Towards the Web Ontology Language (OWL)

Petr Křemen

petr.kremen@cvut.cz

Winter 2025

Outline

1. How to extend $\mathcal{ALC}$?
2. Web Ontology Language
   - OWL Profiles
   - Advanced Material (Optional)

How to extend $\mathcal{ALC}$?

Extending $\mathcal{ALC}$

- We have introduced $\mathcal{ALC}$. Its expressiveness is higher than the expressiveness of the propositional calculus, still it lacks many constructs needed for practical applications.

Extending $\mathcal{ALC}$

- We have introduced $\mathcal{ALC}$. Its expressiveness is higher than the expressiveness of the propositional calculus, still it lacks many constructs needed for practical applications.
- Let's take a look, how to extend $\mathcal{ALC}$ while preserving decidability.

Extending $\mathcal{ALC}$ (2)

$\mathcal{N}$ (Number restructions) are used for restricting the number of successors in the given role for the given concept.

concept D	its interpretation $D^{\mathcal{I}}$
$(\geq\ n\,R)$	$\left\{a \,\middle|\, \left|\{b \mid (a,b) \in R^{\mathcal{I}}\}\right| \geq n\right\}$
$(\leq\ n\,R)$	$\left\{a \,\middle|\, \left|\{b \mid (a,b) \in R^{\mathcal{I}}\}\right| \leq n\right\}$
$(=\ n\,R)$	$\left\{a \,\middle|\, \left|\{b \mid (a,b) \in R^{\mathcal{I}}\}\right| = n\right\}$

Example

- Concept $Woman \sqcap (\leq\ 3\ hasChild)$ denotes women who have at most 3 children.

Extending $\mathcal{ALC}$ (2)

$\mathcal{N}$ (Number restructions) are used for restricting the number of successors in the given role for the given concept.

concept D	its interpretation $D^{\mathcal{I}}$
$(\geq\ n\, R)$	$\left\{ a \,\middle|\, \left|\{b \mid (a,b) \in R^{\mathcal{I}}\}\right| \geq n \right\}$
$(\leq\ n\, R)$	$\left\{ a \,\middle|\, \left|\{b \mid (a,b) \in R^{\mathcal{I}}\}\right| \leq n \right\}$
$(=\ n\, R)$	$\left\{ a \,\middle|\, \left|\{b \mid (a,b) \in R^{\mathcal{I}}\}\right| = n \right\}$

Example

- Concept $Woman \sqcap (\leq\ 3\ hasChild)$ denotes women who have at most 3 children.
- What denotes the axiom $Car \sqsubseteq (\geq\ 4\ hasWheel)$?

Extending $\mathcal{ALC}$ (2)

$\mathcal{N}$ (Number restructions) are used for restricting the number of successors in the given role for the given concept.

concept D	its interpretation $D^{\mathcal{I}}$
$(\geq n\, R)$	$\left\{ a \,\middle|\, \left\|\{b \mid (a,b) \in R^{\mathcal{I}}\}\right\| \geq n \right\}$
$(\leq n\, R)$	$\left\{ a \,\middle|\, \left\|\{b \mid (a,b) \in R^{\mathcal{I}}\}\right\| \leq n \right\}$
$(= n\, R)$	$\left\{ a \,\middle|\, \left\|\{b \mid (a,b) \in R^{\mathcal{I}}\}\right\| = n \right\}$

Example

- Concept $Woman \sqcap (\leq 3\ hasChild)$ denotes women who have at most 3 children.
- What denotes the axiom $Car \sqsubseteq (\geq 4\ hasWheel)$?
- ... and $Bicycle \equiv (= 2\ hasWheel)$?

Extending $\mathcal{ALC}$ (3)

$\mathcal{Q}$ (Qualified number restrictions) are used for restricting the number of successors *of the given type* in the given role for the given concept.

concept D	its interpretation $D^{\mathcal{I}}$
$(\geq\ n\ R\ C)$	$\left\{ a \middle\vert\ \left\vert \{ b \mid (a,b) \in R^{\mathcal{I}} \wedge b^{\mathcal{I}} \in C^{\mathcal{I}} \} \right\vert \geq n \right\}$
$(\leq\ n\ R\ C)$	$\left\{ a \middle\vert\ \left\vert \{ b \mid (a,b) \in R^{\mathcal{I}} \wedge b^{\mathcal{I}} \in C^{\mathcal{I}} \} \right\vert \leq n \right\}$
$(=\ n\ R\ C)$	$\left\{ a \middle\vert\ \left\vert \{ b \mid (a,b) \in R^{\mathcal{I}} \wedge b^{\mathcal{I}} \in C^{\mathcal{I}} \} \right\vert = n \right\}$

Example

- Concept $Woman \sqcap (\geq\ 3\ hasChild\ Man)$ denotes women who have at least 3 sons.

Extending $\mathcal{ALC}$ (3)

$\mathcal{Q}$ (Qualified number restrictions) are used for restricting the number of successors *of the given type* in the given role for the given concept.

concept D	its interpretation $D^{\mathcal{I}}$
$(\geq\ n\, R\, C)$	$\left\{ a \middle\| \left\| \{ b \mid (a,b) \in R^{\mathcal{I}} \wedge b^{\mathcal{I}} \in C^{\mathcal{I}} \} \right\| \geq n \right\}$
$(\leq\ n\, R\, C)$	$\left\{ a \middle\| \left\| \{ b \mid (a,b) \in R^{\mathcal{I}} \wedge b^{\mathcal{I}} \in C^{\mathcal{I}} \} \right\| \leq n \right\}$
$(=\ n\, R\, C)$	$\left\{ a \middle\| \left\| \{ b \mid (a,b) \in R^{\mathcal{I}} \wedge b^{\mathcal{I}} \in C^{\mathcal{I}} \} \right\| = n \right\}$

Example

- Concept $Woman \sqcap (\geq\ 3\ hasChild\ Man)$ denotes women who have at least 3 sons.
- What denotes the axiom $Car \sqsubseteq (\geq\ 4\ hasPart\ Wheel)$?

