Semantic Web

— to Artificial Intelligence

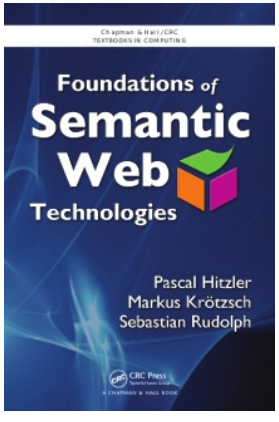
Yue Ma

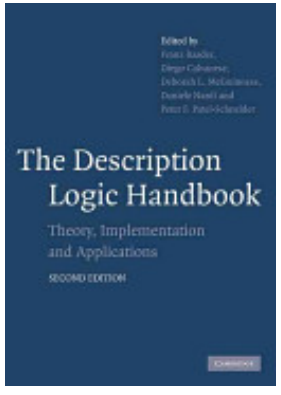
Laboratoire de Recherche en Informatique (LRI)
Université Paris Sud
ma@lri.fr

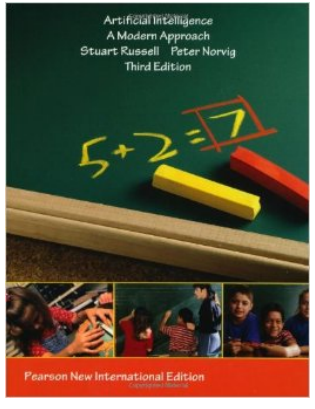
General Information

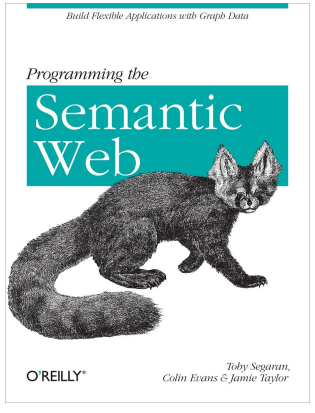
- This lecture is inspired by the following courses :
  - Knowledge Representation for the Semantic Web
    Pascal Hitzler, Wright State University, US
  - Foundations of Semantic Web Technologies
    Sebastian Rudolph, TU Dresden, Germany
  - Knowledge Graph Analysis
    Jens Lehmann, University of Bonn, Germany
  - Introduction à l'intelligence artificielle
    Laurent Simon, Polytech Paris Sud

- Reference books :









General Information

- Course materials : `www.lri.fr/~ma/M2DK`
- Organizational matters :
  - Initial plans :
    - Part 1 : Semantic Web Standards (RDF, RDFS, OWL, SPARQL)
    - Part 2 : Semantics and Reasoning for Semantic Web
    - Projects : Querying Semantic Data, Ontology Reasoning
  - Grading : average(Projects, Exam)

Outline

OWL ontology from a formal perspective : Description Logics (DLs)

- Components of an ontology
- A basic DL : synatx 句法
- A basic DL : semantics 语义
- Ontology reasoning tasks
- History of Description Logics

Inference algorithms for a simple formal language

- A special sub-language of OWL : Horn rules
- Forward Chaining
- Backforward Chaining

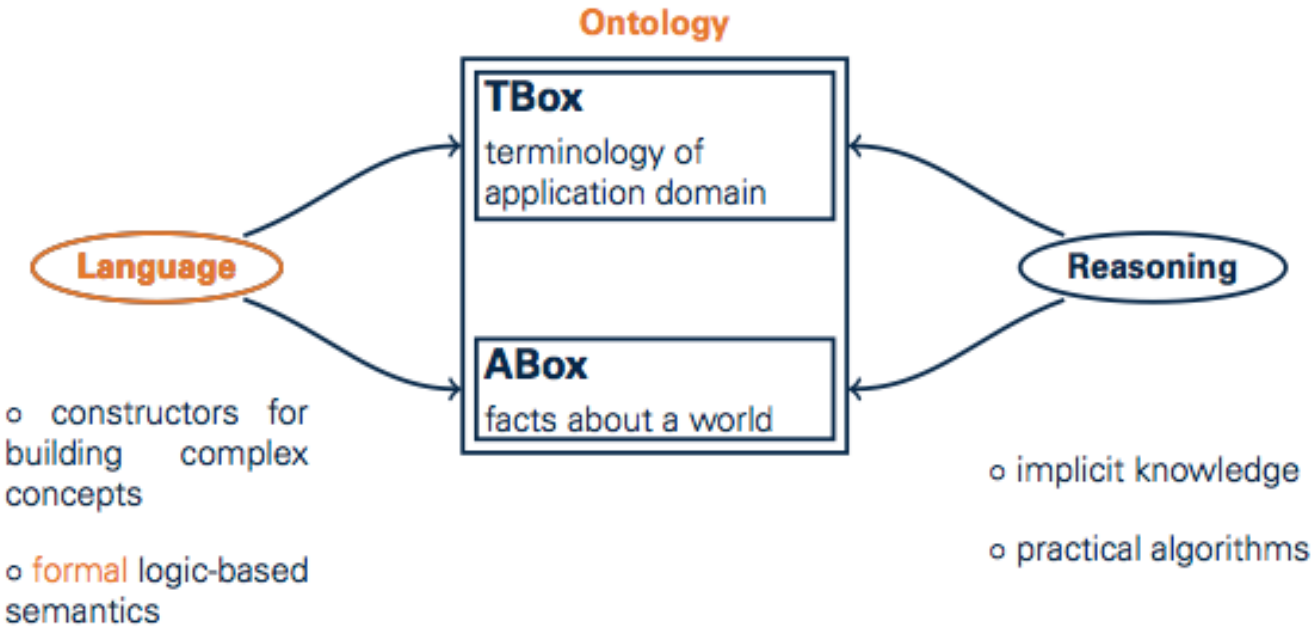
OWL ontology from a formal perspective : Description Logics (DLs)

Expressing knowledge

- Heart disease is a kind of mediastinum disorder.
- Heart disease happens in heart structure.
- Smoking or alcoholism is bad for health.
- Parents are people who have at least one child.
- Each student can register at most two study programs each semester.
- Man and woman are disjoint.

How we can express these pieces of knowledge in a way computers can understand ?

Structure of an ontology



Correspondence between Ontology and NL

Semantic Web Ontology	$\longleftrightarrow$	Natural Language
names in N_C, N_R		single words
complex concepts		phrases
axioms		sentences (statements) expressing knowledge [1]

A step further towards ontologies...

1. like "There is a wash basin in either a kitchen or a bathroom", but not emotional sentences “What a nice day !”

DL ontology

Formalize the terminology (names) of the application domain :

- define important notions of the domain (classes, relations, objects)
- constrain the interpretations of these notions
- deduce consequences : subclass, instance relationships

Example :

- classes (concepts) : Person, Teacher, Course, Student, HeartDisease, HeartStructure, ...
- relations (roles) : gives, attends, likes, findingSite, ...
- objects (individuals) : CourseSemanticweb, John, Jim, ...
- constraints/knowledge :

  every course is given by a teacher,

  every student must attend at least one course

  heart disease is a kind of mediastinum disorder

  heart disease happens in heart structure

  happy parents are those whose children are happy

  John is a happy boy

  John's father is Jim

A basic Description Logic $\mathcal{ALC}$: syntax

- concept names are also called atomic concepts
- all other concepts are called complex
- N_C : a set of concept names
- N_R a set of role name

Example.
Let N_C=$\{Female, Parent, Student\}$ and $N_R = \{hasChild\}$.

A basic Description Logic $\mathcal{ALC}$: syntax

- concept names are also called atomic concepts
- all other concepts are called complex
- N_C : a set of concept names
- N_R a set of role name

Example.
Let N_C=$\{Female, Parent, Student\}$ and $N_R = \{hasChild\}$.

