in \Delta | \forall y \text{ if } \text{eats}(x, y) \text{ then } y \in \text{Animal} \} = \{ a, b, c, e, f \} \]

<table>
<thead>
<tr>
<th></th>
<th>\text{allAnimal}</th>
<th>\neg</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{F}</td>
<td>\text{F}</td>
<td>\text{F}</td>
</tr>
<tr>
<td>\text{F}</td>
<td>\text{T}</td>
<td>\text{T}</td>
</tr>
<tr>
<td>\text{T}</td>
<td>\text{F}</td>
<td>\text{F}</td>
</tr>
<tr>
<td>\text{T}</td>
<td>\text{T}</td>
<td>\text{T}</td>
</tr>
</tbody>
</table>

ALC: Existential Restriction

Existential restriction: \( \exists R.C \)

- \( R \) is a binary relation, e.g. \( \text{eats}(x, y) \)
- \( C \) is a concept expression, e.g. Animal

Consider:

\[ \exists y \text{ if } \text{eats}(x, y) \text{ and } y \in \text{Animal} \]

defines the set \( x \): Beats. Animal "things that eat some Animal"

In the formal model theory, where the domain is \( \Delta \), \( \text{eats} \) is represented by a set of tuples, e.g.

\( \text{eats} = \{ (d, a), (d, e), (e, d), (e, f), (c, f) \} \) meaning \( \text{eats}(d, a) \text{ eats}(d, e) \ldots \)

\( \text{Animal} = \{ d, e \} \)

The set corresponding to \( \exists y \text{ if } \text{eats}(x, y) \text{ and } y \in \text{Animal} \) is:

\[ \{ x \in \Delta | \exists y \text{ if } \text{eats}(x, y) \text{ and } y \in \text{Animal} \} = \{ a, b, c, e, f \} \]

In general, \( \exists R.C \) is interpreted as:

\[ \{ x \in \Delta | \exists y \text{ if } \text{eats}(x, y) \text{ and } y \in \text{Animal} \} = \{ a, b, c, e, f \} \]

<table>
<thead>
<tr>
<th></th>
<th>\text{allAnimal}</th>
<th>\neg</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{F}</td>
<td>\text{F}</td>
<td>\text{F}</td>
</tr>
<tr>
<td>\text{F}</td>
<td>\text{T}</td>
<td>\text{T}</td>
</tr>
<tr>
<td>\text{T}</td>
<td>\text{F}</td>
<td>\text{F}</td>
</tr>
<tr>
<td>\text{T}</td>
<td>\text{T}</td>
<td>\text{T}</td>
</tr>
</tbody>
</table>
DL Inference

- Inference can be expressed in terms of the model:
  - Satisfiability of $C$: $C$ is non-empty
  - Subsumption $C \sqsubseteq D$ iff $C \subseteq D$ ("$C$ is subsumed by $D")
  - Equivalence $C \equiv D$ iff $C = D$
  - Disjointness $(C \cap D) \perp \perp$ iff $C \cap D = \phi$
- Tractable/terminating inference algorithms exist

### Example

**MeatEater** $\equiv$ Yeats. Animal
**Vegetarian** $\equiv$ Yeats. ~Animal
**Omnivore** $\equiv$ Yeats. Animal

**Query:**
- a) Vegetarian $\equiv$ MeatEater
  - No
- b) (MeatEater $\cap$ Vegetarian) $\perp \perp$
  - No
- c) (Omnivore $\cap$ Vegetarian) $\perp \perp$
  - Yes

### Answer:

<table>
<thead>
<tr>
<th>MeatEater</th>
<th>Vegetarian</th>
<th>Omnivore</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\perp \perp$</td>
<td>$\perp \perp$</td>
<td>$\perp \perp$</td>
</tr>
</tbody>
</table>

FACT Algorithm

1. **The FACT tableaux method**
   - A tractable, extendable procedure
     - extendable to more expressive DLs than ALC e.g. with cardinality constraints and role expressions
   - Assume an unfoldable terminology
     - exclude: Human $\equiv \exists$hasParent. Human
   - Assume all definitions are necessary and sufficient $\equiv$
   - Proof is by unsatisfiability
     - To show $C$ and $D$ are disjoint or in a subsumption relation, a goal expression $G$ is formed, and
     - the aim is to reject $G$
2. **4 steps:**
   - Steps 1-3 transform the goal into negation normal form
   - Step 4 constructs a tableaux (a labelled tree)