Extending $\mathcal{ALC}$ (3)

$\mathcal{Q}$ (Qualified number restrictions) are used for restricting the number of successors *of the given type* in the given role for the given concept.

concept D	its interpretation $D^{\mathcal{I}}$
$(\geq\ n\ R\ C)$	$\left\{ a \,\middle|\, \left\|\{b \mid (a,b) \in R^{\mathcal{I}} \wedge b^{\mathcal{I}} \in C^{\mathcal{I}}\}\right\| \geq n \right\}$
$(\leq\ n\ R\ C)$	$\left\{ a \,\middle|\, \left\|\{b \mid (a,b) \in R^{\mathcal{I}} \wedge b^{\mathcal{I}} \in C^{\mathcal{I}}\}\right\| \leq n \right\}$
$(=\ n\ R\ C)$	$\left\{ a \,\middle|\, \left\|\{b \mid (a,b) \in R^{\mathcal{I}} \wedge b^{\mathcal{I}} \in C^{\mathcal{I}}\}\right\| = n \right\}$

Example

- Concept $Woman \sqcap (\geq\ 3\ hasChild\ Man)$ denotes women who have at least 3 sons.
- What denotes the axiom $Car \sqsubseteq (\geq\ 4\ hasPart\ Wheel)$?
- Which qualified number restrictions can be expressed in $\mathcal{ALC}$?

Extending $\mathcal{ALC}$ (4)

$\mathcal{O}$ (Nominals) can be used for naming a concept elements explicitely.

concept D	its interpretation $D^{\mathcal{I}}$
$\{a_1, \ldots, a_n\}$	$\{a_1^{\mathcal{I}}, \ldots, a_n^{\mathcal{I}}\}$

Example

- Concept $\{MALE, FEMALE\}$ denotes a gender concept that must be interpreted with at most two elements. Why at most ?

Extending $\mathcal{ALC}$ (4)

$\mathcal{O}$ (Nominals) can be used for naming a concept elements explicitely.

concept D	its interpretation $D^{\mathcal{I}}$
$\{a_1, \ldots, a_n\}$	$\{a_1^{\mathcal{I}}, \ldots, a_n^{\mathcal{I}}\}$

Example

- Concept $\{MALE, FEMALE\}$ denotes a gender concept that must be interpreted with at most two elements. Why at most ?
- $Continent \equiv \{EUROPE, ASIA, AMERICA, AUSTRALIA, AFRICA, ANTARCTICA\}$?

Extending $\mathcal{ALC}$ (5)

$\mathcal{I}$ (Inverse roles) are used for defining role inversion.

role S	its interpretation $S^{\mathcal{I}}$
R^-	$(R^{\mathcal{I}})^-$

Example

- Role *hasChild*$^-$ denotes the relationship *hasParent*.

Extending $\mathcal{ALC}$ (5)

$\mathcal{I}$ (Inverse roles) are used for defining role inversion.

role S	its interpretation $S^{\mathcal{I}}$
R^-	$(R^{\mathcal{I}})^-$

Example

- Role $hasChild^-$ denotes the relationship $hasParent$.
- What denotes axiom $Person \sqsubseteq (= 2\ hasChild^-)$?

Extending $\mathcal{ALC}$ (5)

$\mathcal{I}$ (Inverse roles) are used for defining role inversion.

role S	its interpretation $S^{\mathcal{I}}$
R^-	$(R^{\mathcal{I}})^-$

Example

- Role $hasChild^-$ denotes the relationship $hasParent$.
- What denotes axiom $Person \sqsubseteq (= 2\ hasChild^-)$?
- What denotes axiom $Person \sqsubseteq \exists hasChild^- \cdot \exists hasChild \cdot \top$?

Extending $\mathcal{ALC}$ (6)

$._{trans}$ (Role transitivity axiom) denotes that a role is transitive. Attention – it is not a transitive closure operator.

axiom α	$\mathcal{I} \models \alpha$ iff
trans(R)	$R^{\mathcal{I}}$ is transitive

Example

- Role *isPartOf* can be defined as transitive, while role *hasParent* is not. What about roles *hasPart*, *hasPart*$^{-}$, *hasGrandFather*$^{-}$?

Extending $\mathcal{ALC}$ (6)

.trans (Role transitivity axiom) denotes that a role is transitive. Attention – it is not a transitive closure operator.

axiom α	$\mathcal{I} \models \alpha$ iff
$trans(R)$	$R^{\mathcal{I}}$ is transitive

Example

- Role *isPartOf* can be defined as transitive, while role *hasParent* is not. What about roles *hasPart*, $hasPart^{-}$, $hasGrandFather^{-}$?
- What is a transitive closure of a relationship ? What is the difference between a transitive closure of $hasDirectBoss^{\mathcal{I}}$ and $hasBoss^{\mathcal{I}}$.

Extending $\mathcal{ALC}$ (7)

$\mathcal{H}$ (Role hierarchy) serves for expressing role hierarchies (taxonomies) – similarly to concept hierarchies.

axiom α	$\mathcal{I} \models \alpha$ iff
$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$

Example

- Role *hasMother* can be defined as a special case of the role *hasParent*.

Extending $\mathcal{ALC}$ (7)

$\mathcal{H}$ (Role hierarchy) serves for expressing role hierarchies (taxonomies) – similarly to concept hierarchies.

axiom α	$\mathcal{I} \models \alpha$ iff
$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$

Example

- Role *hasMother* can be defined as a special case of the role *hasParent*.
- What is the difference between a concept hierarchy $Mother \sqsubseteq Parent$ and role hierarchy $hasMother \sqsubseteq hasParent$.