We may express the concept "female who has a student child" :

A basic Description Logic $\mathcal{ALC}$: syntax

- concept names are also called atomic concepts
- all other concepts are called complex
- N_C : a set of concept names
- N_R a set of role name

Example.
Let N_C=$\{Female, Parent, Student\}$ and $N_R = \{hasChild\}$.

We may express the concept "female who has a student child" :

A basic Description Logic $\mathcal{ALC}$: syntax

- concept names are also called atomic concepts
- all other concepts are called complex
- N_C : a set of concept names
- N_R a set of role name

Example.
Let N_C=$\{Female, Parent, Student\}$ and $N_R = \{hasChild\}$.

We may express the concept "female who has a student child" :

$$Female \sqcap \exists hasChild.Student$$

How about "female whose children are students" :

A basic Description Logic $\mathcal{ALC}$: syntax

- concept names are also called atomic concepts
- all other concepts are called complex
- N_C : a set of concept names
- N_R a set of role name

Example.
Let N_C=$\{Female, Parent, Student\}$ and $N_R = \{hasChild\}$.

We may express the concept "female who has a student child" :

$$Female \sqcap \exists hasChild.Student$$

How about "female whose children are students" :

$$Female \sqcap \forall hasChild.Student$$

Syntax of $\mathcal{ALC}$: constructing complex concepts

Let N_C and N_R two disjoint sets of concept names and role names, respectively. $\mathcal{ALC}$ concepts are defined by induction :

- if $A \in N_C$,then A is an $\mathcal{ALC}$ concept
- if C, D are $\mathcal{ALC}$ concepts and $r \in N_R$, then the following are $\mathcal{ALC}$ concepts :
  - $C \sqcap D$ (conjunction)
  - $C \sqcup D$ (disjunction)
  - $\neg C$ (negation)
  - $\exists r.C$ (existential restriction)
  - $\forall r.C$ (value restriction)
- Abbreviations :
  - $\top = A \sqcup \neg A$ (top)
  - $\bot = A \sqcap \neg A$ (bottom)

Example. $Female \sqcap \exists hasChild.Student$

For you : try to write down your (atom, complex) $\mathcal{ALC}$ concepts.

$Female \sqcap \forall hasChild.(Student \sqcap \exists hasScore.GoodScore)$

Syntax of $\mathcal{ALC}$: constructing complex concepts

Let N_C and N_R two disjoint sets of concept names and role names, respectively. $\mathcal{ALC}$ concepts are defined by induction :

- if $A \in N_C$,then A is an $\mathcal{ALC}$ concept
- if C, D are $\mathcal{ALC}$ concepts and $r \in N_R$, then the following are $\mathcal{ALC}$ concepts :
  - $C \sqcap D$ (conjunction)
  - $C \sqcup D$ (disjunction)
  - $\neg C$ (negation)
  - $\exists r.C$ (existential restriction)
  - $\forall r.C$ (value restriction)
- Abbreviations :
  - $\top = A \sqcup \neg A$ (top)
  - $\bot = A \sqcap \neg A$ (bottom)

Example. $Female \sqcap \exists hasChild.Student$

For you : try to write down your (atom, complex) $\mathcal{ALC}$ concepts.

$Female \sqcap \forall hasChild.(Student \sqcap \exists hasScore.GoodScore)$

Syntax of $\mathcal{ALC}$: constructing complex concepts

Let N_C and N_R two disjoint sets of concept names and role names, respectively. $\mathcal{ALC}$ concepts are defined by induction :

- if $A \in N_C$,then A is an $\mathcal{ALC}$ concept
- if C, D are $\mathcal{ALC}$ concepts and $r \in N_R$, then the following are $\mathcal{ALC}$ concepts :
  - $C \sqcap D$ (conjunction)
  - $C \sqcup D$ (disjunction)
  - $\neg C$ (negation)
  - $\exists r.C$ (existential restriction)
  - $\forall r.C$ (value restriction)
- Abbreviations :
  - $\top = A \sqcup \neg A$ (top)
  - $\bot = A \sqcap \neg A$ (bottom)

Example. $Female \sqcap \exists hasChild.Student$

For you : try to write down your (atom, complex) $\mathcal{ALC}$ concepts.

$Female \sqcap \forall hasChild.(Student \sqcap \exists hasScore.GoodScore)$

Syntax of $\mathcal{ALC}$: constructing complex concepts

Let N_C and N_R two disjoint sets of concept names and role names, respectively. $\mathcal{ALC}$ concepts are defined by induction :

- if $A \in N_C$,then A is an $\mathcal{ALC}$ concept
- if C, D are $\mathcal{ALC}$ concepts and $r \in N_R$, then the following are $\mathcal{ALC}$ concepts :
  - $C \sqcap D$ (conjunction)
  - $C \sqcup D$ (disjunction)
  - $\neg C$ (negation)
  - $\exists r.C$ (existential restriction)
  - $\forall r.C$ (value restriction)
- Abbreviations :
  - $\top = A \sqcup \neg A$ (top)
  - $\bot = A \sqcap \neg A$ (bottom)

Example. $Female \sqcap \exists hasChild.Student$

For you : try to write down your (atom, complex) $\mathcal{ALC}$ concepts.

$Female \sqcap \forall hasChild.(Student \sqcap \exists hasScore.GoodScore)$

Syntax of $\mathcal{ALC}$: constructing complex concepts

Let N_C and N_R two disjoint sets of concept names and role names, respectively. $\mathcal{ALC}$ concepts are defined by induction :

- if $A \in N_C$,then A is an $\mathcal{ALC}$ concept
- if C, D are $\mathcal{ALC}$ concepts and $r \in N_R$, then the following are $\mathcal{ALC}$ concepts :
  - $C \sqcap D$ (conjunction)
  - $C \sqcup D$ (disjunction)
  - $\neg C$ (negation)
  - $\exists r.C$ (existential restriction)
  - $\forall r.C$ (value restriction)
- Abbreviations :
  - $\top = A \sqcup \neg A$ (top)
  - $\bot = A \sqcap \neg A$ (bottom)

Example. $\mathit{Female} \sqcap \exists \mathit{hasChild}.\mathit{Student}$

For you : try to write down your (atom, complex) $\mathcal{ALC}$ concepts.

$\mathit{Female} \sqcap \forall \mathit{hasChild}.(\mathit{Student} \sqcap \exists \mathit{hasScore}.\mathit{GoodScore})$

Syntax of $\mathcal{ALC}$: constructing complex concepts

Let N_C and N_R two disjoint sets of concept names and role names, respectively. $\mathcal{ALC}$ concepts are defined by induction :