DL Inference

Inference has 2 equivalent notions - so implementing one lets us prove all 4 properties

- **Reduction to subsumption $\sqsubseteq$**:
  - Unsatisfiability of $C$: $C \subseteq \bot$
  - Equivalence $C \equiv D$ iff $C \subseteq D$ and $D \subseteq C$
  - Disjointness $(C \cap D) \perp \perp$

- **Reduction to unsatisfiability $C \cap D = \phi$**:
  - Subsumption $C \subseteq D$ iff $(C \cap \neg D)$ is unsatisfiable
  - Equivalence $C \equiv D$ iff $(C \cap \neg D)$ and $(D \cap \neg C)$ are unsatisfiable
  - Disjointness $(C \cap D)$ is unsatisfiable

FACT Algorithm

1. Given two expressions $C$ and $D$, replace all defined terms by their definition, e.g. if $C \equiv E \perp F$ then replace $C$ by $E \perp F$
   - Continue until all defined terms are replaced ($E$ and $F$ may be defined)
   - Do this for $C$ to get $C'$ and $D$ to get $D'$
2. Construct the goal $G$
   - To show $C$ and $D$ are disjoint, $G$ is $C' \cap D'$
   - To show $C \sqsubseteq D$, $G$ is $C' \cap \neg D'$
3. Convert $G$ to negation normal form using these equivalences:
   - $\neg \exists R. A = \exists R. \neg A$
   - $\neg \forall R. A = \forall R. \neg A$
   - $\neg (A \cap B) = \neg A \cup \neg B$
   - $\neg (A \cup B) = \neg A \cap \neg B$
   - As a result, the ‘not’ operator is pushed to the inner-most term and only atomic concept expressions are negated
FACT Algorithm

4. Tableaux method - FACT algorithm (Ian Horrocks) for ALC
   - The tableaux is represented by a tree
   - The tree is constructed from a root node, \( a_0 \), whose label is the goal \( G \): \( L(a_0) = \{G\} \)
   - Nodes represent individuals (\( a_0 \) and \( a_1 \) in the figure below)
   - Edges represent roles (relationships)
     - Edges are labelled with role names
     - If the edge \( <x,y> \) is labelled \( R \) then “\( y \) is an \( R \) successor of \( x \)”
   - \( L(x) \) is the label of node \( x \)
     - The individual \( x \) must be in the extension of every concept in \( L(x) \)
   - The tree contains a clash if \( \{C, \neg C\} \) is in \( L(x) \)

Tableaux method - rules that construct the tree

1. \( \cap \)-rule: \( (C \cap D) \in L(x) \) then add \( C \) and \( D \) to \( L(x) \)
2. \( \cup \)-rule: \( (C \cup D) \in L(x) \) then add \( C \) or \( D \) to \( L(x) \)
3. \( \exists \)-rule: \( \exists R.C \in L(x) \) then add \( L(<x,y>) = R \) (if it does not yet exist) and \( C \in L(y) \)
4. \( \forall \)-rule: \( \forall R.C \in L(x) \) then if \( C \) does not occur in \( L(y) \) add \( C \) to \( L(y) \)

Are Vegetarian and Omnivore disjoint?
Vegetarian \( \cap \) Omnivore \( \subseteq \bot \)

Replace named classes by their definition:
Vegetarian \( \equiv \) Yeats.\( \neg \)Animal
Omnivore \( \equiv \exists \)eats.Animal

Construct goal: Yeats.\( \neg \)Animal \( \cap \exists \)eats.Animal
{Yeats.\( \neg \)Animal \( \cap \exists \)eats.Animal} \[\text{already in NNF}\]

A) Are Vegetarian and Omnivore disjoint?
\( \{\text{Yeats.} \neg \text{Animal} \cap \exists \text{eats.Animal}\} \)
Apply 1 then 3 then 4.

\textbf{Description Logic}

Construct goal: Yeats.\( \neg \)Animal \( \cap \exists \)eats.Animal
{Yeats.\( \neg \)Animal \( \cap \exists \)eats.Animal} \[\text{already in NNF}\]

Apply 1 then 3 then 4.