Extending $\mathcal{ALC}$ (8)

$\mathcal{R}$ (role extensions) serve for defining expressive role constructs, like role chains, role disjunctions, etc.

axiom α	$\mathcal{I} \models \alpha$ iff
$R \circ S \sqsubseteq P$	$R^{\mathcal{I}} \circ S^{\mathcal{I}} \sqsubseteq P^{\mathcal{I}}$
$Dis(R, S)$	$R^{\mathcal{I}} \cap S^{\mathcal{I}} = \emptyset$
$\exists R \cdot Self$	$\{a \mid (a, a) \in R^{\mathcal{I}}\}$

Example

- How would you define the role *hasUncle* by means of *hasSibling* and *hasParent* ?

Extending $\mathcal{ALC}$ (8)

$\mathcal{R}$ (role extensions) serve for defining expressive role constructs, like role chains, role disjunctions, etc.

axiom α	$\mathcal{I} \models \alpha$ iff
$R \circ S \sqsubseteq P$	$R^{\mathcal{I}} \circ S^{\mathcal{I}} \sqsubseteq P^{\mathcal{I}}$
$Dis(R, S)$	$R^{\mathcal{I}} \cap S^{\mathcal{I}} = \emptyset$
$\exists R \cdot Self$	$\{a \vert (a, a) \in R^{\mathcal{I}}\}$

Example

- How would you define the role *hasUncle* by means of *hasSibling* and *hasParent* ?
- how to express that R is transitive, using a role chain ?

Extending $\mathcal{ALC}$ (8)

$\mathcal{R}$ (role extensions) serve for defining expressive role constructs, like role chains, role disjunctions, etc.

axiom α	$\mathcal{I} \models \alpha$ iff
$R \circ S \sqsubseteq P$	$R^{\mathcal{I}} \circ S^{\mathcal{I}} \sqsubseteq P^{\mathcal{I}}$
$Dis(R, S)$	$R^{\mathcal{I}} \cap S^{\mathcal{I}} = \emptyset$
$\exists R \cdot Self$	$\{a \mid (a, a) \in R^{\mathcal{I}}\}$

Example

- How would you define the role *hasUncle* by means of *hasSibling* and *hasParent* ?
- how to express that R is transitive, using a role chain ?
- Whom does the following concept denote $Person \sqcap \exists likes \cdot Self$?

Syntactic Sugar

- R is functional means $\top \sqsubseteq (\leq 1\ R)$,

Syntactic Sugar

- R is functional means $\top \sqsubseteq (\leq 1\ R)$,
- R is inverse functional means $\top \sqsubseteq (\leq 1\ R^-)$

Syntactic Sugar

- R is functional means $\top \sqsubseteq (\leq 1\ R)$,
- R is inverse functional means $\top \sqsubseteq (\leq 1\ R^{-})$
- R is reflexive means $\top \sqsubseteq \exists R \cdot Self$,

Syntactic Sugar

- R is functional means $\top \sqsubseteq (\leq 1\ R)$,
- R is inverse functional means $\top \sqsubseteq (\leq 1\ R^{-})$
- R is reflexive means $\top \sqsubseteq \exists R \cdot \mathit{Self}$,
- R is irreflexive means $\exists R \cdot \mathit{Self} \sqsubseteq \bot$,

Syntactic Sugar

- R is functional means $\top \sqsubseteq (\leq 1 R)$,
- R is inverse functional means $\top \sqsubseteq (\leq 1 R^{-})$
- R is reflexive means $\top \sqsubseteq \exists R \cdot Self$,
- R is irreflexive means $\exists R \cdot Self \sqsubseteq \bot$,
- R is symmetric means $R \sqsubseteq R^{-}$,

Syntactic Sugar

- R is functional means $\top \sqsubseteq (\leq 1\ R)$,
- R is inverse functional means $\top \sqsubseteq (\leq 1\ R^-)$
- R is reflexive means $\top \sqsubseteq \exists R \cdot Self$,
- R is irreflexive means $\exists R \cdot Self \sqsubseteq \bot$,
- R is symmetric means $R \sqsubseteq R^-$,
- R is asymmetric means $Dis(R, R^-)$,

Syntactic Sugar

- R is functional means $\top \sqsubseteq (\leq 1\ R)$,
- R is inverse functional means $\top \sqsubseteq (\leq 1\ R^{-})$
- R is reflexive means $\top \sqsubseteq \exists R \cdot Self$,
- R is irreflexive means $\exists R \cdot Self \sqsubseteq \bot$,
- R is symmetric means $R \sqsubseteq R^{-}$,
- R is asymmetric means $Dis(R, R^{-})$,
- R is transitive means $R \circ R \sqsubseteq R$

Syntactic Sugar

- R is functional means $\top \sqsubseteq (\leq 1\ R)$,
- R is inverse functional means $\top \sqsubseteq (\leq 1\ R^{-})$
- R is reflexive means $\top \sqsubseteq \exists R \cdot Self$,
- R is irreflexive means $\exists R \cdot Self \sqsubseteq \bot$,
- R is symmetric means $R \sqsubseteq R^{-}$,
- R is asymmetric means $Dis(R, R^{-})$,
- R is transitive means $R \circ R \sqsubseteq R$
- $I = J$ means $\{I\} \sqsubseteq \{J\}$ (individual equality assertions)

