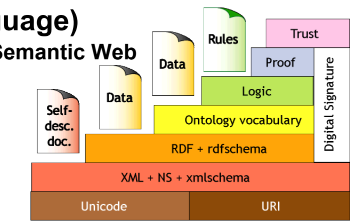
Description Logic and OWL

- **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 W3C standard
  - 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):

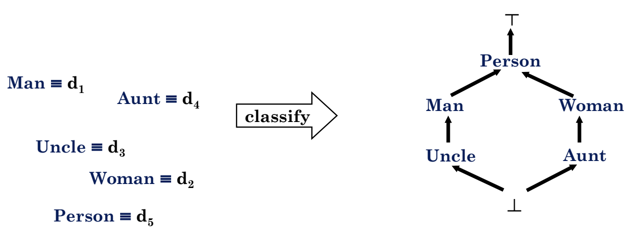
Ch1-3

Horrocks, Ian. (1997) Optimising tableaux decision procedures for Description Logics, and many papers on-line

Baader, F., Calvanese, D., McGuiness, D., Nardi, D., and Patel-Schneider, P. *Description Logic Handbook* (Chapter 2)

Description Logic

- A Classifier (a reasoning engine) can be used to construct the class hierarchy from the definitions of individual concepts in the ontology
- Concept definitions are composed from primitive elements and so the ontology is more maintainable



Description Logic Terminology

Description Logics separate assertions and concept definitions

- **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]

Description Logic Terminology

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 ≡ D where C does not occur in the expression D
- Language families
  - AL: Attributive Language
  - ALC adds full negation to AL

Description Logic

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
¬	'Not' forms the complement of a concept
⊔	'Union' forms the union (OR) of two concepts
⊓	'Intersection' forms the intersection (AND) of two concepts
∀	'Value restriction'
∃	'Exists restriction'

Grammar for C: ⊥ | ⊤ | CN | ¬C | C ⊔ D | C ⊓ D | ∀R.C | ∃R.C

Description Logic

Language elements for terminological axioms:

C ≡ D 'is defined by' C is equivalent to D

C ⊑ 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 ≡ D | C ⊑ D

The cases of most interest are where CN is given a

'necessary and sufficient definition': CN ≡ D

And where CN is given a

'necessary definition': CN ⊑ D

Description Logic ALC

CN, DN	Atomic concept	Sets CN, DN
⊥	Bottom	Empty set
⊤	Universal concept, Top	Universal set
¬C	Full Negation	Complement of C
C ⊔ D	Union	Union of C and D
C ⊓ D	Intersection	Intersection of C and D
∀R.C	Value restriction	The set {x| ∀y R(x, y) ⇒ y∈C}
∃R.C	Full existential restriction	The set {x| ∃y R(x, y)∧ y∈C}

Terminological axioms: Inclusions and equalities

Concepts: C ⊑ D and C ≡ D

Roles: R ⊑ S and R ≡ S

Description Logic ALC

Example concept expressions:

Parent ≡ ***"Persons who have (amongst other things) <u>some</u> children"***

Person $\sqcap$ $\exists$hasChild.$\top$

ParentOfBoys ≡ ***"Persons who have <u>some</u> children, and <u>only</u> have children that are male"***

Person $\sqcap$ ($\exists$hasChild.$\top$) $\sqcap$ ($\forall$hasChild.Male)

ScottishParent ≡ ***"Persons who <u>only</u> have children that drink (amongst other things) some IrnBru"***

Person $\sqcap$ ($\forall$hasChild. ($\exists$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**

Description Logic ALC

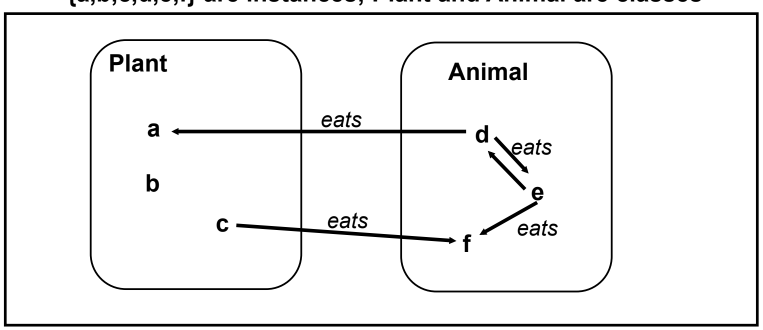
ALC Model Theory: $(wff)^I$ = {…a set…}; R^I is a set = {<d,r>,…}