A) Are Vegetarian and Omnivore disjoint?
\( \{\text{Yeats.} \neg \text{Animal} \cap \exists \text{eats.Animal}\} \)
Apply 1 then 3 then 4.

\textbf{Description Logic}

Are Vegetarian and Omnivore disjoint?
Vegetarian \( \cap \) Omnivore \( \subseteq \bot \)

Replace named classes by their definition:
Vegetarian \( \equiv \) Yeats.\( \neg \)Animal
Omnivore \( \equiv \exists \)eats.Animal

Construct goal: Yeats.\( \neg \)Animal \( \cap \exists \)eats.Animal
{Yeats.\( \neg \)Animal \( \cap \exists \)eats.Animal} \[\text{already in NNF}\]

A) Are Vegetarian and Omnivore disjoint?
\( \{\text{Yeats.} \neg \text{Animal} \cap \exists \text{eats.Animal}\} \)
Apply 1 then 3 then 4.

\textbf{Description Logic}

Are Vegetarian and Omnivore disjoint?
Vegetarian \( \cap \) Omnivore \( \subseteq \bot \)

Replace named classes by their definition:
Vegetarian \( \equiv \) Yeats.\( \neg \)Animal
Omnivore \( \equiv \exists \)eats.Animal

Construct goal: Yeats.\( \neg \)Animal \( \cap \exists \)eats.Animal
{Yeats.\( \neg \)Animal \( \cap \exists \)eats.Animal} \[\text{already in NNF}\]

A) Are Vegetarian and Omnivore disjoint?
\( \{\text{Yeats.} \neg \text{Animal} \cap \exists \text{eats.Animal}\} \)
Apply 1 then 3 then 4.
Description Logic

Are Vegetarian and MeatEater disjoint?
Vegetarian \( \cap \) MeatEater \( \perp \)
Replace named classes by their definition:
Vegetarian \( \equiv \) Yeats.\~Animal
MeatEater \( \equiv \) Yeats.Animal
Construct goal: Yeats.\~Animal \( \cap \) Yeats.Animal
\( \{\)Yeats.\~Animal \( \cap \) Yeats.Animal\( \} \) [already in NNF] 
\( \{\)Yeats.\~Animal, Yeats.Animal\( \} \) [a0: by \( \exists \) elimination split term in \( L(a0) \)]
No more rules apply, therefore disjointness cannot be proven.
Note, \( \forall \) elimination cannot be applied unless an edge labelled eats already exists.

KMM ontology Lecture 3 / 4

Description Logic

Does Vegetarian subsume Omnivore?
Omnivore \( \equiv \) Vegetarian
Omnivore \( \cap \) \~Vegetarian \( \perp \)
Replace named classes by their definition:
Omnivore \( \equiv \) Yeats.Animal
Vegetarian \( \equiv \) Yeats.\~Animal
Construct goal: Yeats.Animal \( \cap \) \~Yeats.\~Animal
\( \{\)Yeats.Animal \( \cap \) \~Yeats.Animal\( \} \) [after conversion to NNF] 
\( \{\)Yeats.Animal, \~Yeats.Animal\( \} \) [a0: by \( \exists \) elimination split term in \( L(a0) \), by \( \forall \) elimination add edge, and add \~Animal to \( L(a1) \), \( L(a1) = \{\)\~Animal\( \} \) ]
[a0: \( \forall \) elimination would add \~Animal to \( L(a1) \) ]
No more rules apply, subsumption is not proven

KMM ontology Lecture 3 / 4

Description Logic

Does MeatEater subsume Vegetarian?
Vegetarian \( \equiv \) Yeats.\~Animal
MeatEater \( \equiv \) Yeats.Animal
Replace named classes by their definition:
Vegetarian \( \equiv \) Yeats.\~Animal
MeatEater \( \equiv \) Yeats.Animal
Construct goal: Yeats.\~Animal \( \cap \) Yeats.Animal
\( \{\)Yeats.\~Animal \( \cap \) Yeats.Animal\( \} \) [after conversion to NNF] 
\( \{\)Yeats.\~Animal, Yeats.\~Animal\( \} \) [a0: by \( \exists \) elimination split term in \( L(a0) \), by \( \forall \) elimination add edge, and add \~Animal to \( L(a1) \), \( L(a1) = \{\)\~Animal\( \} \) ]
[a0: \( \forall \) elimination would add \~Animal to \( L(a1) \) ]
No more rules apply, subsumption is not proven

