Reasoning in DL

Prof. Mauro Gaspari
Dipartimento di Informatica Scienze e Ingegneria (DISI)
mauro.gaspari@unibo.it

Extending ALC

In many cases, the expressive power of $\mathcal{ALC}$ does not suffice:

- an elephant has precisely four legs
- every elephant has a bodypart which is a trunk and every trunk is a bodypart of an elephant

Many extensions of $\mathcal{ALC}$ have been developed, for example:

- qualified number restrictions $(\leq n\ R\ C)$ and $(\geq n\ R\ C)$
- inverse roles R^- to be used in existential and universal restriction

But: Increasing expressivity also increases computational complexity

$\Longrightarrow$!! tradeoff between expressivity and computational complexity !!

ALC extensions

- **S**: often used for ALC extended with transitive roles: e.g.,

  transitive(Ancestor)

- Additional letters indicate other extensions:
  - **H for role hierarchy** (e.g., hasDaughter $\sqsubseteq$ hasChild)
  - **O for nominals/**singleton classes (e.g., {Italy})
  - **I for inverse roles** (e.g., inverse(HasSister,SisterOf))
  - **Q for qualified number restrictions**: e.g.,
    ≤ 2 hasChild.Female
    ≥ 1 hasParent.Male
  - **N for number restrictions** (e.g., ≤ 2 hasChild.T)
  - **R for complex role inclusion (e.g., R1 $\circ$ R2) (+** reflexive, irreflexive and disjointness),

Lisp syntax for DL

$\top$	top or *top*
$\bot$	bottom or *bottom*
$\neg A$	(not A)
$C \sqcap D$	(and C D)
$C \sqcup D$	(or C D)
$\forall R.C$	(all R C)
$\exists R.\top$	(some R top)
$\exists R.C$	(some R C)
$\exists_{\geq n} R$	(≥n R) or (at-least n R)
$\exists_{\leq n} R$	(≤n R) or (at-most n R)

Nominals

- The term **nominals** is used when individual names that usually appears in the Abox are used in the description language (Tbox).
- They are used to describe concepts enumerating theit contents.
- The most basic is **set** (one-of is also used):
  - one-of(a_1,…,a_n) or {a_1,…,a_n} where a_i are individuals.
  - Semantics

$$\{a_1, \ldots, a_n\}^{\mathcal{I}} \;=\; \{a_1^{\mathcal{I}}, \ldots, a_n^{\mathcal{I}}\}.$$

  - Example: we can define the concept covid RedArea as:

    one-of(Lombardia, EmiliaRomagna, Veneto)

Fill constructor

- Also the **fill** costructor that concerns roles is defined using individuals for a given role R has the following syntax:

  R:a

- Intuitively it indicates all the pairs of the role R which include a in the range, in other words have a as a filler..
- Semantics: $(R : a)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid (d, a^{\mathcal{I}}) \in R^{\mathcal{I}}\}$
- The fill operator can be implemented with existensial quantifier and nominal as follows: R:a ≡ ∃R.{a}
- Example: Department : DISI, DIFA

Terminological Axioms

- Terminological axioms make statements about how concepts or roles are related to each other.
- Let C, D concepts and R, S roles, terminological axioms have the form:
  - Inclusions: $C \sqsubseteq D$ $(R \sqsubseteq S)$
  - Equalities: $C \equiv D$ $(R \equiv S)$
- Semantics considering concepts:
  - $C \sqsubseteq D$ if $C^I \subseteq D^I$
  - $C \equiv D$ if $C^I = D^I$
- For roles it is similar but it invloves set of pairs.

Definitions

- If T is a set of axioms, then **I satisfies T** iff I satisfies each element of T.
- If I satisfies a set of axioms T it is a **model** of T.
- Two sets of axioms T1 and T2 are **equivalent** if they have the same models.
- An equality whose left-hand side is an atomic concept is a **definition.** They are used to introduce symbolic names for complex descriptions.

$$\text{Mother} \equiv \text{Woman} \sqcap \exists \text{hasChild.Person}$$

$$\text{Parent} \equiv \text{Mother} \sqcup \text{Father}$$

- **Tbox**: a finite set of definitions where for every atomic concept there is at most one axiom.