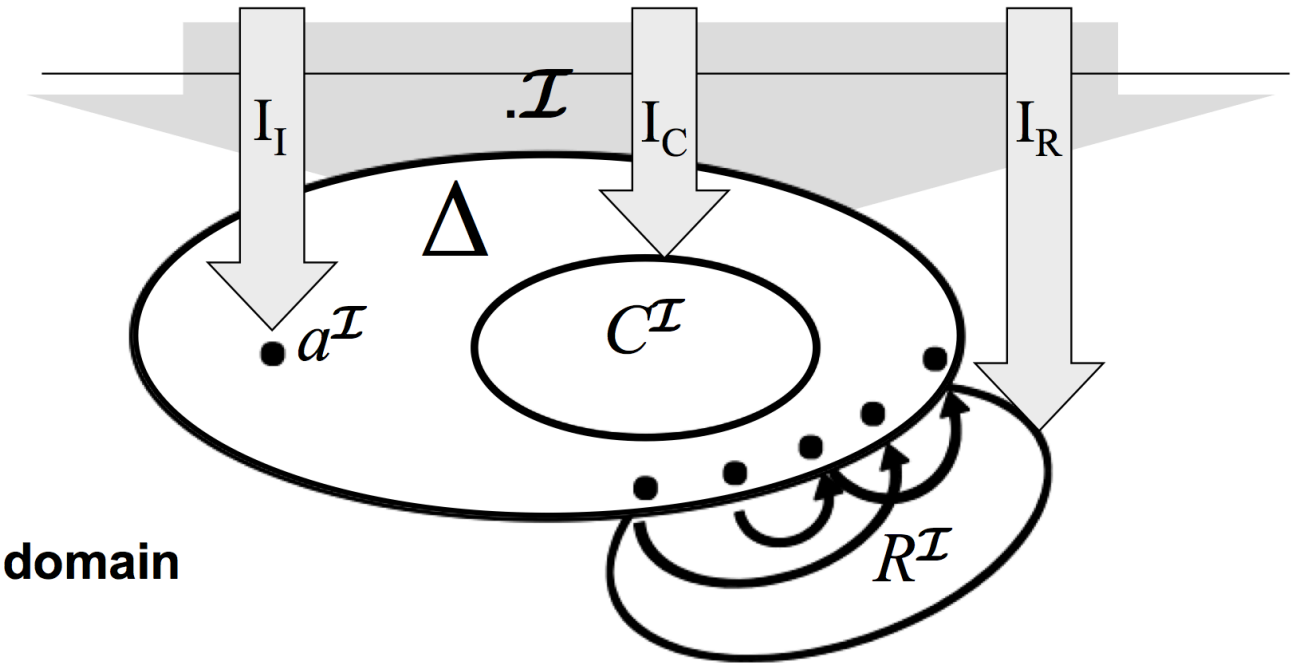
- if $A \in N_C$,then A is an $\mathcal{ALC}$ concept
- if C, D are $\mathcal{ALC}$ concepts and $r \in N_R$, then the following are $\mathcal{ALC}$ concepts :
  - $C \sqcap D$ (conjunction)
  - $C \sqcup D$ (disjunction)
  - $\neg C$ (negation)
  - $\exists r.C$ (existential restriction)
  - $\forall r.C$ (value restriction)
- Abbreviations :
  - $\top = A \sqcup \neg A$ (top)
  - $\bot = A \sqcap \neg A$ (bottom)

Example. $\mathit{Female} \sqcap \exists \mathit{hasChild}.\mathit{Student}$
For you : try to write down your (atom, complex) $\mathcal{ALC}$ concepts.
$\mathit{Female} \sqcap \forall \mathit{hasChild}.(\mathit{Student} \sqcap \exists \mathit{hasScore}.\mathit{GoodScore})$

Semantics of Description Logics

- **model-theoretic semantics**
- **starts with interpretations**
- **an interpretation $\mathcal{I}$ maps**

  individual names, class names and property names...



...into a domain

Semantics of $\mathcal{ALC}$

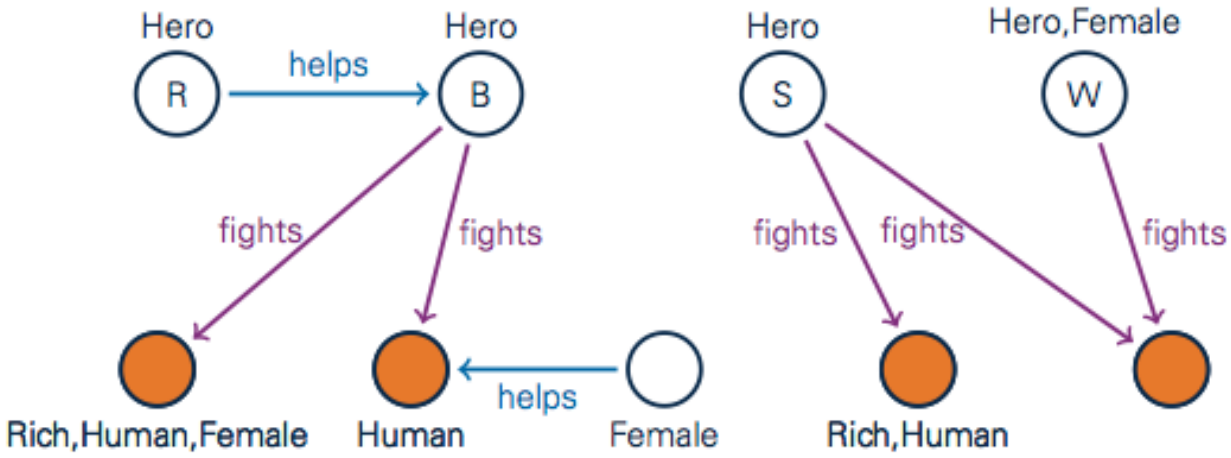
An interpretation $I = (\Delta^I, \cdot^I)$ consists of :

- a non-empty domain Δ^I , and
- an extension mapping $\cdot^I$ (also called interpretation function) :
  - $A^I \subseteq \Delta^I$ for all $A \in N_C$ (concepts interpreted as sets)
  - $r^I \subseteq \Delta^I \times \Delta^I$ for all $r \in N_R$ (roles interpreted as binary relations)

The extension mapping is extended to complex concepts :

$$
\begin{aligned}
(C \sqcap D)^I &:= C^I \cap D^I \\
(C \sqcup D)^I &:= C^I \cup D^I \\
(\neg C)^I &:= \Delta^I \setminus C^I \\
(\exists r.C)^I &:= \{d \in \Delta^I \mid \text{there is } e \in \Delta^I \text{ with } (d,e) \in r^I \text{ and } e \in C^I\} \\
(\forall r.C)^I &:= \{d \in \Delta^I \mid \text{for all } e \in \Delta^I \text{ with } (d,e) \in r^I, \text{ it holds } e \in C^I\}
\end{aligned}
$$

Interpretation Example



$$(Hero \sqcap \exists fights.Human)^I = \{B, S\}$$

$$(Hero \sqcap \forall fights.(Rich \sqcup \neg Human))^I = \{R, S, W\}$$

$$(\forall helps.Human)^I = \Delta^I \setminus \{R\}$$

Interpretation Example

Can you give an interpretation for the following concept ?

$$\mathit{Female} \sqcap \exists \mathit{hasChild}.\mathit{Student}$$

To define what is an ontology, we need the following definitions :

- TBox and ABox axioms
  - TBox : terminology box containing concept definitions and/or GCIs 术语
  - ABox : assertion box, containing individual information.

Defining a Concept (aka. Concept Definitions)

Definitions.

- A concept definition is of the form $A = C$ where
  - A is a concept name.
  - C is a concept description.
- An interpretation I satisfies, also called a model of, the concept definition $A = C$ if $A^I = C^I$.

Examples.

$$\boxed{\mathit{Heroine} = \mathit{Hero} \sqcap \mathit{Female}}$$

$$\boxed{\mathit{Student} = \mathit{People} \sqcap \exists \mathit{registes}.(\mathit{School} \sqcup \mathit{University})}$$

Defining a Concept (aka. Concept Definitions)

Definitions.

- A concept definition is of the form $A = C$ where
  - A is a concept name.
  - C is a concept description.
- An interpretation I satisfies, also called a model of, the concept definition $A = C$ if $A^I = C^I$.

Examples.

$\boxed{\mathit{Heroine} = \mathit{Hero} \sqcap \mathit{Female}}$

Does the following interpretation I satisfy this concept definition ?

$$\Delta^I = \{a, b\}$$
$$\mathit{Heroine}^I = \{a\}$$
$$\mathit{Hero}^I = \{a, b\}$$
$$\mathit{Female}^I = \{b\}$$

If yes, why ?