KMM ontology Lecture 3 / 4

Description Logic

Show C and D are disjoint:
\[ C \equiv \forall r.\~A \cap \forall r.\exists s.\~B \]
\[ D \equiv \exists r.((\forall s.\exists B) \cup A) \]
[\( a0: \) Apply \( \exists \) elimination, then \( \forall \) elimination to create edge to \( a1 \).
Add \( (\forall s.\exists B) \cup A \) to \( L(a1) \).
Apply \( \forall \) elim. to remaining \( a0 \) terms]

KMM ontology Lecture 3 / 4

Description Logic

Show C and D are disjoint:
\[ C \equiv \forall r.\~A \cap \forall r.\exists s.\~B \]
\[ D \equiv \exists r.((\forall s.\exists B) \cup A) \]
[\( a0: \) Apply \( \exists \) elimination, then \( \forall \) elimination to create edge to \( a1 \).
Add \( (\forall s.\exists B) \cup A \) to \( L(a1) \).
Apply \( \forall \) elim. to remaining \( a0 \) terms]

KMM ontology Lecture 3 / 4
**Description Logic**

- Defining concepts:
  - Value restrictions are often combined with appropriate classes using intersection:
    - Vegan ≡ Person ⊓ Veats.Plant
    - Vegetarian ≡ Person ⊓ Veats.(Plant ⊓ Dairy)
  - Value restrictions may need an existential expression
    - If we want to prevent people who don’t eat at all being classified as Vegan:
      - Vegan ≡ Person ⊓ Veats.Plant ⊓ Veats.Fish
  - Classes are not disjoint by default
    - Explicit disjointness assertions are needed
  - For all does not imply some
    - Veats.Fish and Veats.¬Fish are not necessarily contradictory unless Veats. T

---

**Description Logic**

Tableaux method can be extended:
- Transitive roles, e.g. part-of is a transitive relation
- Number restrictions, e.g. ParentsWithThreeOrMoreChildren

General Terminologies
- C ⊑ D iff (C ∧ ¬D) = ∅ for all models I of T
- Add ¬C ⊑ D to all I(x) as a meta-constraint M
- Cope with non-terminating terminologies by a blocking rule
  - If the label occurs earlier in the tree then stop
    - Human ⊧ hasParent.Human
  - node (a1) is blocked showing satisfiability

More ‘Syntactic’ Proofs
- Is there a model for: Veats.¬Animal ⊓ Veats.Animal ?
  - [Previously, the tableaux was shown to have a clash]
- Apply the ¬∀ equivalence rule:
- There is no intersection between ¬P and P for any concept expression P, and so the answer is no
- The tableaux construction rules can be modified to detect such contradictions

---

**Description Logic**

Relationship to first-order logic (advanced topic)
- Necessity/All time/Knows
  - ∀R.C (x) = ∀y R(x, y) ⇒ φ(y) [for CN: ∀y R(x, y) ⇒ CN(y)]
  - ¬R.C (x) = ∃y R(x, y) ∧ φ(y) [for CN: ∃y R(x, y) ∧ CN(y)]

Modal Logics
- □ P iff ∀w r(v,w) ⇒ [P]w
- ◇ P iff ∃w r(v,w) ∧ [P]w

DL and (multi) modal K have the same duality between operators
- ◇R.C = ∃R.¬C
- □R.¬C = ¬R. ◇C
- □R.P = ◇R.¬P
- ◇R.¬P = □R.¬P

KMM ontology Lecture 3 / 4
Description Logics and their properties

- ALC
  - Sound and complete subsumption testing
- ALCN
  - ALC + number restriction \( \preceq R \)
- ALC\( \rightarrow \)
  - ALC + transitivity closed roles
- SHIQ
  - SH family: ALC + transitive roles and role hierarchy
- SHOQ(D)
  - Adds datatypes (D) and enumerated types to SHIQ
- SHIF(D)
  - Adds datatypes transitive roles and role hierarchy, plus functional attributes to SHIQ (OWL-Lite)
- SHOIN(D)
  - Adds nominals to class descriptions (oneOf \{a,b,c\}) and arbitrary cardinality constraints (OWL-DL)