Syntactic Sugar

- R is functional means $\top \sqsubseteq (\leq 1\ R)$,
- R is inverse functional means $\top \sqsubseteq (\leq 1\ R^{-})$
- R is reflexive means $\top \sqsubseteq \exists R \cdot \mathit{Self}$,
- R is irreflexive means $\exists R \cdot \mathit{Self} \sqsubseteq \bot$,
- R is symmetric means $R \sqsubseteq R^{-}$,
- R is asymmetric means $\mathit{Dis}(R, R^{-})$,
- R is transitive means $R \circ R \sqsubseteq R$
- $I = J$ means $\{I\} \sqsubseteq \{J\}$ (individual equality assertions)
- $I \neq J$ means $\{I\} \sqsubseteq \neg\{J\}$ (individual equality assertions)

Syntactic Sugar

- R is functional means $\top \sqsubseteq (\leq 1\ R)$,
- R is inverse functional means $\top \sqsubseteq (\leq 1\ R^-)$
- R is reflexive means $\top \sqsubseteq \exists R \cdot Self$,
- R is irreflexive means $\exists R \cdot Self \sqsubseteq \bot$,
- R is symmetric means $R \sqsubseteq R^-$,
- R is asymmetric means $Dis(R, R^-)$,
- R is transitive means $R \circ R \sqsubseteq R$
- $I = J$ means $\{I\} \sqsubseteq \{J\}$ (individual equality assertions)
- $I \neq J$ means $\{I\} \sqsubseteq \neg\{J\}$ (individual equality assertions)
- $\neg R(I, J)$ means $\{I\} \sqsubseteq \neg\exists R \cdot \{J\}$ (negative property assertions)

Other extensions

Modal Logic introduces *modal operators* – possibility/necessity, used in multiagent systems.

Other extensions

Modal Logic introduces *modal operators* – possibility/necessity, used in multiagent systems.

Example

Other extensions

Modal Logic introduces *modal operators* – possibility/necessity, used in multiagent systems.

Example

- ($\Box$ represents e.g. the "believe" operator of an agent)

$$\Box(Man \sqsubseteq Person \sqcap \forall hasFather \cdot Man) \tag{1}$$

Other extensions

Modal Logic introduces *modal operators* – possibility/necessity, used in multiagent systems.

Example

- ($\Box$ represents e.g. the "believe" operator of an agent)

$$\Box(Man \sqsubseteq Person \sqcap \forall hasFather \cdot Man) \quad (1)$$

- As $\mathcal{ALC}$ is a syntactic variant to a multi-modal propositional logic, where each role represents the accessibility relation between worlds in Kripke structure, the previous example can be transformed to the modal logic as:

Vague Knowledge - fuzzy, probabilistic and possibilistic extensions

Other extensions

Modal Logic introduces *modal operators* – possibility/necessity, used in multiagent systems.

Example

- ($\Box$ represents e.g. the "believe" operator of an agent)

$$\Box(Man \sqsubseteq Person \sqcap \forall hasFather \cdot Man) \quad (1)$$

- As $\mathcal{ALC}$ is a syntactic variant to a multi-modal propositional logic, where each role represents the accessibility relation between worlds in Kripke structure, the previous example can be transformed to the modal logic as:
-

$$\Box(Man \implies Person \wedge \Box_{hasFather} Man) \quad (2)$$

Vague Knowledge - fuzzy, probabilistic and possibilistic extensions

Data Types ($\mathcal{D}$) allow integrating a data domain (numbers, strings), e.g. $Person \sqcap \exists hasAge \cdot 23$ represents the concept describing "23-years old persons".

Web Ontology Language

Description logics behind OWL

- From the previously introduced extensions, two prominent decidable supersets of $\mathcal{ALC}$ can be constructed:

Description logics behind OWL

- From the previously introduced extensions, two prominent decidable supersets of $\mathcal{ALC}$ can be constructed:
  - $\mathcal{SHOIN}$ is a description logics that backs OWL-DL.

Description logics behind OWL

- From the previously introduced extensions, two prominent decidable supersets of $\mathcal{ALC}$ can be constructed:
  - $\mathcal{SHOIN}$ is a description logics that backs OWL-DL.
  - $\mathcal{SROIQ}$ is a description logics that backs OWL2-DL.

Description logics behind OWL

- From the previously introduced extensions, two prominent decidable supersets of $\mathcal{ALC}$ can be constructed:
  - $\mathcal{SHOIN}$ is a description logics that backs OWL-DL.
  - $\mathcal{SROIQ}$ is a description logics that backs OWL2-DL.
  - Both OWL-DL and OWL2-DL are semantic web languages – they extend the corresponding description logics by:

Description logics behind OWL

- From the previously introduced extensions, two prominent decidable supersets of $\mathcal{ALC}$ can be constructed:
  - $\mathcal{SHOIN}$ is a description logics that backs OWL-DL.
  - $\mathcal{SROIQ}$ is a description logics that backs OWL2-DL.
  - Both OWL-DL and OWL2-DL are semantic web languages – they extend the corresponding description logics by:

    syntactic sugar – axioms NegativeObjectPropertyAssertion, AllDisjoint, etc.