CN^I, DN^I	Atomic concepts	Non-empty sets CN^I, $DN^I \subseteq \Delta^I$
$\bot^I$	Bottom	ϕ
$\top^I$	Universal concept, Top	Δ^I
$(\neg C)^I$	Full Negation	$\Delta^I \setminus C^I$
$(C \sqcup D)^I$	Union	$C^I \cup D^I$
$(C \sqcap D)^I$	Intersection	$C^I \cap D^I$
$(\forall R.C)^I$	Value restriction	$\{x \in \Delta^I \mid \forall y\ \langle x,y\rangle \in R^I \Rightarrow y \in C^I\}$
$(\exists R.C)^I$	Full existential restriction	$\{x \in \Delta^I \mid \exists y\ \langle x,y\rangle \in R^I \wedge y \in C^I\}$

Terminological axioms: Inclusions and equalities
Concepts: $C \sqsubseteq D$ iff $C^I \subseteq D^I$
$C \equiv D$ iff $C^I = D^I$

Value and Exists Restrictions

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

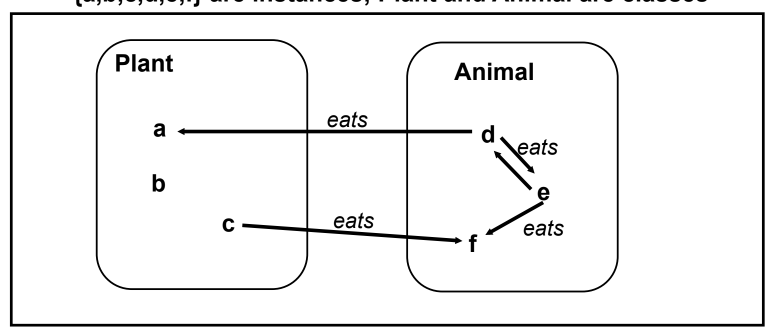


Plant $\sqcap$ Animal $\sqsubseteq$ $\bot$ (disjointness)

$\top$ $\sqsubseteq$ Plant $\sqcup$ Animal (partition)

Value and Exists Restrictions

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



$\exists$eats.Animal = {c,d,e}

$\forall$eats.Animal = {a,b,c,e,f}

$\exists$eats.Animal $\sqcap$ $\forall$eats.Animal = {c,e}

Description Logic ALC

Model Theory

Δ^I universal domain of individuals, let

Δ^I ={a,b,c,d,e,f}

eats^I set of pairs for the relation eats, let

eats^I = {<d,a>,<d,e>,<e,d>,<e,f>,<c,f>}

For all concepts C:

i) $C^I \subseteq \Delta^I$

ii) $C^I \neq \phi$

Let Animal^I = {d,e,f}

∴ (¬Animal)^I = {a,b,c}

∴ (∀eats. Animal)^I = {a,b,c,e,f}

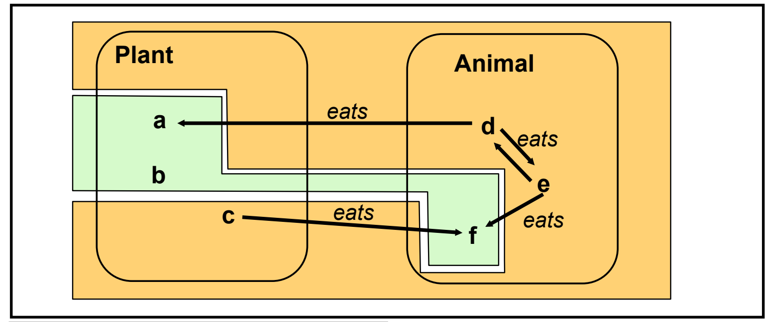
∴ (∃eats. Animal)^I = {c,d,e}

MeatEater ≡ ∀eats. Animal = {a,b,c,e,f}
Vegetarian≡ ∀eats. ¬Animal = {a,b,f}
Omnivore ≡ ∃eats. Animal = {c,d,e}

Inference:
So MeatEater subsumes Vegetarian
and Vegetarian is disjoint from Omnivore
in this model, by these definitions
- BUT the problem is to prove properties
for ALL models