Web Ontology Language: OWL

- Web Ontology Language (OWL) is W3C Recommendation for an ontology language for the web
  - Has an XML syntax
- OWL is layered on RDF and RDFS (other W3C standards)
  - Conforms to the RDF/RDFS semantics
- OWL has 3 versions:
  - OWL-Lite - the simpler OWL DL
  - OWL-DL - more expressive DL
  - OWL-Full - not confined to DL, closer to FOL
- OWL DLs extend ALC
  - Allow instances to be represented (A Box)
  - Provides datatypes
  - Provides number restrictions
- OWL 1.1 and 2 extend OWL DL

OWL Object Properties

OWL makes a distinction between Object types and Datatypes

Object types and Object properties are the same as in ALC

<table>
<thead>
<tr>
<th>CN, DN</th>
<th>Atomic concepts</th>
<th>Non-empty sets CN', DN' ( C, D' )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C' )</td>
<td>owl:Nothing</td>
<td>( \not! )</td>
</tr>
<tr>
<td>( T' )</td>
<td>owl:Thing</td>
<td>( \not! )</td>
</tr>
<tr>
<td>( \neg C' )</td>
<td>Full Negation</td>
<td>( \not! { C } )</td>
</tr>
<tr>
<td>( C \cup D' )</td>
<td>Union</td>
<td>( C' \cup D' )</td>
</tr>
<tr>
<td>( C \cap D' )</td>
<td>Intersection</td>
<td>( C' \cap D' )</td>
</tr>
<tr>
<td>( (\exists R.C) )</td>
<td>Value restriction</td>
<td>( { x \in \not!</td>
</tr>
<tr>
<td>( (\exists R.C) )</td>
<td>Full existential quantification</td>
<td>( { x \in \not!</td>
</tr>
</tbody>
</table>

Terminological axioms: Inclusions and equalities

Concepts: \( C \sqsubseteq D \) iff \( C' \sqsubseteq D' \)
\[ C \equiv D \iff C' = D' \]

OWL Datatypes

- Datatypes \( D' \) are distinct from Object types \( D' \)
  - A datatype relation \( U \), e.g. age, relates an object type, e.g. Person to an integer
    - \( \# \text{Age.Integer} \) [the set of things that have some Integer as age]
  - Data types correspond to XML Schema types
  - OWL also provides hasValue: \( U:v \) to represent specific datatype values
    - \( \text{age:29} \) [the set of things age 29]

<table>
<thead>
<tr>
<th>D</th>
<th>Data Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>D'</td>
<td>( D' \sqsubseteq \Delta D )</td>
</tr>
<tr>
<td>( \forall U.D )</td>
<td>Value restriction</td>
</tr>
<tr>
<td>( \exists U.D )</td>
<td>Full existential quantification</td>
</tr>
</tbody>
</table>

OWL Number Restrictions

- OWL adds (unqualifying) number restrictions to ALC
  - Defines the set of instances, $x$, for which there $n$ or more instances, $y$, such that $R(x, y)$
  - BusyParent $\geq 3$ hasChild
  - Defines the set of instances, $x$, for which there $n$ or less instances, $y$, such that $R(x, y)$

| $\geq n R$ | Minimum cardinality $\{x \in \Delta^I \mid \#(<x, y> \in R) \geq n\}$ |
| $\leq n R$ | Maximum cardinality $\{x \in \Delta^I \mid \#(<x, y> \in R) \leq n\}$ |

Disjointness axioms

Assume C and D are asserted to be disjoint in Protégé - example of an axiom.
Q. Can anything be a subset of C and D?
Define a new class: TestClass $\equiv C \cap D$
Goal: $C \cap D$
$L(a_0) = \{C \cap D\}$
$L(a_0) = \{C, D\} \text{ no clash}$

Disjointness means: $\top \not\subseteq \neg C \cup \neg D$ [equivalent to $C \cap D \models \bot$]
$L(a_0) = \{C, D, \neg C \cup \neg D\}$