Description logics behind OWL

- From the previously introduced extensions, two prominent decidable supersets of $\mathcal{ALC}$ can be constructed:
  - $\mathcal{SHOIN}$ is a description logics that backs OWL-DL.
  - $\mathcal{SROIQ}$ is a description logics that backs OWL2-DL.
  - Both OWL-DL and OWL2-DL are semantic web languages – they extend the corresponding description logics by:

    syntactic sugar – axioms NegativeObjectPropertyAssertion, AllDisjoint, etc.

    extralogical constructs – imports, annotations

Description logics behind OWL

- From the previously introduced extensions, two prominent decidable supersets of $\mathcal{ALC}$ can be constructed:
  - $\mathcal{SHOIN}$ is a description logics that backs OWL-DL.
  - $\mathcal{SROIQ}$ is a description logics that backs OWL2-DL.
  - Both OWL-DL and OWL2-DL are semantic web languages – they extend the corresponding description logics by:

    syntactic sugar – axioms NegativeObjectPropertyAssertion, AllDisjoint, etc.

    extralogical constructs – imports, annotations

    data types – XSD datatypes are used

From DL to OWL

All entities (concepts/roles/individuals) are identified by IRIs.

```
Prefix: : <http://ex.owl/>
Ontology: <http://ex.owl/o1>
  ObjectProperty: :hasChild
  Class: :Man
  Class: :FatherOfSons
    SubClassOf: :hasChild some owl:Thing and :hasChild only :Man
  Individual: :John
    Types: :FatherOfSons
```

classes – DL concepts (e.g. `ex:Man`, `ex:Employee`, etc.)

individuals – DL individuals (e.g. `ex:John`)

object/data properties – DL roles (e.g. `ex:hasChild`) / data roles (e.g. `ex:hasName`)

OWL namespace is `http://www.w3.org/2002/07/owl#`, prefixed as `owl:`.

OWL Ontology Header

```
Prefix: : <http://ex.owl/>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Ontology: <http://ex.owl/o3> <http://ex.owl/o3-v1>
  Import: <http://ex.owl/o4>
  Import: <http://ex.owl/o5>
  Annotations: rdfs:comment "An example ontology"@en,
               :creator :John
  AnnotationProperty: :creator
  Individual: :John
```

- An ontology is identified by

OWL Ontology Header

```
Prefix: : <http://ex.owl/>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Ontology: <http://ex.owl/o3> <http://ex.owl/o3-v1>
  Import: <http://ex.owl/o4>
  Import: <http://ex.owl/o5>
  Annotations: rdfs:comment "An example ontology"@en,
               :creator :John
  AnnotationProperty: :creator
  Individual: :John
```

- An ontology is identified by
  ontology IRI (`http://ex.owl/o3`) logically identifies an ontology (although it might be stored e.g. in a local file)

OWL Ontology Header

```
Prefix: : <http://ex.owl/>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Ontology: <http://ex.owl/o3> <http://ex.owl/o3-v1>
  Import: <http://ex.owl/o4>
  Import: <http://ex.owl/o5>
  Annotations: rdfs:comment "An example ontology"@en,
               :creator :John
  AnnotationProperty: :creator
  Individual: :John
```

- An ontology is identified by
  - ontology IRI (`http://ex.owl/o3`) logically identifies an ontology (although it might be stored e.g. in a local file)
  - version IRI (`http://ex.owl/o3-v1`) which is optional

OWL Ontology Header

```
Prefix: : <http://ex.owl/>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Ontology: <http://ex.owl/o3> <http://ex.owl/o3-v1>
  Import: <http://ex.owl/o4>
  Import: <http://ex.owl/o5>
  Annotations: rdfs:comment "An example ontology"@en,
               :creator :John
  AnnotationProperty: :creator
  Individual: :John
```

- An ontology is identified by
  - ontology IRI (`http://ex.owl/o3`) logically identifies an ontology (although it might be stored e.g. in a local file)
  - version IRI (`http://ex.owl/o3-v1`) which is optional
- `Import:` allows importing other ontologies (for backward compatibility with OWL 1, the imported ontology is syntactically included in case it has no `Ontology:` header)

OWL Ontology Header

```
Prefix: : <http://ex.owl/>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Ontology: <http://ex.owl/o3> <http://ex.owl/o3-v1>
  Import: <http://ex.owl/o4>
  Import: <http://ex.owl/o5>
  Annotations: rdfs:comment "An example ontology"@en,
               :creator :John
  AnnotationProperty: :creator
  Individual: :John
```

- An ontology is identified by
  - ontology IRI (`http://ex.owl/o3`) logically identifies an ontology (although it might be stored e.g. in a local file)
  - version IRI (`http://ex.owl/o3-v1`) which is optional
- `Import:` allows importing other ontologies (for backward compatibility with OWL 1, the imported ontology is syntactically included in case it has no `Ontology:` header)
- `Annotations:` allows arbitrary ontology annotations (creators, comments, backward compatibility, etc.)

DL Syntax vs. Manchester Syntax vs. Turtle

- DL
  $FatherOfSons \sqsubseteq \exists hasChild \cdot \top \sqcap \forall hasChild \cdot Man$
- OWL Manchester Syntax

```
  Class: :FatherOfSons
    SubClassOf: :hasChild some owl:Thing and :hasChild only :Man
```

- OWL / RDF serialization in Turtle

```
:FatherOfSons rdf:type owl:Class ;
    rdfs:subClassOf [ rdf:type owl:Class ;
    owl:intersectionOf ( [ rdf:type owl:Restriction ;
                              owl:onProperty :hasChild ;
                              owl:someValuesFrom owl:Thing]
                            [ rdf:type owl:Restriction ;
                              owl:onProperty :hasChild ;
                              owl:allValuesFrom :Man ] )
```

Annotations

Each resource can be assigned a set of annotations (i.e. classes, properties, reified axioms, or even annotations themselves):

```
Class: :FatherOfSons
  Annotations:
        :creator :John,
        Annotations: :creator :Jack
                rdfs:label "Father of sons"@en
  SubClassOf:
        Annotations: :creator :Mary
                :hasChild some owl:Thing and :hasChild only :Man
```

Question

What do different creators refer to ?

Punning

Should `ex:Dog` be considered a class (representing a set of dogs), or an individual (representing a particular species) ?

Punning is the mechanism of reusing the same IRI for entities of different type for the sake of metamodeling but certain typing constraints must be fulfilled to stay in OWL 2 DL.