Example

$$
\begin{aligned}
\mathsf{Woman} &\equiv \mathsf{Person} \sqcap \mathsf{Female} \\
\mathsf{Man} &\equiv \mathsf{Person} \sqcap \neg\mathsf{Woman} \\
\mathsf{Mother} &\equiv \mathsf{Woman} \sqcap \exists \mathsf{hasChild}.\mathsf{Person} \\
\mathsf{Father} &\equiv \mathsf{Man} \sqcap \exists \mathsf{hasChild}.\mathsf{Person} \\
\mathsf{Parent} &\equiv \mathsf{Father} \sqcup \mathsf{Mother} \\
\mathsf{Grandmother} &\equiv \mathsf{Mother} \sqcap \exists \mathsf{hasChild}.\mathsf{Parent} \\
\mathsf{MotherWithManyChildren} &\equiv \mathsf{Mother} \sqcap \geqslant 3\, \mathsf{hasChild} \\
\mathsf{MotherWithoutDaughter} &\equiv \mathsf{Mother} \sqcap \forall \mathsf{hasChild}.\neg\mathsf{Woman} \\
\mathsf{Wife} &\equiv \mathsf{Woman} \sqcap \exists \mathsf{hasHusband}.\mathsf{Man}
\end{aligned}
$$

Defined and Primitive Concepts

- Let T be a Tbox we can divide the atomic concepts of T in two subsets:
    - **Name simbols (defined concepts)**: that occur in the left-hand-side of some axiom.
    - **Base symbols (primitive concepts)**: that occur only in the right-hand side.
- A **base interpretation** for T is an interpretation that interprets only the base symbols.
- Let J be a base interpretation, an interpretation I that interprets also the name symbols is **an extension of J** if $\Delta^I = \Delta^J$, and if it agrees with J for the base symbols.

Properties of Tbox

- We say that T is **definitorial** if every base interpretation has exactly one extension that is a model of T.
- If we know what the base symbols stand for, and T is definitorial, then the meaning of the name symbols is completely determined.
- If a terminology is definitorial, then every equivalent terminology is also definitorial.
- The question whether a terminology is definitorial or not is related to the question whether or not its definitions are cyclic.
- If a terminology T is acyclic, then it is definitorial.

Abox

- It describes a state of an application domain in terms of concepts and roles.
- Some of the concept and role atoms in the ABox may be defined names of the TBox.
- In the ABox, one introduces individuals, by givingthem names, and one asserts properties of these individuals.
- Example:

MotherWithoutDaughter(MARY)
hasChild(MARY, PETER)
hasChild(MARY, PAUL)

Father(PETER)
hasChild(PETER, HARRY)

Open World Semantics

- The schema of a database is compared to the TBox and the instance with the actual data is compared to the ABox. However, the semantics of ABoxes differs from the usual semantics of database instances.
- Database instance represents exactly one interpretation (that of the DB schema), an ABox represents many different interpretations (all its models).
- Absence of information in a database instance is interpreted as negative information (**closed-world semantics**), while absence of information in an ABox only indicates lack of knowledge (**open-world semantics**).
- This means that, while the information in a database is always understood to be complete, the information in an ABox is in general viewed as being incomplete.

Example

- If we know hasChild(PETER, HARRY)
  - In a DB this is understood as a representation of the fact that Peter has only one child, Harry.
  - In an ABox, the assertion only expresses that, in fact, Harry is a child of Peter.
- The ABox has several models, some in which Harry is the only child and others in which he has brothers or sisters.
- Consequently, even if one also knows (by an assertion) that Harry is male, one cannot deduce that all of Peter’s children are male.
- The only way of stating in an ABox that Harry is the only child is by by adding the assertion ≤1 hasChild)(PETER).