Value and Exists Restrictions

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



Vegetarian = {a,b,f} partition?
Omnivore= {c,d,e}

MeatEater= {a,b,c,e,f}

ALC: Value Restriction

Value restriction: ∀R.C
R is a binary relation, e.g. eats(x, y)
C is a concept expression, e.g. Animal
Consider: ∀eats. Animal "things that eat only Animal"

defines the set x: ∀y if eats(x, y) then y ∈ Animal

In the formal model theory, where the domain is Δ^I, eats is represented by a set of tuples, e.g.
eats^I = {<d,a>,<d,e>,<e,d>,<e,f>,<c,f>} meaning eats(d,a) eats(d,e)…
Animal^I = {d,e,f}
The set corresponding to ∀eats. Animal is:
$\{x \in \Delta^I \mid \forall y <x,y> \in eats^I \Rightarrow y \in Animal^I\}$ = {a,b,c,e,f}
In general, ∀R.C is interpreted as:
$\{x \in \Delta^I \mid \forall y <x,y> \in R^I \Rightarrow y \in C^I\}$

eats(b,a)	a∈Animal^I	⇒
F	F	T
F	T	T
T	F	F
T	T	T

ALC: Existential Restriction

Existential restriction: ∃R.C
R is a binary relation, e.g. eats(x, y)
C is a concept expression, e.g. Animal
Consider: ∃eats. Animal "things that eat some Animal"

defines the set x: ∃y eats(x, y) and y ∈ Animal

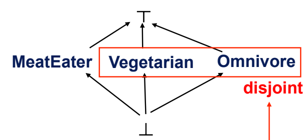
In the formal model theory, where the domain is Δ^I, eats is represented by a set of tuples, e.g.
eats^I = {<d,a>,<d,e>,<e,d>,<e,f>,<c,f>} meaning eats(d, a) eats(d, e)…
Animal^I = {b,e}
The set corresponding to ∃eats. Animal is:
$\{x \in \Delta^I \mid \exists y <x,y> \in eats^I \wedge y \in Animal^I\}$ = {c,d,e}
In general, ∃R.C is interpreted as:
$\{x \in \Delta^I \mid \exists y <x,y> \in R^I \wedge y \in C^I\}$

DL Inference

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

MeatEater $\equiv \forall$eats. Animal
Vegetarian $\equiv \forall$eats. $\neg$Animal
Omnivore $\equiv \exists$eats. Animal

Query:	Answer:
a) Vegetarian $\sqsubseteq$ MeatEater	No
b) (MeatEater $\sqcap$ Vegetarian) $\sqsubseteq \bot$	No
c) (Omnivore $\sqcap$ Vegetarian) $\sqsubseteq \bot$	Yes



DL Inference

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

- **Reduction to subsumption $\sqsubseteq$:**
  - Unsatisfiability of C: $C \sqsubseteq \bot$
  - Equivalence $C \equiv D$ iff $C \sqsubseteq D$ and $D \sqsubseteq C$
  - Disjointness $(C \sqcap D) \sqsubseteq \bot$
- **Reduction to unsatisfiability $C^I = \phi$:**
  - Subsumption $C \sqsubseteq D$ iff $(C \sqcap \neg D)$ is unsatisfiable
  - Equivalence $C \equiv D$ iff $(C \sqcap \neg D)$ and $(D \sqcap \neg C)$ are unsatisfiable
  - Disjointness $(C \sqcap D)$ is unsatisfiable

FACT Algorithm

- **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
- **4 steps:**
  - Steps 1-3 transform the goal into negation normal form
  - Step 4 constructs a tableaux (a labelled tree)

FACT Algorithm

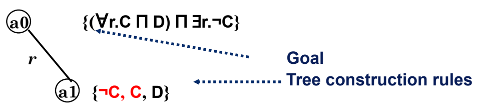
1. Given two expressions C and D, replace all defined terms by their definition, e.g. if $C \equiv E \sqcap F$ then replace C by $E \sqcap F$
   - Continue until all defined terms are replaced (E and F may be defined)
   - Do this for C to get C' and D to get D'
2. Construct the goal G
   - To show C and D are disjoint, G is $C' \sqcap D'$
   - To show $C \sqsubseteq D$, G is $C' \sqcap \neg D'$
3. Convert G to negation normal form using these equivalences:
   $\neg\forall R.A = \exists R.\neg A$
   $\neg\exists R.A = \forall R.\neg A$
   $\neg(A \sqcap B) = \neg A \sqcup \neg B$
   $\neg(A \sqcup B) = \neg A \sqcap \neg B$