OWL 2 DL Typing constraints

- All IRIs have to be declared to be either *class*, *datatype*, *object property*, *data property*, *annotation property*, *individual* in the *axiom closure of an ontology*
- Each IRI can be (declared/used as) only one of (object property, data property, annotation property)
- Each IRI can be (declared/used as) only one of (class, datatype)

Punning example

Correct:

```
Individual: ex:Dog
  Facts: ex:isExtinct false

Individual: ex:Lucky
  Types: ex:Dog
```

Incorrect:

```
Individual: ex:John
  Facts: ex:hasName ex:JohnsFirstName
  Facts: ex:hasName "John"@en
```

Property Expressions

... just inverse:

```
inverse :hasChild
```

Inverse property goes in the opposite direction. Inverse properties can be used in class frames, property frames as well as individuals frames.

Object Property Frames

```
ObjectProperty: :hasMother
    Characteristics: Functional, Irreflexive, Asymmetric
    Domain: :Person
    Range: :Woman
    SubPropertyOf: :hasParent
    EquivalentTo: inverse :isMotherOf
    DisjointWith: :hasFather
    InverseOf: :isMotherOf
    SubPropertyChain: :hasFather o :isWifeOf
```

`Characteristics` – selection of `Functional`,`InverseFunctional`, `Transitive`, `Reflexive`, `Irreflexive`, `Symmetric`, `Asymmetric` – interpreted in their mathematical sense

`Domain,Range` have the same meaning as in RDFS

`SubPropertyOf` specifies props representing supersets of the frame property

`EquivalentTo` specifies props semantically equivalent to the frame class

`DisjointWith` specifies props disjoint with the frame property

`InverseOf` specifies inverse props (like `inverse` property expression)

`SubPropertyChain` specifies a property composition

Data Property Frames

```
DataProperty: :hasBirthNumber
    Characteristics: Functional
    Domain: :Person
    Range: xsd:string
    SubPropertyOf: :hasIdentifyingNumber
```

The only **Characteristics** available is `Functional`. Other sections have the same meaning as for Object properties.

Basic Data Ranges

OWL 2 supports basic modeling constructs for custom data ranges:

`and,or,not` have the meaning of standard set intersection, union and complement,

```
(xsd:nonNegativeInteger and xsd:nonPositiveInteger)
    or xsd:string
```

Basic Data Ranges

OWL 2 supports basic modeling constructs for custom data ranges:

and,or,not have the meaning of standard set intersection, union and complement,

```
(xsd:nonNegativeInteger and xsd:nonPositiveInteger)
    or xsd:string
```

individual enumeration lists individuals belonging to a class expression.

```
{"true"^^xsd:boolean 1}
```

Facets

Facets restrict a particular datatype to a subset of its values.

```
xsd:integer[ >= 5, < 10  ]
```

Available facets

length, minLength, maxLength – string lengths

pattern – string regular expression

langRange – range of language tags

<=,<,>=,> – number comparison

New datatypes can be used by means of datatype frame axioms:

```
Datatype: :MyNumber
    EquivalentTo: xsd:integer[ >= 5, < 10  ]
```

Boolean operators

OWL 2 supports many class modeling constructs including boolean connectives, individual enumeration, and object/data value restrictions.

`owl:Thing`, `owl:Nothing` are two predefined OWL classes containing all (resp. no) individuals,

Boolean operators

OWL 2 supports many class modeling constructs including boolean connectives, individual enumeration, and object/data value restrictions.

`owl:Thing`, `owl:Nothing` are two predefined OWL classes containing all (resp. no) individuals,

`and`,`or`,`not` have the meaning of standard set intersection, union and complement,

```
(:FlyingObject and not :Bat) or :Pinguin
```

Boolean operators

OWL 2 supports many class modeling constructs including boolean connectives, individual enumeration, and object/data value restrictions.

`owl:Thing`, `owl:Nothing` are two predefined OWL classes containing all (resp. no) individuals,

`and`,`or`,`not` have the meaning of standard set intersection, union and complement,

```
(:FlyingObject and not :Bat) or :Pinguin
```

individual enumeration lists individuals belonging to a class expression.

```
{:John :Mary}
```

Object value Restrictions (1)

existential quantification says that a property filler exists (not necessarily in data !)

```
:hasChild some :Man
```

Object value Restrictions (1)

existential quantification says that a property filler exists (not necessarily in data !)

```
:hasChild some :Man
```

universal quantification says that each property filler belongs to a class

```
:hasChild only :Man
```

Object value Restrictions (1)

existential quantification says that a property filler exists (not necessarily in data !)

```
:hasChild some :Man
```

universal quantification says that each property filler belongs to a class

```
:hasChild only :Man
```

cardinality restriction restricts the number of property fillers

```
:hasPart exactly 2 :Wheel
:hasPart min 4 :Wheel
:hasPart max 1 :Wheel
```

Object Value Restrictions (2)

individual value restriction restricts a property filler to a specified individual

```
:hasChild value :John
```

Object Value Restrictions (2)

individual value restriction restricts a property filler to a specified individual

```
:hasChild value :John
```

self restriction restricts a property filler to the same individual

```
:trusts Self
```

Complex Value Restrictions

- analogous counterparts to the object value restrictions are available (except the Self restriction) as *data value restrictions*:

```
:hasName some xsd:string[length 2]
```

What does this class expression describe ?

```
(:hasPart only (not :Tail))
and (:hasPart max 2 (:hasPart some :Knee))
and (:doesAssignmentWith Self)
and (:hasGrade only xsd:string[pattern "[AB]"])
```

Class frames

```
Class: :Father
  SubClassOf: :Parent
  EquivalentTo: :Man and :hasChild some :Person
  DisjointWith: :Mother
  DisjointUnionOf: :HappyFather :SadFather
  HasKey: :hasBirthNumber
```

SubClassOf section defines axioms specifying supersets of the frame class

EquivalentTo section defines axioms specifying classes semantically equivalent to the frame class

DisjointWith section defines classes sharing no individuals with the frame class

DisjointUnionOf section defines classes that are mutually disjoint and union of which is semantically equivalent to the frame class

HasKey section defines a set of properties that build up a *key* for the class – all instances of `Father` sharing the same value for the key (`:hasBirthNumber`) are semantically identical

Individual Frames

```
Individual: :John
  Types: :Person , :hasName value "Johnny"
  Facts: :hasChild :Jack, not :hasName "Bob"
  SameAs: :Johannes
  DifferentFrom: :Jack
```

Individual frames contain assertions, subject of which is the individual.