Reasoning in DL

- A KB in DL (like in FOL) contains implicit knowledge that can be made explicit through inference.
- A DL KB (TBox and ABox) has a semantics that makes it equivalent to a set of axioms in first-order predicate logic.
- Intuitively, the semantics of a DL KB extends the ABox with all the possible concepts and roles that can be inferred according to the Tbox.
- In DL there is one main inference problem, namely the consistency check for ABoxes, to which all other inferences can be reduced.

ALC as a fragment of FOL

concept names A	$\Longleftrightarrow$	unary predicates P_A
role names R	$\Longleftrightarrow$	binary predicates P_R
concepts	$\Longleftrightarrow$	formulas with one free variable

$$
\begin{aligned}
\varphi^x(A) &= P_A(x) \\
\varphi^x(\neg C) &= \neg\varphi^x(C) \\
\varphi^x(C \sqcap D) &= \varphi^x(C) \wedge \varphi^x(D) \\
\varphi^x(C \sqcup D) &= \varphi^x(C) \vee \varphi^x(D) \\
\varphi^x(\exists R.C) &= \exists y.P_R(x,y) \wedge \varphi^y(C) \\
\varphi^x(\forall R.C) &= \forall y.P_R(x,y) \rightarrow \varphi^y(C)
\end{aligned}
$$

φ^y symmetric

with x and y exchanged

Note:
- two variables suffices (no "=", no constants, no function symbols)
- formulas obtained by translation have "guarded" structure
- not all DLs are purely first-order (transitive closure, etc.)

Reasoning with Concepts

- A knowledge engineer models a domain creating a **terminological KB** T, defining new concepts, possibly in terms of others that have been defined before.
- During this process, it is important to find out whether a newly defined concept makes sense or whether it is contradictory.
- A concept is **satisfiable** with respect to T if there is some interpretation that satisfies the axioms of T (that is, a model of T) such that the concept denotes a nonempty set in that interpretation.
- A concept is said to be **unsatisfiable** with respect to T if such a model does not exist,
- **Checking satisfiability of concepts is a key inference**: a number of other inferences for concepts can be reduced to the (un)satisfiability.

Subsumption

- The subsumption problem checks whether some concept is more general than another one.
- A **concept C is subsumed by a concept D** with respect to T if in every model of T the set denoted by C is a subset of the set denoted by D ($C^I \subseteq D^I$).
- We write:

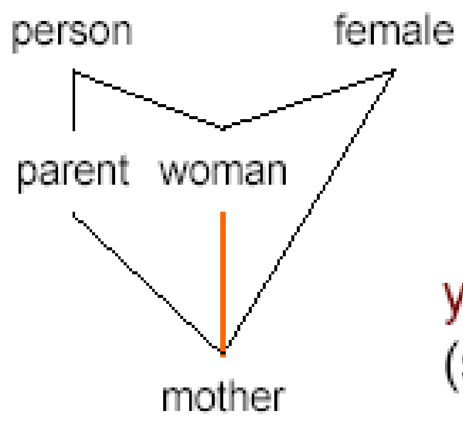
$$C \sqsubseteq_T D \text{ or}$$

$$T \models C \sqsubseteq D.$$

- Algorithms that check subsumption are also employed to organize the concepts of a TBox in a taxonomy according to their generality.

Example

- Consider the question
  Is a mother always a woman?
- Subsumes the concept woman the concept mother?
- Description logic reasoners offer the computation of a subsumption hierarchy (taxonomy) of all named concepts



parent $\equiv$ person $\sqcap$ $\exists$has_child.person
woman $\equiv$ person $\sqcap$ female
mother $\equiv$ parent $\sqcap$ female

yes, woman subsumes mother
(see also proof on previous slide)

Other Relationships

- Two concepts C and D are **equivalent** with respect to T if $C^I = D^I$ for every model I of T .
- In this case we write:

  $C \equiv_T D$ or

  $T \models C \equiv D$.

- Two concepts C and D are **disjoint** with respect to T if $C^I \cap D^I = \emptyset$ for every model I of T .

