Open Information Systems

Lecture 3: Description Logics

Christophe Debruyne

Outline

- Limitations of RDFS
- Why the Web Ontology Language?
- Description Logics as the foundation of OWL
- Concepts, roles, individuals
- Open/Closed World Assumption
- Unique Name Assumption
- Interpretations and reasoning tasks
- Reasoning with the Tableau Algorithm

Web Ontology Language

RDF(S) too weak to describe resources in sufficient detail

- No localized range and domain constraints.
  - E.g., cannot state that the range of hasChild is person when applied to persons and elephant when applied to elephants
- No combination of classes with union, intersection, and complement.
- No existence/cardinality constraints.
  - E.g., cannot state that all persons have a mother that is also a person, or that persons have exactly 2 parents
- No transitive, inverse or symmetrical properties.
  - E.g., cannot state that isPartOf is a transitive property, that hasPart is the inverse of isPartOf, or that touches is symmetrical
- …

Difficult to provide reasoning support

- No “native” reasoners for non-standard semantics

Example

Dogs and cats can have names. Is the following correct?

```
:name a rdf:Property ;
   rdfs:domain :Dog ;
   rdfs:domain :Cat ;
   rdfs:range xsd:string ;
   .
```

No. Why not?

If something plays the role of having a name, then it is an instance of both :Dog and :Cat. In other words: multiple domain/range declarations act as a conjunction.

Can we model something that represents Dogs UNION Cats?

In RDFS only by using superclasses, which may not be elegant.

Motivation

- Why do we need reasoning?
  - Ensure that a knowledge base (KB) is
    - Meaningful - all named classes can have instances
    - Correct - captured intuitions of domain experts
    - Minimally redundant - no unintended synonyms

- Answer queries over ontology classes and instances, e.g.:
  - Find more general and specific classes
  - Retrieve individuals or tuples matching a given query

Web Ontology Language (OWL)

- Ontology languages SHOULD allow users to provide an explicit, formal conceptualization of a shared universe of discourse.

- OWL is a W3C Recommendation
  - OWL Web Ontology Language was published in 2004
  - OWL Web Ontology Language 2 was published in 2012

- Motivations: a well-defined syntax, a formal semantics, and efficient reasoning support

- But before we shall cover OWL, we will cover Description Logics (DL). DLs provide the formal foundation of OWL. In this lecture, we will introduce DLs with the goal of better understanding some of OWLs aspects (in the next lecture).

What are Description Logics?

- A family of Knowledge Representation languages
  - Describe the domain of discourse in terms of Concepts, Roles, and Individuals.
  - formal semantics
  - DLs are decidable fragments of First Order Logic (FOL)
    - FOL satisfiability well known to be undecidable. A sound, complete and terminating algorithm is therefore impossible.
- Allows one to extract implicit knowledge about concepts and individuals via a reasoning engine.
- Among those reasoning tasks are, for example,
  - Subsumption checking to check whether a concept is a subset (i.e., "subclass") of another concept.
  - Concept consistency to check whether a concept can have instances.

Why Description Logics?

- They are well suited to describe a UoD and have been put forward as the foundation of the Web Ontology Language
- They have been well-studied and guarantee certain properties.
- Supported by various tools and libraries.

- Note, however, that there are different assumptions in DL and OWL, which we will highlight and emphasize in both lectures.

DL Basics

- Individuals (individuals in OWL, and constants in FOL):
  - objects, elements of a specific domain
  - christophe, open_information_systems, trinity_college_dublin, ...
- Concepts (classes in OWL, and unary predicates in FOL):
  - sets of individuals that share some properties
  - Student, Graduate Course, ProfessorOnlyTeachingGradCourses, ...
- Roles (properties in OWL, and binary predicates in FOL):
  - binary relations between concepts
  - teaches, attends, supervises, teaches and supervises, ...
- Constructors:
  - to build complex concepts and roles from atomic concepts and roles
  - Example of a complex concept: Student and Broke (Broke Students)

DL Basics

- DL dialects are named following a naming convention stating which operators are allowed starting from a base language

- The base language $\mathcal{AL}$ (Attributive Language) allows
  - *atomic* negation, *but not on the LHS of inclusion and equality axioms*
  - concept intersection
  - universal restrictions
  - existential restriction, *but only using the universal concept*

- Additional letters indicate other extensions ...