- `Types` specifies class descriptions that are types (`rdf:type`) for the frame individual,
- `Facts` specifies the object and data property assertions,
- `SameAs` specifies individuals being semantically identical to the frame individual,
- `DifferentFrom` specifies individuals being semantically different to the frame individual

Unique Name Assumption

OWL **does not accept** unique name assumption, i.e. it is not known whether two individuals `:John` and `:Jack` represent the same object, or not. By `SameAs` and `DifferentFrom`, either possibility can be enforced.

```
Individual: :John
  Types: :hasChild exactly 1 owl:Thing
  Facts: :hasChild :Jack, :hasChild :Jim
```

Global Constraints

We have discussed the typing constraints. Additionally, there are syntactic constraints that ensure decidability of reasoning. These constraints must be fulfilled for each OWL 2 DL ontology:

simple object property
: are properties that have no direct or indirect (through property hierarchy) subproperties that are transitive or defined by means of a property chain.

```
ObjectProperty: :hasChild
  SubPropertyOf: :hasDescendant
ObjectProperty: :hasDescendant
  Characteristics: Transitive
  SubPropertyOf: :hasRelative
ObjectProperty: :hasSon
  SubPropertyOf: :hasChild
ObjectProperty: :hasDaughter
  SubPropertyOf: :hasChild
ObjectProperty: :hasUncle
  SubPropertyOf: :hasRelative
  SubPropertyChain: :hasParent o :hasSibling
```

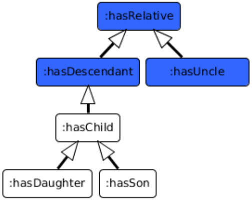


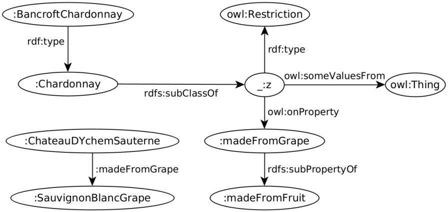
Figure: White properties are simple, blue ones are not.

Global Constraints (2)

Formal specification is in [**Patel-Schneider:12:OWOSS**], informally:

- `owl:topDataProperty` cannot be stated equal to any other data property (e.g. through `EquivalentTo` or `SubPropertyOf`).
- datatype definitions must be acyclic
- the following constructs are only allowed with *simple properties*:
  - cardinality restrictions (`min`, `max`, `exactly`),
  - self restriction ((Self)),
  - property characteristics `Functional`, `InverseFunctional`, `Irreflexive`,`Asymmetric`,
  - property axiom `DisjointWith`
- property chains must not be cyclic
- (restriction on anonymous individuals (that we haven't discussed))

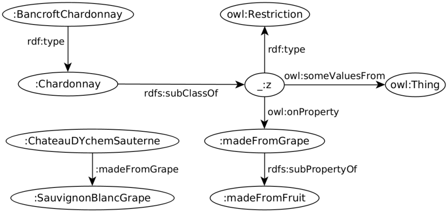
SPARQL Evaluation Semantics




```
PREFIX : <http://ex.org/e1>
SELECT ?x
WHERE { ?x :madeFromFruit _:d }
```

Simple-entailment No result.

SPARQL Evaluation Semantics

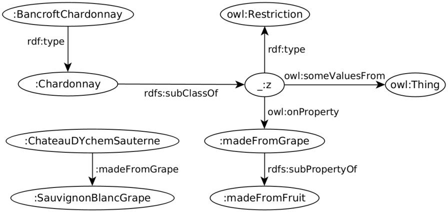



```
PREFIX : <http://ex.org/e1>
SELECT ?x
WHERE { ?x :madeFromFruit _:d }
```

Simple-entailment No result.

RDF-entailment No result.

SPARQL Evaluation Semantics



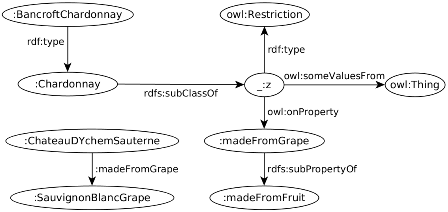

```
PREFIX : <http://ex.org/e1>
SELECT ?x
WHERE { ?x :madeFromFruit _:d }
```

Simple-entailment No result.

RDF-entailment No result.

RDFS-entailment One result: `?x=:ChateauDYchemSauterne`.

SPARQL Evaluation Semantics




```
PREFIX : <http://ex.org/e1>
SELECT ?x
WHERE { ?x :madeFromFruit _:d }
```

Simple-entailment No result.

RDF-entailment No result.

RDFS-entailment One result: ?x=:ChateauDYchemSauterne.