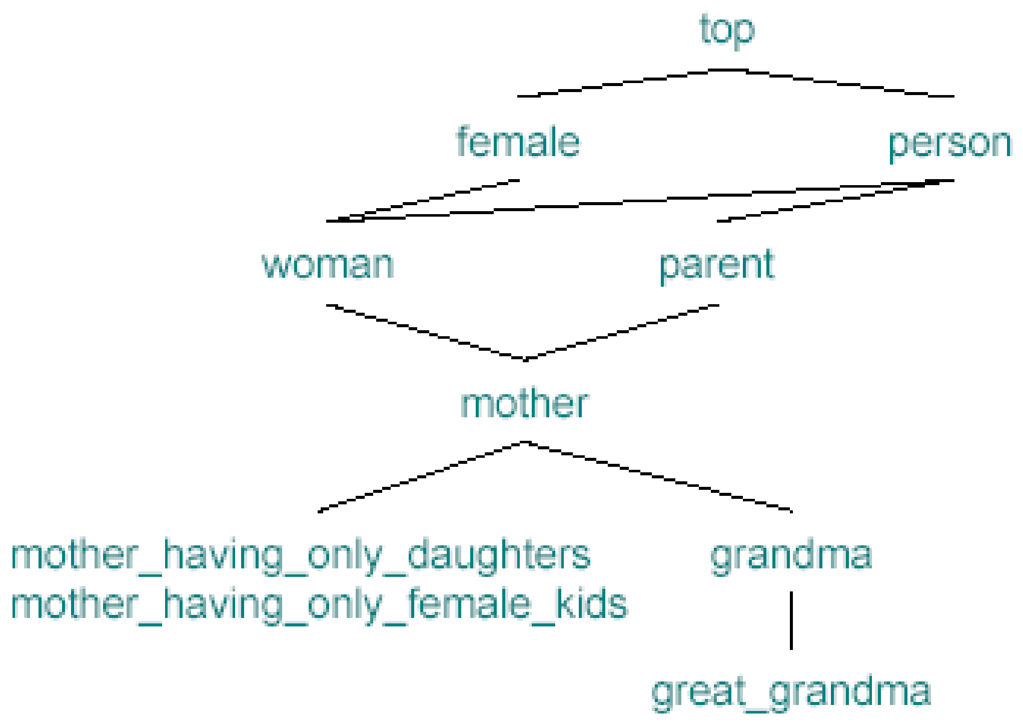
Example

- woman $\equiv$ person $\sqcap$ female
- parent $\equiv$ person $\sqcap$ $\exists$has_child.person
- mother $\equiv$ parent $\sqcap$ female
- mother_having_only_female_kids $\equiv$ mother $\sqcap$ $\forall$has_child.female
- mother_having_only_daughters $\equiv$ woman $\sqcap$ parent $\sqcap$ $\forall$has_child.woman