If no, modify I to make it satisfy the concept definition.

Defining a Concept (aka. Concept Definitions)

Definitions.

- A concept definition is of the form $A = C$ where
  - A is a concept name.
  - C is a concept description.
- An interpretation I satisfies, also called a model of, the concept definition $A = C$ if $A^I = C^I$.

Examples.

$Heroine = Hero \sqcap Female$

$\boxed{Student = People \sqcap \exists registes.(School \sqcup University)}$

For you : give a model of this concept definition

GCIs (General Concept Inclusions) and TBoxes

Definitions.

- A general concept inclusion (GCI) is of the form $C \sqsubseteq D$, where C, D are concepts.
- A TBox $\mathcal{T}$ is a finite set of GCIs and/or concept definitions.
- An interpretation I satisfies the GCI $C \sqsubseteq D$ iff $C^I \subseteq D^I$. I is a model of a TBox T iff it satisfies all GCIs in T.
- Two TBoxes are equivalent if they have the same models.

Examples.
$\mathit{Hero} \sqcap \mathit{Villain} \sqsubseteq \bot$: two concepts are disjoint
$\mathit{Heroine} = \mathit{Hero} \sqcap \mathit{Female}$
$\mathit{Kitchen} \sqcup \mathit{Bathroom} \sqsubseteq \exists \mathit{hasWashbasin}.\top$
$\mathit{Cold} \sqcap \exists \mathit{causedBy}.\mathit{Virus} \sqsubseteq \mathit{Disease}$

GCIs (General Concept Inclusions) and TBoxes

Definitions.

- A general concept inclusion (GCI) is of the form $C \sqsubseteq D$, where C, D are concepts.
- A TBox $\mathcal{T}$ is a finite set of GCIs and/or concept definitions.
- An interpretation I satisfies the GCI $C \sqsubseteq D$ iff $C^I \subseteq D^I$. I is a model of a TBox T iff it satisfies all GCIs in T.
- Two TBoxes are equivalent if they have the same models.

Examples.
$\mathit{Hero} \sqcap \mathit{Villain} \sqsubseteq \bot$: two concepts are disjoint
$\mathit{Heroine} = \mathit{Hero} \sqcap \mathit{Female}$
$\mathit{Kitchen} \sqcup \mathit{Bathroom} \sqsubseteq \exists \mathit{hasWashbasin}.\top$
$\mathit{Cold} \sqcap \exists \mathit{causedBy}.\mathit{Virus} \sqsubseteq \mathit{Disease}$

GCIs (General Concept Inclusions) and TBoxes

Definitions.

- A general concept inclusion (GCI) is of the form $C \sqsubseteq D$, where C, D are concepts.
- A TBox $\mathcal{T}$ is a finite set of GCIs and/or concept definitions.
- An interpretation I satisfies the GCI $C \sqsubseteq D$ iff $C^I \subseteq D^I$. I is a model of a TBox T iff it satisfies all GCIs in T.
- Two TBoxes are equivalent if they have the same models.

Examples.
$\mathit{Hero} \sqcap \mathit{Villain} \sqsubseteq \bot$: two concepts are disjoint
$\mathit{Heroine} = \mathit{Hero} \sqcap \mathit{Female}$
$\mathit{Kitchen} \sqcup \mathit{Bathroom} \sqsubseteq \exists \mathit{hasWashbasin}.\top$
$\mathit{Cold} \sqcap \exists \mathit{causedBy}.\mathit{Virus} \sqsubseteq \mathit{Disease}$

GCIs (General Concept Inclusions) and TBoxes

Definitions.

- A general concept inclusion (GCI) is of the form $C \sqsubseteq D$, where C, D are concepts.
- A TBox $\mathcal{T}$ is a finite set of GCIs and/or concept definitions.
- An interpretation I satisfies the GCI $C \sqsubseteq D$ iff $C^I \subseteq D^I$. I is a model of a TBox T iff it satisfies all GCIs in T.
- Two TBoxes are equivalent if they have the same models.

Examples.
$\mathit{Hero} \sqcap \mathit{Villain} \sqsubseteq \bot$: two concepts are disjoint
$\mathit{Heroine} = \mathit{Hero} \sqcap \mathit{Female}$
$\mathit{Kitchen} \sqcup \mathit{Bathroom} \sqsubseteq \exists \mathit{hasWashbasin}.\top$
$\mathit{Cold} \sqcap \exists \mathit{causedBy}.\mathit{Virus} \sqsubseteq \mathit{Disease}$
For you : give a model for these GCIs and definitions. How many models does each of them have ?

GCIs (General Concept Inclusions) and TBoxes

Definitions.

- A general concept inclusion (GCI) is of the form $C \sqsubseteq D$, where C, D are concepts.
- A TBox $\mathcal{T}$ is a finite set of GCIs and/or concept definitions.
- An interpretation I satisfies the GCI $C \sqsubseteq D$ iff $C^I \subseteq D^I$. I is a model of a TBox T iff it satisfies all GCIs in T.
- Two TBoxes are equivalent if they have the same models.

Examples.
$\mathit{Hero} \sqcap \mathit{Villain} \sqsubseteq \bot$: two concepts are disjoint
$\mathit{Heroine} = \mathit{Hero} \sqcap \mathit{Female}$
$\mathit{Kitchen} \sqcup \mathit{Bathroom} \sqsubseteq \exists \mathit{hasWashbasin}.\top$
$\mathit{Cold} \sqcap \exists \mathit{causedBy}.\mathit{Virus} \sqsubseteq \mathit{Disease}$
For you : give a model for these GCIs and definitions. How many models does each of them have ?

Assertions and ABoxes

Definitions.

- An assertion is of the form $C(a)$ (concept assertion) or $r(a, b)$ (role assertion) where C is a concept, r a role, and a, b are individual names from a set N_I (disjoint with N_C, N_R)
- An ABox $\mathcal{A}$ is a finite set of assertions
- An interpretation I is a model of the ABox $\mathcal{A}$ if it satisfies all its assertions :
  - $a^I \in C^I$ for all $C(a) \in \mathcal{A}$.
  - $(a^I, b^I) \in R^I$ for all $R(a, b) \in \mathcal{A}$.

Examples.
$Kitchen(room1)$: the room1 is a kitchen
$locatedIn(whitechair1, room1)$: the whitechair1 is in the room1
$\neg Student(Jean)$: Jean is not a student

Assertions and ABoxes

Definitions.

- An assertion is of the form $C(a)$ (concept assertion) or $r(a, b)$ (role assertion) where C is a concept, r a role, and a, b are individual names from a set N_I (disjoint with N_C, N_R)
- An ABox $\mathcal{A}$ is a finite set of assertions
- An interpretation I is a model of the ABox $\mathcal{A}$ if it satisfies all its assertions :
  - $a^I \in C^I$ for all $C(a) \in \mathcal{A}$.
  - $(a^I, b^I) \in R^I$ for all $R(a, b) \in \mathcal{A}$.

Examples.
$Kitchen(room1)$: the room1 is a kitchen
$locatedIn(whitechair1, room1)$: the whitechair1 is in the room1
$\neg Student(Jean)$: Jean is not a student
For you : name some models for these ABox assertions. How many models does each of them have ?

Ontology

Definitions.

- An ontology $O = (\mathcal{T}, \mathcal{A})$ consists of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$.
- The interpretation I is a model of the ontology $O = (\mathcal{T}, \mathcal{A})$ iff it is a model of $\mathcal{T}$ and a model of $\mathcal{A}$.