OWL-entailment Two results: ?x=:ChateauDYchemSauterne and ?x=:BancroftChardonnay.

```
Individual: :BancroftChardonnay
  Types: :Chardonnay
Class: :Chardonnay
  SubClassOf: :madeFromGrape some owl:Thing
```

OWL Profiles

OWL (2) Language Family

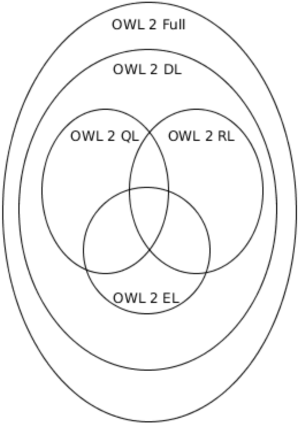
OWL (Full) interprets any RDF graph under OWL-RDF entailment regime (undecidable).

OWL 2 DL interprets OWL 2 ontologies (parsed only from **compliant** RDF graphs) by means of **decidable** $\mathcal{SROIQ}$ description logic semantics,

OWL 2 EL is a subset of OWL 2 DL for rich class taxonomies,

OWL 2 QL is a subset of OWL 2 DL for large data,

OWL 2 RL is a subset of OWL 2 DL with weaker rule-based semantic.



OWL 2 EL

$\sim$ EL++ description logic

- all axioms are limited to these class constructors $\exists R \cdot C$, $\exists R \cdot \{I\}$, $\exists R \cdot Self$, $C \sqcap D$
- inverse properties not allowed
- unavailable axioms:
  - $Dis(R, Q)$,
  - reflexive / functional / inverse functional / symmetric role R
- the most useful reasoning procedure is **subsumption checking** (polynomial time)
- e.g. for SNOMED-CT

OWL 2 QL

$\sim$ DL-Lite$_R$ description logic

- allowed subclasses[1] – A, $\exists R \cdot \top$,
- allowed superclasses – $C \sqcap D$, $\neg C$, $\exists R \cdot C$
- unavailable axioms:
  - $R \sqsubseteq S$ (subproperties),
  - functional / inverse functional / transitive R,
  - individual equality assertions,
  - negative property assertions,
- the most useful reasoning procedure is **query answering** – done by means of rewriting a conjunctive query into a set of database (SQL) queries (LOGSPACE)

[1] Note this also applies “syntactic sugar axioms” – equivalent classes, disjoint classes, etc.

OWL 2 RL

$\sim$ rule-based semantics of OWL 2 DL axioms

- allowed subclasses – $\{I\}$, $C \sqcap D$, $C \sqcup D$, $\exists R \cdot C$
- allowed superclasses – $C \sqcap D$, $\neg C$, $\exists R \cdot C$, $\forall R \cdot C$, $(\leq 1\ R\ C)$
- unavailable axioms – disjoint unions, reflexive object properties
- expressive, yet efficient reasoning – traded for weakened (rule-based) semantics of the constructs and axioms
  - no non-deterministic reasoning
  - no generation of new individuals

Advanced Material (Optional)

OWL 2 RDF-Based Semantics

defines an entailment $\models_{OWL2-RDF}$) allowing to **interpret all RDF graphs** (called *OWL 2 Full*)

- is an extension of *D*-entailment (inteprets the whole RDF graph)
- undecidable, but *incomplete* entailment rules are provided [**Schneider:12:OWO**]

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <http://example.org/2014-osw-14/>.
_:y a owl:Ontology .
_:x rdfs:subClassOf :Parent ;
    a owl:Restriction ;
:hasChild a owl:ObjectProperty .
:John :hasChild :Mary .
```

```
@prefix : <http://www.example.org/2014-osw-14/> .
:hasChild a rdf:Property .
:Mary a owl:NamedIndividual .
```

The following entailment holds:

$$G_1 \models_{OWL2-RDF} G_2$$

OWL 2 Direct Semantics

defines an entailment $\models_{OWL2-DL}$ in terms of the $\mathcal{SROIQ}(\mathcal{D})$ DL.

- interprets only "logically-backed" knowledge, while **ignoring the rest** (e.g. annotations, declarations, etc.)
- $F(G)$ is an OWL 2 DL ontology, for G sat. OWL 2 DL restrictions.

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <http://example.org/2014-osw-14/>.
_:y a owl:Ontology .
_:x rdfs:subClassOf :Parent ;
    a owl:Restriction ;
    owl:onProperty :hasChild ;
    owl:someValuesFrom owl:Thing .
:John :hasChild :Mary .
:John a owl:NamedIndividual .
:Mary a owl:NamedIndividual .
:hasChild a owl:ObjectProperty .
```

```
@prefix : <http://www.example.org/2014-osw-14/> .
:John a :Parent .
:John rdfs:label "john"@en .
```

The following entailment holds:

$$F(G_3) \models_{OWL2-DL} F(G_4)$$

(For the sake of brevity, $F(\bullet)$ is often omitted whenever G is a serialization of an OWL-DL ontology $F(G)$)

OWL 2 Correspondence Theorem (CT)

- direct and RDF-based semantics for OWL are different (i.e. there exist entailments valid for one semantic and not for the other one)
- CT says that **OWL RDF semantic can express anything that OWL DL semantics can**

OWL 2 Correspondence Theorem – simplified version

For any two RDF graphs G_1 and G_2, there exist two RDF graphs G'_1 and G'_2, s.t. $F(G_1) \models\!\!\dashv_{OWL-DL} F(G'_1)$ and $F(G_2) \models\!\!\dashv_{OWL2-DL} F(G'_2)$, and

$$F(G'_1) \models_{OWL2-DL} F(G'_2) \quad \text{implies} \quad G'_1 \models_{OWL2-RDF} G'_2,$$

where $F(G)$ is an OWL-DL ontology corresponding to the RDF graph G.

- For example $G_1 \not\models_{OWL2-DL} G_2$, while $G_3 \not\models_{OWL2-RDF} G_4$
- Removing last triple (label) from G_4, we get G'_4, s.t. $F(G_4) \models\!\!\dashv_{OWL-DL} F(G'_4)$ and $G_4 \models_{OWL-RDF} G'_4$