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, a0, whose label is the goal G: L(a0) = {G}
- Nodes represent individuals (a0 and a1 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, ¬C} ⊆ L(x)



FACT Algorithm

Tableaux method - rules that construct the tree

1. ⊓ -rule: (C ⊓ D) ∈ L(x) then add C and D to L(x)

   a0 {C ⊓ D} => a0 {C,D}

2. ⊔ -rule: (C ⊔ D) ∈ L(x) then add C or D to L(x)

   a0 {C ⊔ D} => a0 {C} OR {D}

3. ∃-rule: ∃R.C ∈ L(x) then add L(<x,y>)=R (if it does not yet exist) and C ∈ L(y)

   a0 {∃R.C} add an edge R to a new node {C} (unless both exist already)

   R

   a1 {C}

4. ∀-rule:∀R.C ∈ L(x) then IF there is some y s.t. L(<x,y>)=R and L(y) does not contain C, add C to L(y)

   a0 {∀R.C} and if there is an edge labelled R

   R

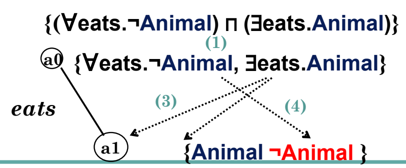
   a1 {C}

FACT Algorithm

Tableaux method - summary of rules

1. ⊓ -rule: (C ⊓ D) ∈ L(x) then add C and D to L(x)
2. ⊔ -rule: (C ⊔ D) ∈ L(x) then add C or D to L(x)
3. ∃-rule: ∃R.C ∈ L(x) then add L(<x,y>)=R (if it does not yet exist) and C ∈ L(y)
4. ∀-rule:∀R.C ∈ L(x) then IF there is some y s.t. L(<x,y>)=R and L(y) does not contain C, add C to L(y)

A)Are Vegetarian and Omnivore disjoint ?



Apply 1 then 3 then 4.

Description Logic

Are Vegetarian and Omnivore disjoint?

Vegetarian ⊓ Omnivore ⊑ ⊥

Replace named classes by their definition:

Vegetarian ≡ ∀eats.¬Animal

Omnivore ≡ ∃eats.Animal

Construct goal: ∀eats.¬Animal ⊓ ∃eats.Animal

{∀eats.¬Animal ⊓ ∃eats.Animal} [already in NNF]

a0 {∀eats.¬Animal, ∃eats.Animal} [a0:by ⊓ elimination split term in L(a0), by ∃ elimination add edge, and add Animal to L(a1), L(a1)={Animal}]

eats

[a0:by ∀ elimination add ¬Animal to L(a1)]

a1 {Animal, ¬Animal} Proven: tableaux shows a clash in L(a1)

Description Logic

Are Vegetarian and MeatEater disjoint?

Vegetarian ⊓ MeatEater ⊑ ⊥

Replace named classes by their definition:

Vegetarian ≡ ∀eats.¬Animal

MeatEater ≡ ∀eats.Animal

Construct goal: ∀eats.¬Animal ⊓ ∀eats.Animal

{∀eats.¬Animal ⊓ ∀eats.Animal} [already in NNF]

a0 {∀eats.¬Animal, ∀eats.Animal} [a0:by ⊓ elimination split term in L(a0)]

No more rules apply, therefore disjointness cannot be proven.

Note, ∀ elimination cannot be applied unless an edge labelled eats already exists.

Description Logic

Does MeatEater subsume Vegetarian?

Vegetarian ⊑ MeatEater

Vegetarian ⊓ ¬MeatEater ⊑ ⊥

Replace named classes by their definition:

Vegetarian ≡ ∀eats.¬Animal

MeatEater ≡ ∀eats.Animal

Construct goal: ∀eats.¬Animal ⊓ ¬∀eats.Animal

{∀eats.¬Animal ⊓ ∃eats. ¬Animal} [after conversion to NNF]

a0 {∀eats.¬Animal, ∃eats. ¬Animal} [a0: by ⊓ elimination split term in L(a0), by ∃ elimination add edge, and add ¬Animal to L(a1), L(a1)={¬Animal}]

eats

[a0: ∀ elimination would add ¬Animal to L(a1)]

a1 {¬Animal} no more rules apply, subsumption is not proven

Description Logic

Does Vegetarian subsume Omnivore?