Examples.
Many ontologies from http ://swoogle.umbc.edu and
http ://bioportal.bioontology.org

TBox or ABox axioms ?

```
:Mary rdf:type :Woman .
```

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

```
:John owl:differentFrom :Bill .
```

{John} ⊓ {Bill} ⊑ ⊥

```
:James owl:sameAs :Jim.
```

{John} ≡ {Jim}

```
:John  :hasAge "51"^^xsd:nonNegativeInteger .
```

```
[]  rdf:type                  owl:NegativePropertyAssertion ;
    owl:sourceIndividual      :Bill ;
    owl:assertionProperty     :hasWife ;
    owl:targetIndividual      :Mary .
```

¬hasWife(Bill,Mary)

```
[]  rdf:type                  owl:NegativePropertyAssertion ;
    owl:sourceIndividual      :Jack ;
    owl:assertionProperty     :hasAge ;
    owl:targetValue           53 .
```

TBox or ABox axioms ?

```
:Woman rdfs:subClassOf :Person .
```

```
:Person owl:equivalentClass :Human .
```

```
[]  rdf:type      owl:AllDisjointClasses ;
    owl:members   ( :Woman  :Man ) .
```

Woman $\sqcap$ **Man** $\sqsubseteq \bot$

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

```
:hasWife rdfs:domain :Man ;
         rdfs:range  :Woman .
```

TBox or ABox axioms ?

```
:Mother  owl:equivalentClass  [
  rdf:type              owl:Class ;
  owl:intersectionOf  ( :Woman :Parent )
] .
```

Mother ≡ Woman ⊓ Parent

```
:Parent  owl:equivalentClass  [
  rdf:type      owl:Class ;
  owl:unionOf  ( :Mother :Father )
] .
```

Parent ≡ Mother ⊔ Father

```
:ChildlessPerson  owl:equivalentClass  [
  rdf:type              owl:Class ;
  owl:intersectionOf  ( :Person  [ owl:complementOf  :Parent ] )
] .
```

ChildlessPerson ≡ Person ⊓ ¬Parent

```
:Grandfather  rdfs:subClassOf  [
  rdf:type              owl:Class ;
  owl:intersectionOf  ( :Man  :Parent )
] .
```

TBox or ABox axioms ?

Person ⊓ ¬Parent (Jack)

```
:Jack  rdf:type  [
  rdf:type              owl:Class ;
  owl:intersectionOf    ( :Person
                          [ rdf:type           owl:Class ;
                            owl:complementOf   :Parent     ]
                        )
] .
```

TBox or ABox axioms ?

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

Parent ≡ ∃hasChild.Person

```
:Orphan  owl:equivalentClass  [
  rdf:type              owl:Restriction ;
  owl:onProperty        [ owl:inverseOf  :hasChild ] ;
  owl:allValuesFrom     :Dead
] .
```

Orphan ≡ ∀ hasChild⁻.Dead

TBox or ABox axioms ?

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

JohnsChildren ≡ ∃hasParent.{John}

```
:NarcisticPerson owl:equivalentClass  [
  rdf:type          owl:Restriction ;
  owl:onProperty    :loves ;
  owl:hasSelf       "true"^^xsd:boolean .
] .
```

NarcisticPerson ≡ ∃loves.Self

TBox or ABox axioms ?

≤4 hasChild.Parent (John)

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

```
:John  rdf:type  [
  rdf:type                          owl:Restriction ;
  owl:minQualifiedCardinality       "2"^^xsd:nonNegativeInteger ;
  owl:onProperty                    :hasChild ;
  owl:onClass                       :Parent
] .
```

≥2 hasChild.Parent (John)

```
:John  rdf:type  [
  rdf:type                          owl:Restriction ;
  owl:qualifiedCardinality          "3"^^xsd:nonNegativeInteger ;
  owl:onProperty                    :hasChild ;
  owl:onClass                       :Parent
] .
```

=3 hasChild.Parent (John)

TBox or ABox axioms ?

```
:John  rdf:type  [
  rdf:type          owl:Restriction ;
  owl:cardinality   "5"^^xsd:nonNegativeInteger ;
  owl:onProperty    :hasChild
] .
```

=5 hasChild.⊤ (John)

```
:MyBirthdayGuests  owl:equivalentClass  [
  rdf:type   owl:Class ;
  owl:oneOf  ( :Bill  :John  :Mary )
] .
```

MyBirthdayGuests ≡ {Bill, John, Mary}

TBox or ABox axioms ?

```
:hasParent owl:inverseOf :hasChild .

:Orphan  owl:equivalentClass  [
  rdf:type              owl:Restriction ;
  owl:onProperty        [ owl:complementOf   :hasChild ] ;
  owl:allValuesFrom     :Dead
] .

:hasSpouse  rdf:type  owl:SymmetricProperty .

:hasChild  rdf:type  owl:AsymmetricProperty .

:hasParent  owl:propertyDisjointWith  :hasSpouse .

:hasRelative  rdf:type  owl:ReflexiveProperty .

:parentOf  rdf:type  owl:IrreflexiveProperty .

:hasHusband  rdf:type  owl:FunctionalProperty .

:hasHusband  rdf:type  owl:InverseFunctionalProperty .

:hasAncestor  rdf:type  owl:TransitiveProperty .
```

$\mathbf{Orphan} \equiv \forall\, \mathbf{hasChild}^{-}.\mathbf{Dead}$

TBox or ABox axioms ?

```
:hasGrandparent owl:propertyChainAxiom ( :hasParent :hasParent ).
```

hasParent ∘ hasParent ⊑ hasGrandParent

```
:Person owl:hasKey ( :hasSSN ) .
```

In OWL 2 a collection of (data or object) properties can be assigned as a key to a class expression. This means that each named instance of the class expression is uniquely identified by the set of values which these properties attain in relation to the instance.

TBox or ABox axioms ?

```
:personAge  owl:equivalentClass
 [ rdf:type  rdfs:Datatype;
   owl:onDatatype  xsd:integer;
   owl:withRestrictions (
      [ xsd:minInclusive  "0"^^xsd:integer ]
      [ xsd:maxInclusive  "150"^^xsd:integer ]
   )
 ] .
```

Datatype *facets*

```
:majorAge  owl:equivalentClass
  [ rdf:type  rdfs:Datatype;
    owl:intersectionOf (
       :personAge
       [ rdf:type  rdfs:Datatype;
         owl:datatypeComplementOf  :minorAge ]
    )
```

Let us now define the interesting reasoning services that we can benefit from ontologies

Terminological Reasoning of an Ontology

Let $\mathcal{T}$ be a TBox. Terminological reasoning refers to deciding the following problems :

- Satisfiability : a concept C is satisfiable w.r.t. $\mathcal{T}$ if and only if there is a model I of $\mathcal{T}$ such that $C^I \neq \emptyset$.
- Subsumption : C is subsumed by D w.r.t. $\mathcal{T}$ if and only if $C^I \subseteq D^I$ for all models I of $\mathcal{T}$.
- Equivalence : C is equivalent to D w.r.t. $\mathcal{T}$ if and only if $C^I = D^I$ for all models I of $\mathcal{T}$.
- Entailment : An ontology O entails $C \sqsubseteq D$, written $O \models C \sqsubseteq D$ ($O \models C(a)$, or $O \models R(a, b)$) if and only if $C^I \subseteq D^I$ (resp. $a^I \in C^I$, $(a^I, b^I) \in R^I$) for all models I of O.

Examples (next page)

Ontology : Reasoning Problems and Services

Examples.