Using OWL

- Datatypes $\Delta_D^I$ and Object types $\Delta^I$

<table>
<thead>
<tr>
<th>$\Delta_D^I$, $\Delta^I$</th>
<th>Non-empty sets $\Delta_D^I$, $\Delta^I$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BN, CN $\subseteq \Delta_D^I$</td>
<td>Non-empty sets $\Delta_D^I$, $\Delta^I$</td>
</tr>
<tr>
<td>D $\subseteq \Delta_D^I$</td>
<td>${x \in \Delta^I \mid x \in \text{BN} \lor x \in \text{CN}}$</td>
</tr>
<tr>
<td>$(\text{B} \cup \text{C})^I$</td>
<td>${x \in \Delta^I \mid x \in \text{B} \lor x \in \text{C}}$</td>
</tr>
<tr>
<td>$(\text{B} \cap \text{C})^I$</td>
<td>${x \in \Delta^I \mid x \in \text{B} \land x \in \text{C}}$</td>
</tr>
<tr>
<td>$(\forall \text{R.C})^I$</td>
<td>${x \in \Delta^I \mid \forall y (&lt;x, y&gt; \in \text{R} \Rightarrow y \in \text{C})}$</td>
</tr>
<tr>
<td>$(\exists \text{R.C})^I$</td>
<td>${x \in \Delta^I \mid \exists y &lt;x, y&gt; \in \text{R} \land y \in \text{C}}$</td>
</tr>
<tr>
<td>$(\forall \text{U.D})^I$</td>
<td>${x \in \Delta^I \mid \forall y (&lt;x, y&gt; \in \text{U} \Rightarrow y \in \text{D})}$</td>
</tr>
<tr>
<td>$(\exists \text{U.D})^I$</td>
<td>${x \in \Delta^I \mid \exists y &lt;x, y&gt; \in \text{U} \land y \in \text{D}}$</td>
</tr>
</tbody>
</table>

OWL-DL Cardinality

- Cardinality

<table>
<thead>
<tr>
<th>$\Delta_D^I$, $\Delta^I$</th>
<th>Non-empty sets $\Delta_D^I$, $\Delta^I$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BN, CN $\subseteq \Delta_D^I$</td>
<td>Non-empty sets $\Delta_D^I$, $\Delta^I$</td>
</tr>
<tr>
<td>$(\forall \text{R.C})^I$</td>
<td>${x \in \Delta^I \mid \forall y (&lt;x, y&gt; \in \text{R} \Rightarrow y \in \text{C})}$</td>
</tr>
<tr>
<td>$(\exists \text{R.C})^I$</td>
<td>${x \in \Delta^I \mid \exists y &lt;x, y&gt; \in \text{R} \land y \in \text{C}}$</td>
</tr>
<tr>
<td>$(\exists \text{R.C})^I$</td>
<td>${x \in \Delta^I \mid \exists y &lt;x, y&gt; \in \text{R} \land y \in \text{C}}$</td>
</tr>
<tr>
<td>$(\forall \text{U.D})^I$</td>
<td>${x \in \Delta^I \mid \forall y (&lt;x, y&gt; \in \text{U} \Rightarrow y \in \text{D})}$</td>
</tr>
<tr>
<td>$(\exists \text{U.D})^I$</td>
<td>${x \in \Delta^I \mid \exists y &lt;x, y&gt; \in \text{U} \land y \in \text{D}}$</td>
</tr>
</tbody>
</table>

hasWheel$^I = \{<a_0, a_1>, <a_0, a_2>\}$ therefore:
$\geq 0 \text{ hasWheel}; \geq 1 \text{ hasWheel}; \geq 2 \text{ hasWheel};$ and
$\leq 2 \text{ hasWheel}; \leq 2 \text{ hasWheel}; \leq 3 \text{ hasWheel} \ldots$
OWL-DL Cardinality

Bicycle ≡ \( \exists 2 \text{hasWheel} \land \forall \text{hasPart.} \sim \text{Engine} \)

- Unicycles would have 1 wheel, tricycles 3 wheels, motorcycles would have 2 wheels and an Engine……
- hasWheel is needed, rather than hasPart, as OWL-DL cannot specify the type of the range to be Wheel
  - Define hasWheel a subProperty of hasPart
  - Range of hasWheel: Wheel
- An example of ‘bias’ being introduced because of the expressivity of the representation

Domain and range specifications

\[
domain(R, C) :: \exists 1 R \subseteq C
\]

Consider:

1) \( \exists \text{hasChild.Male} : \) anything with a male child
2) \( \exists \text{Person} \land \exists \text{hasChild.Male} : \) person with a male child:

The Person intersection in 2) is implicit in 1) if the domain of hasChild is defined as Person

\[
range(R, C) :: \top \subseteq \forall R.C
\]