Omnivore ⊑ Vegetarian

Omnivore ⊓ ¬Vegetarian ⊑ ⊥

Replace named classes by their definition:

Omnivore ≡ ∃eats.Animal

Vegetarian ≡ ∀eats.¬Animal

Construct goal: ∃eats.Animal ⊓ ¬∀eats.¬Animal

{∃eats.Animal ⊓ ∃eats.Animal} [after conversion to NNF]

a0 {∃eats.Animal, ∃eats.Animal} [a0: by ⊓ elimination split term in L(a0), by ∃ elimination add edge, and add Animal to L(a1), L(a1)={Animal}]

eats

a1 {Animal} no more rules apply, subsumption is not proven

Model:
Δ^I = {a0, a1}
$eats^I$ ={<a0,a1>}
$Animal^I$ = {a1}

Description Logic

Show C and D are disjoint:

C ≡ ∀r.¬A ⊓ ∀r.∃s.¬B

D ≡ ∃r.((∀s.B) ⊔ A)

[a0: Apply ⊓ elimination, then ∃ elimination to create edge to a1. Add ((∀s.B) ⊔ A) to L(a1). Apply ∀ elim. to remaining a0 terms]

a0 { (∀r.¬A ⊓ ∀r.∃s.¬B) $\boxed{⊓}$ (∃ r.((∀s.B) ⊔ A))}

{ ∀r.¬A $\boxed{⊓}$ ∀r.∃s.¬B, ∃ r.((∀s.B) ⊔ A)}

{ ∀r.¬A, ∀r.∃s.¬B, $\boxed{∃}$ r.((∀s.B) ⊔ A)}

r

a1 {¬A, ∃s.¬B, ((∀s.B) $\boxed{⊔}$ A)}

{¬A, $\boxed{∃}$s.¬B, ∀s.B} OR { ¬A, ∃s.¬B, A}

s

[a1: apply ⊔ elim., then by ∃ elim. add an edge labelled s to a2, add B to L(a2). Clash immediately closes tree for 2nd disjunct.]

a2 {¬B, B }

clash closes tree for 1st disjunct

Description Logic

- **Defining concepts:**
  - Value restrictions are often combined with appropriate classes using intersection:
    - Vegan ≡ Person ⊓ ∀eats.Plant
    - Vegetarian ≡ Person ⊓ ∀eats.(Plant ⊔ Dairy)
    - Omnivore ≡ Person ⊓ ∃eats.Animal ⊓ ∃eats.(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 ⊓ ∀eats.Plant ⊓ ∃eats.Plant
  - Classes are not disjoint by default
    - » Explicit disjointness assertions are needed
  - *Forall* does not imply *some*
    ∀eats.Fish and ∀eats.¬Fish are not necessarily contradictory unless ∃eats. ⊤

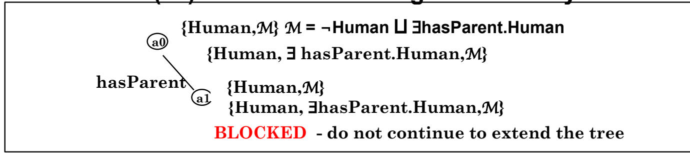
Description Logic

Tableaux method can be extended:

- Transitive roles, e.g. *part-of* is a transitive relation
- Number restrictions, e.g. *ParentsWithThreeOrMoreChilden*

General Terminologies

- C ⊑$_T$D iff (C ⊓ ¬D)^I = ϕ for all models $\mathcal{I}$ of T
- Add ¬C ⊔ D to all L(x) as a meta-constraint $\mathcal{M}$
- 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



Description Logic

More 'Syntactic' Proofs

- Is there a model for: ∀eats.¬Animal ⊓ ∃eats.Animal ?
  [Previously, the tableaux was shown to have a clash]
- Apply the ¬∀ equivalence rule:

  ∀eats.¬Animal ⊓ ∃eats.Animal =

  ¬∃eats.Animal ⊓ ∃eats.Animal =

  ¬ P ⊓ P for P= ∃eats.Animal

  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)

$\phi_{\forall R.C}$ (x) = ∀y R(x, y) ⇒ ϕ_C(y) [for CN: ∀y R(x, y) ⇒ CN(y)]

$\phi_{\exists R.C}$ (x) = ∃y R(x, y) ∧ ϕ_C(y) [for CN: ∃y R(x, y) ∧ CN(y)]