- $A \sqcap \neg A$ is unsatisfiable
- $\forall r.A \sqcap \forall r.\neg A$ is satisfiable
- $\forall r.A \sqcap \exists r.\neg A$ is unsatisfiable Improve it is empty
- $\exists r.(A \sqcap B)$ is subsumbed by $\exists r.A$
- $\{A \sqsubseteq B, B \sqsubseteq C\} \models A \sqsubseteq C$ (entailment)
- $\{A \sqsubseteq \exists r.B, B \sqsubseteq C\} \models A \sqsubseteq \exists r.C$ (entailment)

Assertional Reasoning of an Ontology

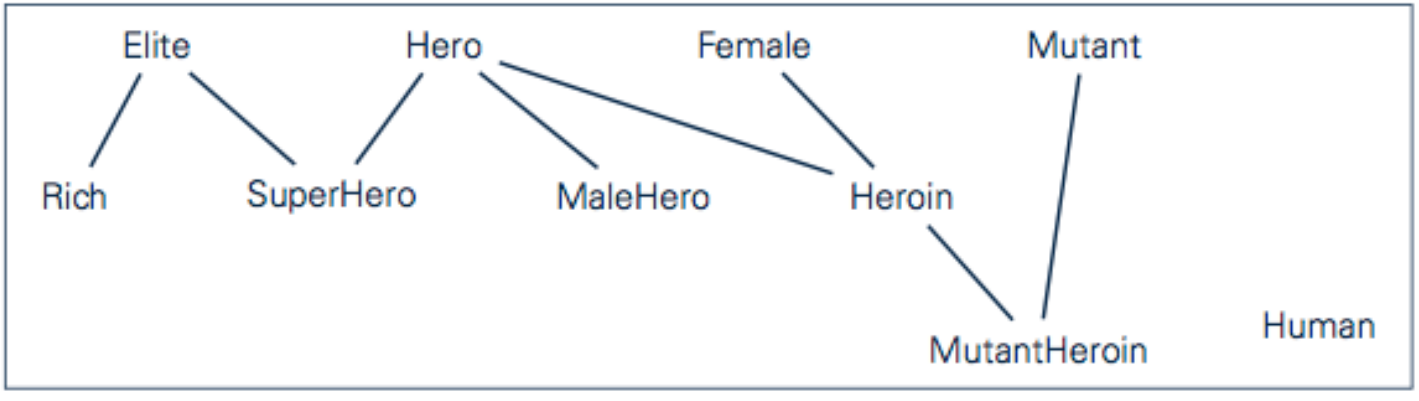
Let $O = (\mathcal{T}, \mathcal{A})$ be an ontology with TBox $\mathcal{T}$ and ABox $\mathcal{A}$.
Assertional reasoning refers to deciding the following problems :

- Consistency : O is consistent iff O has a model.
- Instance : a is an instance of a concept C w.r.t O iff $a^I \in C^I$ for all models I of O

TBox Classification

Computing the subsumption relations between all concept names in $\mathcal{T}$:

$$
\begin{aligned}
\text{Heroine} &\equiv \text{Hero} \sqcap \text{Female} \\
\text{MaleHero} &\equiv \text{Hero} \sqcap \neg\text{Female} \\
\text{MutantHeroine} &\equiv \text{Heroine} \sqcap \text{Mutant} \\
\text{Elite} &\equiv \text{Rich} \sqcup \neg\text{Human} \\
\text{Superhero} &\equiv \text{Hero} \sqcap \text{Elite}
\end{aligned}
$$



Prehistory of Description Logics

a class of logic-based knowledge representation formalisms for representing terminological knowledge

early approaches for representing terminological knowledge

- semantic networks (Quillian, 1968)
- frames (Minsky, 1975)

problems with missing semantics led to

- structured inheritance networks
- the first DL system KL-ONE

History of Description Logics

- Phase 1

  implementation of systems based on incomplete structural subsumption algorithms

- Phase 2
  - tableau-based algorithms and complexity results
  - first tableau-based systems (Kris, Crack)
  - first formal study of optimization methods
- Phase 3
  - tableau-based algorithms for very expressive DLs
  - highly-optimized tableau-based systems (FaCT, Racer)
  - relationship to modal logic and FOL

History of Description Logics

- Phase 4
  - Web Ontology Language (OWL) based on DL
  - industrial-strength reasoners and ontology editors
  - light-weight (tractable) DLs
- Phase 5
  - non-standard reasoning
  - ontology management
  - semantic extensions

These complex algorithms are out of the scope of this course.

History of Description Logics

- Phase 4
  - Web Ontology Language (OWL) based on DL
  - industrial-strength reasoners and ontology editors
  - light-weight (tractable) DLs
- Phase 5
  - non-standard reasoning
  - ontology management
  - semantic extensions

These complex algorithms are out of the scope of this course.

Summary

- Introduction to DLs
- Decidability, complexity, reasoning algorithms (following lectures)

Outline

Inference Algorithms

- We have seen the constructors :
  - rdfs :SubclassOf, rdfs :SubPropertyOf, rdfs :domain, rdfs :range, owl :intersectionOf, owl :complementOf, owl :unionOf, ...
- We have seen some reasoning examples over these operators, e.g.
  - rdfs :domain, rdfs :range (c.f. cours2.pdf — slides "RDFS Vocabulary vs. implicite information")
  - the inconsistency (c.f. cours2.pdf on OWL)
  - the incoherency (c.f. cours2.pdf on OWL)

How to design algorithms to do the reasoning automatically ?

Inference Algorithms

- We have seen the constructors :
  - rdfs :SubclassOf, rdfs :SubPropertyOf, rdfs :domain, rdfs :range, owl :intersectionOf, owl :complementOf, owl :unionOf, ...
- We have seen some reasoning examples over these operators, e.g.
  - rdfs :domain, rdfs :range (c.f. cours2.pdf — slides "RDFS Vocabulary vs. implicite information")
  - the inconsistency (c.f. cours2.pdf on OWL)
  - the incoherency (c.f. cours2.pdf on OWL)

How to design algorithms to do the reasoning automatically ?

Inference Algorithms

- We have seen the constructors :
  - rdfs :SubclassOf, rdfs :SubPropertyOf, rdfs :domain, rdfs :range, owl :intersectionOf, owl :complementOf, owl :unionOf, ...
- We have seen some reasoning examples over these operators, e.g.
  - rdfs :domain, rdfs :range (c.f. cours2.pdf — slides "RDFS Vocabulary vs. implicite information")
  - the inconsistency (c.f. cours2.pdf on OWL)
  - the incoherency (c.f. cours2.pdf on OWL)

How to design algorithms to do the reasoning automatically ?

Inference Algorithms

- We have seen the constructors :
  - rdfs :SubclassOf, rdfs :SubPropertyOf, rdfs :domain, rdfs :range, owl :intersectionOf, owl :complementOf, owl :unionOf, ...
- We have seen some reasoning examples over these operators, e.g.
  - rdfs :domain, rdfs :range (c.f. cours2.pdf — slides “RDFS Vocabulary vs. implicite information”)
  - the inconsistency (c.f. cours2.pdf on OWL)
  - the incoherency (c.f. cours2.pdf on OWL)

How to design algorithms to do the reasoning automatically ?

Inference Algorithms

- We have seen the constructors :
  - rdfs :SubclassOf, rdfs :SubPropertyOf, rdfs :domain, rdfs :range, owl :intersectionOf, owl :complementOf, owl :unionOf, ...
- We have seen some reasoning examples over these operators, e.g.
  - rdfs :domain, rdfs :range (c.f. cours2.pdf — slides “RDFS Vocabulary vs. implicite information”)
  - the inconsistency (c.f. cours2.pdf on OWL)
  - the incoherency (c.f. cours2.pdf on OWL)

How to design algorithms to do the reasoning automatically ?