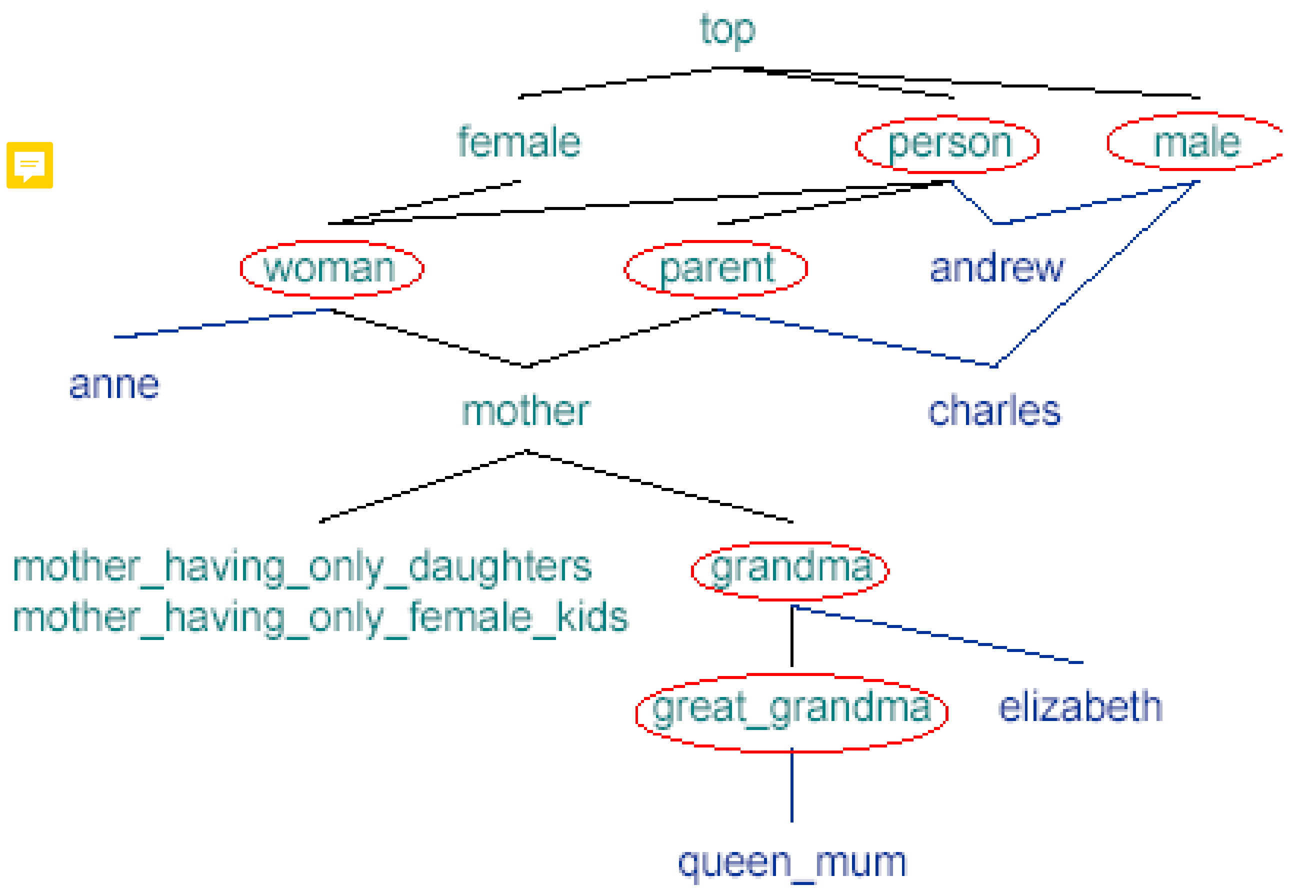
equivalent

- grandma $\equiv$ woman $\sqcap$ $\exists$has_child.parent
- great_grandma $\equiv$ woman $\sqcap$ $\exists$has_child.$\exists$has_child.parent

Taxonomy


top
female
person
woman
parent
mother
mother_having_only_daughters
mother_having_only_female_kids
grandma
great_grandma

Adding individuals



Reduction to Subsumption

- For concepts C, D we have

  i. C is **unsatisfiable** ⇔ C is subsumed by ⊥;

  ii. C and D are **equivalent** ⇔ C is subsumed by D and D is subsumed by C;

  iii. C and D are **disjoint** ⇔ C ⊓ D is subsumed by ⊥.

- These statements also hold with respect to a Tbox.
- Given that all DL languages contains ⊓ and almost all ⊥ most DL system that can check subsumption can perform all four inferences.

Reduction to Unsatisfiability

- For concepts C, D we have

  i. C is **subsumed** by D ⇔ C ⊓ ¬D is unsatisfiable;

  ii. C and D are **equivalent** ⇔ both (C ⊓ ¬D) and (¬C ⊓ D) are unsatisfiable;

  iii. C and D are **disjoint** ⇔ C ⊓ D is unsatisfiable.

- The statements also hold with respect to a Tbox.
- Considering satisfiability checking as the principal inference gave rise to a new kind of algorithms for reasoning in DLs, which can be understood as specialized tableaux calculi.

Reasoning Algorithms

- If a DL language provides negation inference can be reduced to satisfiability.
- Otherwise, if a DL language does not allow negation **structural subsumption algorithms** can be used.
- These algorithms are usually very efficient, but they are only complete for rather simple languages with little expressivity.
- In particular, DL with (full) negation and disjunction cannot be handled by structural subsumption algorithms.
- For more expressive DL languages **tableau-based algorithms** have turned out to be very useful.