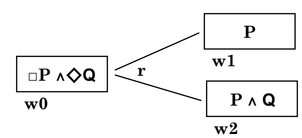
Modal Logics

□ Necessity/All time/Knows

[□P]$_v$ iff ∀w r(v,w) ⇒ [P]$_w$

◇ Possibility/Some time/Believes

[◇P]$_v$ iff ∃w r(v,w) ∧ [P]$_w$



DL and (multi) modal K have the same duality between operators

¬∀R.C = ∃R.¬C	¬□$_R$ P = ◇$_R$ ¬P
¬∃R.C = ∀R. ¬C	¬◇$_R$ P = □$_R$ ¬P

Description Logics and their properties

- ALC
  - Sound and complete subsumption testing
- ALCN
  - ALC + number restriction $\geq$ n R
- ALC_{R+}
  - ALC + transitively 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

CN, DN	Atomic concepts	Non-empty sets CN^I, $DN^I \subseteq \Delta^I$
$\perp^I$	owl:Nothing	ϕ
$\top^I$	owl:Thing	Δ^I
$(\neg C)^I$	Full Negation	$\Delta^I \setminus C^I$
$(C \sqcup D)^I$	Union	$C^I \cup D^I$
$(C \sqcap D)^I$	Intersection	$C^I \cap D^I$
$(\forall R.C)^I$	Value restriction	$\{x \in \Delta^I \mid \forall y <x,y> \in R^I \Rightarrow y \in C^I\}$
$(\exists R.C)^I$	Full existential quantification	$\{x \in \Delta^I \mid \exists y <x,y> \in R^I \wedge y \in C^I\}$

Terminological axioms: Inclusions and equalities

Concepts: $C \sqsubseteq D$ iff $C^I \subseteq D^I$

$C \equiv D$ iff $C^I = D^I$

OWL Datatypes

- Datatypes $\Delta^I{}_D$ are distinct from Object types Δ^I
  - A datatype relation U, e.g. age, relates an object type, e.g. Person to an integer
    - » ∃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
    - » age:29 [the set of things age 29]

D	Data Range	$D^I \subseteq \Delta_D{}^I$
$(\forall U.D)^I$	Value restriction	$\{x \in \Delta^I \mid \forall y <x,y> \in U^I \Rightarrow y \in D^I\}$
$(\exists U.D)^I$	Full existential quantification	$\{x \in \Delta^I \mid \exists y <x,y> \in U^I \wedge y \in D^I\}$

Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. J. of Web Semantics, 1(1):7-26, 2003.

OWL Number Restrictions

- OWL adds (unqualifying) number restrictions to ALC

$\geq n\ R$
  - Defines the set of instances, x, for which there n or more instances, y, such that R(x, y)
  - BusyParent $\equiv$ ≥ 3 hasChild

$\leq n\ R$
  - Defines the set of instances, x, for which there n or less instances, y, such that R(x, y)

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

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 $\sqcap$ D

Goal: C $\sqcap$ D

L(a0) = {C $\sqcap$ D}

L(a0) = {C, D} no clash

Disjointness means: $\top \sqsubseteq \neg C \sqcup \neg D$ [equivalent to $C \sqcap D \sqsubseteq \bot$]

L(a0) = {C, D, $\neg$C $\sqcup$ $\neg$D}

i. L(a0) = {C, D, $\neg$C} clash

ii. L(a0) = {C, D, $\neg$D} clash

OWL

- Datatypes $\Delta^I{}_D$ and Object types Δ^I

BN, CN	Non-empty sets $BN^I, CN^I \subseteq \Delta^I$
D	$D^I \subseteq \Delta_D{}^I$
$(B \sqcup C)^I$	$\{x \in \Delta^I \mid x \in B^I \vee x \in C^I\}$
$(B \sqcap C)^I$	$\{x \in \Delta^I \mid x \in B^I \wedge x \in C^I\}$
$(\forall R.C)^I$	$\{x \in \Delta^I \mid \forall y\ (\langle x,y\rangle \in R^I \Rightarrow y \in C^I)\}$
$(\exists R.C)^I$	$\{x \in \Delta^I \mid \exists y\ \langle x,y\rangle \in R^I \wedge y \in C^I\}$
$(\forall U.D)^I$	$\{x \in \Delta^I \mid \forall y\ (\langle x,y\rangle \in U^I \Rightarrow y \in D^I)\}$
$(\exists U.D)^I$	$\{x \in \Delta^I \mid \exists y\ \langle x,y\rangle \in U^I \wedge y \in D^I\}$