Inference Algorithms

- We have seen the constructors :
  - rdfs :SubclassOf, rdfs :SubPropertyOf, rdfs :domain, rdfs :range, owl :intersectionOf, owl :complementOf, owl :unionOf, ...
- We have seen some reasoning examples over these operators, e.g.
  - rdfs :domain, rdfs :range (c.f. cours2.pdf — slides "RDFS Vocabulary vs. implicite information")
  - the inconsistency (c.f. cours2.pdf on OWL)
  - the incoherency (c.f. cours2.pdf on OWL)

How to design algorithms to do the reasoning automatically ?

Inference Algorithms

- We have seen the constructors :
  - rdfs :SubclassOf, rdfs :SubPropertyOf, rdfs :domain, rdfs :range, owl :intersectionOf, owl :complementOf, owl :unionOf, ...
- We have seen some reasoning examples over these operators, e.g.
  - rdfs :domain, rdfs :range (c.f. cours2.pdf — slides "RDFS Vocabulary vs. implicite information")
  - the inconsistency (c.f. cours2.pdf on OWL)
  - the incoherency (c.f. cours2.pdf on OWL)

How to design algorithms to do the reasoning automatically ?

Inference Algorithms

- We have seen the constructors :
  - rdfs :SubclassOf, rdfs :SubPropertyOf, rdfs :domain, rdfs :range, owl :intersectionOf, owl :complementOf, owl :unionOf, ...
- We have seen some reasoning examples over these operators, e.g.
  - rdfs :domain, rdfs :range (c.f. cours2.pdf — slides "RDFS Vocabulary vs. implicite information")
  - the inconsistency (c.f. cours2.pdf on OWL)
  - the incoherency (c.f. cours2.pdf on OWL)

How to design algorithms to do the reasoning automatically ?

A Simple Case

ONLY rdfs :SubclassOf ($\rightarrow$) and owl :intersectionOf ($\wedge$) are used in an ontology.

Horn formula

A horn formula is of the form :

$$l_1 \wedge l_2 \wedge l_3 \wedge ... \wedge l_n \rightarrow l$$

We call l_i and l are (positive) literals.

Reasoning problem

Given an Horn ontology O and a literal l_0. The reasoning task is to decide if l_0 can be followed from O ? If yes, we denote $O \models l_0$.

(See the questions of TD1 for an example.)

A Simple Case

ONLY rdfs :SubclassOf ($\rightarrow$) and owl :intersectionOf ($\wedge$) are used in an ontology.

Horn formula

A horn formula is of the form :

$$l_1 \wedge l_2 \wedge l_3 \wedge ... \wedge l_n \rightarrow l$$

We call l_i and l are (positive) literals.

Reasoning problem

Given an Horn ontology O and a literal l_0. The reasoning task is to decide if l_0 can be followed from O ? If yes, we denote $O \models l_0$.

(See the questions of TD1 for an example.)

A Simple Case

ONLY rdfs :SubclassOf ($\rightarrow$) and owl :intersectionOf ($\wedge$) are used in an ontology.

Horn formula

A horn formula is of the form :

$$l_1 \wedge l_2 \wedge l_3 \wedge ... \wedge l_n \rightarrow l$$

We call l_i and l are (positive) literals.

Reasoning problem

Given an Horn ontology O and a literal l_0. The reasoning task is to decide if l_0 can be followed from O ? If yes, we denote $O \models l_0$.

(See the questions of TD1 for an example.)

Reasoning algorithm : chaining from two sides

Two chaining algorithms

Forward Chaining

Principles : starting from a base of initial facts, deduce new facts according to rules in the base

- **Releasable rule** : all of its conditions are satisfiable
- **New facts** : the conclusions of releasable rules until saturation or the queried goal is achieved

Backward Chaining

Principles : starting from the goal, replace the goal by sub-goals.
Sub-goals : the conditions of a rule whose conclusion is a goal until that all the sub-goals are either facts or there is not sub-goal can be replaceable (not a conclusion of any rules)

Reasoning algorithm : chaining from two sides

Two chaining algorithms

Forward Chaining

Principles : starting from a base of initial facts, deduce new facts according to rules in the base

- **Releasable rule** : all of its conditions are satisfiable
- **New facts** : the conclusions of releasable rules until saturation or the queried goal is achieved

Backward Chaining

Principles : starting from the goal, replace the goal by sub-goals.
Sub-goals : the conditions of a rule whose conclusion is a goal until that all the sub-goals are either facts or there is not sub-goal can be replaceable (not a conclusion of any rules)

Forward Chaining Algorithm

Iterative Evaluation of Rules over a base of facts

FB : Fact base ; RB : Rule base
$ForwardChaining(FB_{init}, RB)$ computes the smallest fix point of the operator $Cons$ of immediate consequences from FB_{init}

- Cons is applied over FB if there exists a releasable rule over FB
- The result $Cons(FB)$ is obtained by adding FB the conclusion of the rule

Evaluation of a rule

Let FB be a fact base, and the rule R : **if** I_1 **and** ... **and** I_n **then** $I\!I$.

- $match(I, FB) = true$ iff $I \in FB$
- $eval(R, FB) = true$ iff for all i, $match(I_i, FB) = true$

Forward Chaining (ctd.)

```
1 : function ForwardChaining(FB_init, RB)
2 :     s(FB) ← FB_init
3 :     repeat
4 :         FB ← s(FB)
5 :         for all Rule R is not yet applied do
6 :             if eval(R, FB) then s(FB) ← s(FB) ∪ conclusions(R)
7 :             end if
8 :         end for
9 :     until s(FB) = FB
10 :     return s(FB)
11 : end function
```

Forward Chaining (ctd.)

Evaluation of a query Q by forward chaining

Q is satisfiable iff Q appears in the result of $\mathit{ForwardChaining}(FB_{init}, RB)$

Result : the algorithm $\mathit{ForwardChaining}(FB_{init}, RB)$ is polynomial w.r.t. the size of sFB_{init}, the size of RB, and the size of a rule.

Forward Chaining (example)

Tutorial 1, ex1-Q1

Optimisation of the forward chaining algorithm

Idea : don't repeat the operation of making correspondences between facts and conditions

- from one rule to another
- from one iteration to another

We proceed with the propagation of facts over the rules where the facts appear as condition.

- need indexing of atom-conditions regarding rules : $\mathit{InRuleConditions}(I) = \{R \mid I \text{ is an atom appearing in the condition of } R\}$
- for each rule, we keep memorizing the conditions satisfied in previous iterations.
  We update the rules by keeping only their conditions that are not yet satisfied while the propagation of a fact f :

$$\mathit{conditions}(R) \leftarrow \mathit{conditions}(R) - f$$

Optimisation of the forward chaining algorithm

Idea : don't repeat the operation of making correspondences between facts and conditions

- from one rule to another
- from one iteration to another

We proceed with the propagation of facts over the rules where the facts appear as condition.