Structural Subsumption

- These algorithms usually proceed in two phases.
  - First, the descriptions to be tested for subsumption are normalized
  - Then the syntactic structure of the normal forms is compared.
- We explain the ideas underlying this approach for the small language FL0 , which allows for conjunction (C ⊓ D) and value restrictions (∀R.C).
- Also the bottom concept (⊥), atomic negation (¬A), and number restrictions (≤ n R and ≥ n R) can be handled with this tehnique.

Normalization

Flatten all embedded conjunctions :

$$A \sqcap (B \sqcap C) \rightarrow A \sqcap B \sqcap C$$

Factorize all conjunctions of universal quantifiers over the same role

$$\forall R.C \sqcap \forall R.D \rightarrow \forall R.(C \sqcap D)$$

Syntactic Comparison

- The Subsumes(C , D) algorithm

Let $C = C_1 \sqcap \cdots \sqcap C_n$ and $D = D_1 \sqcap \cdots \sqcap D_m$
Subsumes(C, D) returns **true** iff for all C_i :

1. if C_i is atomic or of the form $\exists R$ then there exists D_j such that $C_i = D_j$;
2. if C_i is of the form $\forall R.C'$ then there exists D_j of the form $\forall R.D'$ such that Subsumes(C', D')

DL with structural subsumption

- Soundness: IF $KB^{DL} \vdash^{SS} 1$ THEN $KB^{DL} \models C \sqsubseteq D$
- Completeness: IF $KB^{DL} \models C \sqsubseteq D$ THEN $KB^{DL} \vdash^{SS} 1$
- Decidable.
- Efficient: Polynomial (CxD)
- Expressive Power: POOR, DL with (full) negation and disjunction cannot be handled.

Tableau alghorithms

- Tableau algorithm prove the unsatisfiability of a concept by trying to build a model.
- They take advantage of the "tree model property": if there is a model then there is a model that has a tree shape (the object-relation graph is a tree).
- $C \sqsubseteq D$ iff $C \sqcap \neg D$ is unsatisfiable.

Tableau algorithm

To test the satisfiability of C.
The algorithm tries to build a model I in which $I(C)$ is not empty.

1. put C in negative normal form (all negations beside atomic concept)
2. crate an initial set of ABoxes: $\{\{C(a)\}\}$
3. exhaustively apply the production rules
4. if there is an open ABox (without contradiction of the type $P(x)$ et $\neg P(x)$) in the resulting ABox set, C is satisfiable.

Tableau Rules

For an ABox $\mathcal{A}$ generate one or two new ABoxes $\mathcal{A}'$ and $\mathcal{A}''$

- $\rightarrow_{\sqcap}$ rule if $\mathcal{A}$ contains $(C \sqcap D)(x)$ but not $C(x)$ and $D(x)$ then $A' = \mathcal{A}' = \mathcal{A} \cup \{C(x), D(x)\}$.
- $\rightarrow_{\sqcup}$ rule if $\mathcal{A}$ contains $(C \sqcup D)(x)$ but neither $C(x)$ nor $D(x)$ then $\mathcal{A}' = \mathcal{A} \cup \{C(x)\}$ and $\mathcal{A}'' = \mathcal{A}A \cup \{D(x)\}$.
- $\rightarrow_{\exists}$ rule if $\mathcal{A}$ contains $(\exists r.C)(x)$ but no individual name z such that $C(z)$ and $r(x,z)$ are in A then $\mathcal{A}' = \mathcal{A} \cup \{C(y), r(x,y)\}$.
- $\rightarrow_{\forall}$ rule if $\mathcal{A}$ contains $(\forall r.C)(x)$ and $r(x,y)$ but not $C(y)$ then $\mathcal{A}' = \mathcal{A} \cup \{C(y)\}$.