Resource Description Framework (RDF)

- RDF is a W3C standard, pre-dating OWL, for web semantics
- Identifies ‘things’ through URIs, and describes them in terms of simple properties and property values
- The triple is the basic unit: <subject predicate object>
  - Example: <http://www.example.org/index.html dc:creator http://www.example.org/staffid/85740>
- Subjects and objects are viewed as nodes in a graph, where predicates label the edges
  - Example: dc:creator ns2:85740
  - dc:date "03/03/2004"
- In RDF, predicates represent relationships between resources
  - But RDF provides no way to define these predicates, or state other ontological properties
  - RDF Schema addresses some of these problems

RDF and RDF Schema (RDFS)

- RDFS allows subclasses and the domain and range of properties to be defined (http://www.w3.org/TR/rdf-schema/)
  - e.g. to state that creator has domain Document and range Person, two triples are needed:
    - `<dc:creator rdfs:domain ns:Document>`
    - `<dc:creator rdfs:range ns:Person>`
- RDF Schema
  - rdf:Property the class of properties, an instance of rdfs:Class
  - rdfs:Resource the class of everything
  - rdf:Literal the class of literal values e.g. string, integer
  - rdf:Literal the class of RDF classes
- There is no effective reasoning algorithm for RDFS
  - hence, OWL
### OWL Abstract Syntax

- The ALC-style syntax is not suitable for the WWW
- OWL needs to conform to the RDF/XML syntax

<table>
<thead>
<tr>
<th>OWL/ALC DL Syntax</th>
<th>OWL Abstract Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>(¬C)</td>
<td>&lt;complementOf C&gt;</td>
</tr>
<tr>
<td>(C U D)</td>
<td>&lt;unionOf C D&gt;</td>
</tr>
<tr>
<td>(C I D)</td>
<td>&lt;intersectionOf C D&gt;</td>
</tr>
<tr>
<td>(V R C)</td>
<td>&lt;Restriction&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;onProperty R&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;allValuesFrom C&gt;</td>
</tr>
<tr>
<td>(∃R C)</td>
<td>&lt;Restriction&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;onProperty R&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;someValuesFrom C&gt;</td>
</tr>
<tr>
<td>(C ⊥ D)</td>
<td>&lt;disjoint C D&gt;</td>
</tr>
<tr>
<td>C ⊆ D</td>
<td>&lt;C &lt;subClassof D&gt;&gt;</td>
</tr>
<tr>
<td>C = D</td>
<td>&lt;C &lt;equivalentClass D&gt;&gt;</td>
</tr>
</tbody>
</table>

### OWL in RDF/XML Syntax

- Protégé reads and writes this syntax!
- Use HP’s Jena toolkit in Java applications that need to read/write/manipulate RDF/S or OWL.

**Class definitions C ⊆ D and Property restrictions V R.C in RDF/XML syntax:**

**DieselEngine is a subclass of Engine:**

```
<owl:Class rdf:ID="DieselEngine"
    rdf:about="#Engine">
    <rdfs:subClassOf rdf:resource="#Engine"/>
</owl:Class>
```

**CarPart is a subclass of the parts of the Car:**

```
<owl:Class rdf:ID="CarPart">
    <rdfs:subClassOf rdf:resource="#partOf.Car"/>
</owl:Class>
```

**CarEngine is equivalent to the intersection of Engine and ∃partOf.Car:**

```
<owl:Class rdf:ID="CarEngine">
    <owl:equivalentClass>
        <owl:Class rdf:about="#Engine"/>
        <owl:intersectionOf rdf:parseType="Collection">
            <owl:Class rdf:about="#Engine"/>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#partOf.Car"/>
                <owl:allValuesFrom rdf:resource="#Car"/>
            </owl:Restriction>
        </owl:intersectionOf>
    </owl:equivalentClass>
</owl:Class>
```

**Protégé reads and writes this syntax!**

**Use HP’s Jena toolkit in Java applications that need to read/write/manipulate RDF/S or OWL.**

### OWL

- Is a web-compatible ontology language
- Syntax based on RDF/XML
- Semantics compatible with RDF and RDFS
- OWL-Lite and OWL-DL have a formal interpretation based on DLs
- Extensive documentation at http://www.w3c.org

**Editing Tools**
- Protégé 4