- need indexing of atom-conditions regarding rules : $\mathit{InRuleConditions}(l) = \{R \mid l \text{ is an atom appearing in the condition of } R\}$
- for each rule, we keep memorizing the conditions satisfied in previous iterations.
  We update the rules by keeping only their conditions that are not yet satisfied while the propagation of a fact f :

$$\mathit{conditions}(R) \leftarrow \mathit{conditions}(R) - f$$

Algorithm of Forward Chaining with Propagation

function *ForwardChainingBis*(FB_{init}, RB)
    $FB \leftarrow FB_{init}$
    for all fact $F \in FB_{init}$ **do**
        $FB \leftarrow FB \cup$ *Propagate*(F, RB)
    end for
    return FB
end function

function *Propagate*(F, RB)
    $\Delta F \leftarrow \emptyset$
    for all rule $R \in$ *InRuleConditions*(F) **do**
        Delete from *conditions*(R) the atom I such that *match*(I, *condition*(R))
        if *conditions*(R) $= \emptyset$ **then**
            $RB \leftarrow RB \setminus \{R\}$
            $\Delta F \leftarrow \Delta F \cup$ *conclusions*(R)
        end if
    end for
    $FB \leftarrow \Delta F$
    for all fact $F \in \Delta F$ **do**
        $FB \leftarrow FB \cup$ *Propagate*(F, RB)
    end for
    return FB
end function

Algorithm of Forward Chaining with Propagation

function $ForwardChainingBis(FB_{init}, RB)$
    $FB \leftarrow FB_{init}$
    for all fact $F \in FB_{init}$ **do**
        $FB \leftarrow FB \cup Propagate(F, RB)$
    end for
    return FB
end function

function $Propagate(F, RB)$
    $\Delta F \leftarrow \emptyset$
    for all rule $R \in InRuleConditions(F)$ **do**
        Delete from $conditions(R)$ the atom I such that $match(I, condition(R))$
        if $conditions(R) = \emptyset$ **then**
            $RB \leftarrow RB \setminus \{R\}$
            $\Delta F \leftarrow \Delta F \cup conclusions(R)$
        end if
    end for
    $FB \leftarrow \Delta F$
    for all fact $F \in \Delta F$ **do**
        $FB \leftarrow FB \cup Propagate(F, RB)$
    end for
    return FB
end function

Example

Tutorial 1, ex1-Q2/3

Backforward Chaining

Principle

Algorithm of replacing a goal by sub-goals, by matching a current goal (the user's query at beginning) with the conclusion atoms of rules.

When to use it ?

To guide the questions proposed by users in the case of incomplete information from FB to satisfy the query.

Generally, it is combined with the forward chaining :

- forward chaining to saturate the FB
- backward chaining to guide the interaction with users of a AND/OR tree.

Backforward Chaining

Principle

Algorithm of replacing a goal by sub-goals, by matching a current goal (the user's query at beginning) with the conclusion atoms of rules.

When to use it ?

To guide the questions proposed by users in the case of incomplete information from FB to satisfy the query.

Generally, it is combined with the forward chaining :

- forward chaining to saturate the FB
- backward chaining to guide the interaction with users of a AND/OR tree.

Backforward Chaining

Principle

Algorithm of replacing a goal by sub-goals, by matching a current goal (the user's query at beginning) with the conclusion atoms of rules.

When to use it ?

To guide the questions proposed by users in the case of incomplete information from FB to satisfy the query.

Generally, it is combined with the forward chaining :

- forward chaining to saturate the FB
- backward chaining to guide the interaction with users of a AND/OR tree.

Backward Chaining Algorithm

```
1 : function BackwardChaining(Q, FB_init, RB)
2 :     FB ← FB_init
3 :     if match(Q, FB) then
4 :         return true
5 :     else
6 :         for all Rule r : If l_1 And l_2 And ... And l_n Then Conc t.q.
    match(Q, Conc) do
7 :             bool ← true ; i ← 1
8 :             while bool And i ≤ n do
9 :                 bool ← BackwardChaining(l_i, FB, RB)
10 :                i ← i + 1
11 :            end while
12 :            if bool then return true
13 :            end if
14 :        end for
15 :        return false
16 :    end if
17 : end function
```

Backward Chaining with questions to users

Principles of the questions

To decide the questions that can be asked to users.

- *demandable*(l) : we can ask a question over l.Generally, if l doesn't appear in conclusions of rules, it is demandable.
- *question*(l) : boolean returning the answer of the user to the question : if l is *true*.

Modification Principles : If we cannot find a fact, we can ask the user as last resort.

Backward Chaining with questions to users

Principles of the questions

To decide the questions that can be asked to users.

- *demandable*(l) : we can ask a question over l.Generally, if l doesn't appear in conclusions of rules, it is demandable.
- *question*(l) : boolean returning the answer of the user to the question : if l is *true*.

Modification Principles : If we cannot find a fact, we can ask the user as last resort.

Backward Chaining with questions to users

Principles of the questions

To decide the questions that can be asked to users.

- *demandable*(l) : we can ask a question over l.Generally, if l doesn't appear in conclusions of rules, it is demandable.
- *question*(l) : boolean returning the answer of the user to the question : if l is *true*.

Modification Principles : If we cannot find a fact, we can ask the user as last resort.

Backward Chaining with questions to users

Principles of the questions

To decide the questions that can be asked to users.

- *demandable*(l) : we can ask a question over l.Generally, if l doesn't appear in conclusions of rules, it is demandable.
- *question*(l) : boolean returning the answer of the user to the question : if l is *true*.

Modification Principles : If we cannot find a fact, we can ask the user as last resort.

Backward Chaining with questions to users

Principles of the questions

To decide the questions that can be asked to users.

- *demandable*(l) : we can ask a question over l.Generally, if l doesn't appear in conclusions of rules, it is demandable.
- *question*(l) : boolean returning the answer of the user to the question : if l is *true*.

Modification Principles : If we cannot find a fact, we can ask the user as last resort.

Backward Chaining with interactive questions

function *BackwardChaining2*(Q, FB_{init}, RB)
  $FB \leftarrow FB_{init}$
  if *match*(Q, FB) **then**
    return *true*
  else
    for all rgle r : **if** l_1 **and** l_2 **and** ... **and** l_n **then** *Conc* t.q. *match*$(Q, Conc)$
do
      $bool \leftarrow true$; $i \leftarrow 1$
      while *bool* Et $i \leq n$ **do**
        $bool \leftarrow BackwardChaining(l_i, FB, RB)$
        $i \leftarrow i + 1$
      end while
      if *bool* **then return** *true*
      end if
    end for
    if *demandable*(Q) **then return** *question*(Q)
    elsereturn *false*
    end if
  end if
end function

Exemple

Tutorial 1, ex2.

Recursive Rules

Problem of loopback risk

Solution : don't re-apply the algorithm over the sub-goals that have been examined.

```
function BackwardChaining3(Q, FB_init, RB)
    FB ← FB_init
    if match(Q, FB) then
        return true
    else
        AlreadyTried ← ∅
        for all Rule r : if l_1 and l_2 and ... and l_n then Conc t.q. match(Q, Conc) do
            bool ← true ; i ← 1
            while bool And i ≤ n do
                if l_i ∉ AlreadyTried then
                    AlreadyTried ← AlreadyTried ∪ {l_i}
                    bool ← BackwardChaining(l_i, FB, RB)
                end if
                i ← i + 1
            end while
            if bool then return true
            end if
        end for
        if demandable(Q) then return question(Q)
        elsereturn false
        end if
    end if
end function
```

Recursive Rules

Problem of loopback risk

Solution : don't re-apply the algorithm over the sub-goals that have been examined.

function *BackwardChaining*3(Q, FB_{init}, RB)
  $FB \leftarrow FB_{init}$
  if *match*(Q, FB) **then**
    return *true*
  else
    AlreadyTried $\leftarrow \emptyset$
    for all Rule r : **if** l_1 **and** l_2 **and** ... **and** l_n **then** *Conc* t.q. *match*(Q, *Conc*) **do**
      bool $\leftarrow$ *true* ; $i \leftarrow 1$
      while *bool* And $i \leq n$ **do**
        if $l_i \notin$ *AlreadyTried* **then**
          AlreadyTried $\leftarrow$ *AlreadyTried* $\cup \{l_i\}$
          bool $\leftarrow$ *BackwardChaining*(l_i, FB, RB)
        end if
        $i \leftarrow i + 1$
      end while
      if *bool* **then return** *true*
      end if
    end for
    if *demandable*(Q) **then return** *question*(Q)
    elsereturn *false*
    end if
  end if
end function