DL with Tableau

- Soundness: IF $KB^{DL} \vdash^{T} \varnothing$ THEN $KB^{DL} \models C \sqcap \neg D$
- Completeness: IF $KB^{DL} \models C \sqcap \neg D$ THEN $KB^{DL} \vdash^{T} \varnothing$
- Decidable.
- Efficient: GOOD it depends from the DL language.
- Expressive Power: GOOD
- See the complexity navigator here:

  http://www.cs.man.ac.uk/~ezolin/dl/

Ontology

- An **ontology** is a formalization of a conceptualization.
- It involve terminological knowledge.
- DL are languages for representing terminological knowledge thus they are used for writing ontologies.
- **Ontological engineering**: emphasis on sharing and reusing existing ontologies in a distributed web based setting.
- Mechanisms for:
    - importing.
    - Matching.
    - Extending.
    - Merging.

Ontologies on the Web

- S + role hierarchy (H) + nominals (O) + inverse (I) + NR (N) = SHOIN
- SHOIN is the basis for W3C's OWL Web Ontology Language
- OWL approach:
    - OWL LITE: based on SHIN (+ Datatypes properties)
    - **OWL DL**: based on SHOIN (+ Datatypes properties)
    - OWL Full: expressive and semidecidable
- OWL 2: more expressive based on SROIQ (+ Datatypes properties)

OWL descriptions

Constructor	DL Syntax	Example	FOL Syntax
intersectionOf	$C_1 \sqcap \ldots \sqcap C_n$	Human $\sqcap$ Male	$C_1(x) \wedge \ldots \wedge C_n(x)$
unionOf	$C_1 \sqcup \ldots \sqcup C_n$	Doctor $\sqcup$ Lawyer	$C_1(x) \vee \ldots \vee C_n(x)$
complementOf	$\neg C$	$\neg$Male	$\neg C(x)$
oneOf	$\{x_1\} \sqcup \ldots \sqcup \{x_n\}$	{john} $\sqcup$ {mary}	$x = x_1 \vee \ldots \vee x = x_n$
allValuesFrom	$\forall P.C$	$\forall$hasChild.Doctor	$\forall y.P(x,y) \rightarrow C(y)$
someValuesFrom	$\exists P.C$	$\exists$hasChild.Lawyer	$\exists y.P(x,y) \wedge C(y)$
maxCardinality	$\leqslant nP$	$\leqslant$1hasChild	$\exists^{\leqslant n} y.P(x,y)$
minCardinality	$\geqslant nP$	$\geqslant$2hasChild	$\exists^{\geqslant n} y.P(x,y)$

OWL Descriptions

OWL Syntax	DL Syntax	Example
subClassOf	$C_1 \sqsubseteq C_2$	Human $\sqsubseteq$ Animal $\sqcap$ Biped
equivalentClass	$C_1 \equiv C_2$	Man $\equiv$ Human $\sqcap$ Male
subPropertyOf	$P_1 \sqsubseteq P_2$	hasDaughter $\sqsubseteq$ hasChild
equivalentProperty	$P_1 \equiv P_2$	cost $\equiv$ price
transitiveProperty	$P^+ \sqsubseteq P$	ancestor$^+$ $\sqsubseteq$ ancestor

OWL Syntax	DL Syntax	Example
type	$a : C$	John : Happy-Father
property	$\langle a, b \rangle : R$	$\langle$John, Mary$\rangle$: has-child

Tools and resources

- Racer (Renamed ABox And Concept Expression Reasoner) is a knowledge representation system that implements a highly optimized tableau calculus for the description logic SRIQ(D):

  https://www.ifis.uni-luebeck.de/index.php?id=385

- FaCT++ is a free (LGPL) highly optimised open-source C++-based tableaux reasoner for OWL 2 DL:

  https://bitbucket.org/dtsarkov/factplusplus

- Pellet, a Java-based reasoner for OWL 2, it supports SROIQ Description Logic:

  https://github.com/stardog-union/pellet

- Protégé ontology editor for OWL and OWL2.

  http://protege.stanford.edu/plugins/owl/

- W3C Web-Ontology (WebOnt) working group (OWL)

  http://www.w3.org/2001/sw/WebOnt/