OWL-DL Cardinality

- Cardinality

BN, CN	Non-empty sets $BN^I, CN^I \subseteq \Delta^I$
$(\forall R.C)^I$	$\{x \in \Delta^I \mid \forall y\ (\langle x,y\rangle \in R^I \Rightarrow y \in C^I)\}$
$(\exists R.C)^I$	$\{x \in \Delta^I \mid \exists y\ \langle x,y\rangle \in R^I \wedge y \in C^I\}$
$(\geq n\ R)^I$	$\{x \in \Delta^I \mid \#(\langle x,y\rangle \in R^I) \geq n\}$
$(\leq n\ R)^I$	$\{x \in \Delta^I \mid \#(\langle x,y\rangle \in R^I) \leq n\}$

hasWheel^I = {<a0,a1> <a0,a2>} therefore:
≥ 0 hasWheel; ≥ 1 hasWheel; ≥ 2hasWheel; and ≤ 2 hasWheel; ≤ 3 hasWheel …

OWL-DL Cardinality

Bicycle ≡ ≥ 2 hasWheel $\sqcap \leq 2$ hasWheel

$\sqcap$ $\forall$hasPart. $\neg$Engine

- Unicyles 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

OWL Domain and Range Axioms

Domain and range specifications

domain(R, C) :: ≥ 1 R $\sqsubseteq$ C

Consider:

1) $\exists$hasChild.Male :anything with a male child

2) $\boxed{\text{Person}}$ $\sqcap$ $\exists$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

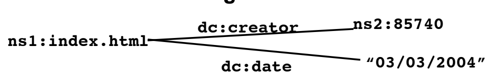
range(R, C) :: $\top \sqsubseteq \forall$R.C

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>

```
<http://www.example.org/index.html
 http://purl.org/dc/elements/1.1/creator
 http://www.example.org/staffid/85740>
```

- Subjects and objects are viewed as nodes in a graph, where predicates label the edges



- 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:Property`	the class of properties, an instance of `rdfs:Class`
`rdfs:Resource`	the class of everything
`rdfs:Literal`	the class of literal values e.g. string, integer
`rdfs:Class`	the class of RDF classes
`rdf:type`	the instance-of relation
`rdfs:domain`	domain definition, an instance of `rdf:Property`
`rdfs:range`	range definition, an instance of `rdf:Property`
`rdfs:subClassOf`	subclass relation
`rdfs:subPropertyOf`	subproperty relation

- 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

OWL/ALC DL Syntax		OWL Abstract Syntax
$(\neg C)$	Full Negation	`< complementOf C >`
$(C \sqcup D)$	Union	`< unionOf C D >`
$(C \sqcap D)$	Intersection	`< intersectionOf C D >`
$(\forall R.C)$	Value restriction	`< Restriction < onProperty R > < allValuesFrom C >>`
$(\exists R.C)$	Full existential quantification	`< Restriction < onProperty R > < someValuesFrom C >>`
$(C \sqcap D) = \bot$	Disjoint concepts	`< disjoint C D >`
$C \sqsubseteq D$	Subclass of /subsumes	`< C <subClassOf D>>`
$C \equiv D$	Equivalent	`<C <equivalentClass D>>`

OWL in RDF/XML Syntax

Class definitions $C \sqsubseteq D$ and Property restrictions $\forall R.C$ in RDF/XML syntax:

DieselEngine is a subclass of Engine: DieselEngine $\sqsubseteq$ Engine

```
<owl:Class rdf:ID ="DieselEngine">
    <rdfs:subClassOf rdf:resource="&base;Engine"/>
</owl:Class>
```

CarPart is a subclass of the parts of the Car:

CarPart $\sqsubseteq$ $\forall$partOf.Car

```
<owl:Class rdf:ID="CarPart">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="&base;partOf"/>
        <owl:allValuesFrom rdf:resource="#Car"/>
      </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

defined locally

imported

<owl:Class> is used to specify the rdf:type

rdf:ID introduces new terms (compare with rdf:about to refer to terms)

&base; is a namespace (assumed to be defined)

OWL in RDF/XML Syntax

CarEngine is equivalent to the intersection of Engine and $\forall$partOf.Car :

CarEngine $\equiv$ Engine $\sqcap$ $\forall$partOf.Car

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

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