DL Basics

$\mathcal{F}$	functional properties
$\mathcal{E}$	qualify existential restrictions
$\mathcal{C}$	complex concept negation
$\mathcal{U}$	concept union
$\mathcal{H}$	role hierarchies
$\mathcal{R}$	role inclusion axioms, reflexivity, irreflexivity, and role disjointness
$\mathcal{O}$	(nominals) using individuals in concept descriptions
$\mathcal{I}$	inverse properties
$\mathcal{N}$	cardinality restrictions (not qualified)
$\mathcal{Q}$	qualified cardinality restrictions
($\mathcal{D}$)	using data type properties, data types and data values.

DL Basics

- $\mathcal{AL}$ is the base language, and $\mathcal{ALC}$is a popular language
- $\mathcal{ALC}$ with transitive roles ($\mathcal{R}_+$) is called $\mathcal{S}$
- $\mathcal{SHOIN}(\mathcal{D})$ – Underlying formalism of OWL 1 DL
  - $\mathcal{S}$ + role hierarchies + nominals + inverse roles + number restrictions + datatypes
- $\mathcal{SROIQ}(\mathcal{D})$ – Underlying formalism of OWL 2

- Since $\mathcal{ALC}$ is equivalent with $\mathcal{ALEU}$, we will also consider qualified existential restrictions and disjunction.
  - Because of $\mathcal{C}$, we have
    - $C \sqcup D \Leftrightarrow \neg(\neg C \sqcap \neg D)$ (De Morgan) and
    - $\exists R.C \Leftrightarrow \neg\forall R.\neg C$
  - $\mathcal{ALEU}$ can thus be "rewritten" as $\mathcal{ALC}$

$\mathcal{ALC}$ Concepts

- Basic vocabulary of the following disjoint sets:
  - N_C of concept names (inc. the universal concept $\top$ and bottom concept $\bot$)
  - N_R of role names
  - N_I of individual names

- The definition of an $\mathcal{ALC}$ concept description is defined as follows:
  - If $A \in N_C$, then A is an $\mathcal{ALC}$ concept description (atomic).
  - If C and D are $\mathcal{ALC}$ concept descriptions, and $r \in N_R$, then the following are $\mathcal{ALC}$ concept descriptions:
    - $C \sqcap D$ (conjunction)
    - $C \sqcup D$ (disjunction)
    - $\neg C$ (negation)
    - $\forall r.C$ (value restriction)
    - $\exists r.C$ (existential restriction)
    - Nothing else is an ALC concept description.

- Note: inverse properties r^- are not part of $\mathcal{ALC}$, but we will cover them in the exercises.

$\mathcal{ALC}$ Concepts

$\neg Student$	NOT Student
$Lecturer \sqcap Student$	Lecturer AND Student
$Lecturer \sqcup Student$	Lecturer OR Student
$\forall attends.Lecture$	attends ONLY Lecture(s)
$\exists teaches.Lecture$	teaches SOME Lecture

$\mathcal{ALC}$ Concepts

Negation is interpreted as the complement of sets of individuals	$\neg Student$
Conjunction is interpreted as the intersection of sets of individuals	$Lecturer \sqcap Student$
Disjunction is interpreted as the union of sets of individuals	$Lecturer \sqcup Student$
Value restriction requires all individuals to belong to the concept associated with the relationship	$\forall attends.Lecture$
Existential restriction requires some individuals to belong to the concept associated with the relationship	$\exists teaches.Lecture$

Which brings us to the semantics of ALC

$\mathcal{ALC}$ Concepts

The semantics is defined by *interpretations*. An interpretation I is a pair $(\Delta^I, .^I)$ where Δ^I is the domain (a non-empty set) and $.^I$ is an interpretation function mapping:

- Every concept to a subset of Δ^I
- Every role to a subset of $\Delta^I \times \Delta^I$
- Every individual to an element of Δ^I

Description	Syntax	Semantics
Top (everything)	$\top$	Δ^I
Bottom (empty concept)	$\bot$	$\emptyset$
Atomic concept ($\in N_C$)	A	$A^I \subseteq \Delta^I$
Atomic role ($\in N_R$)	r	$r^I \subseteq \Delta^I \times \Delta^I$
Concept conjunction	$C \sqcap D$	$C^I \cap D^I$
Concept disjunction	$C \sqcup D$	$C^I \cup D^I$
Concept negation (complement)	$\neg C$	$\Delta^I \backslash C^I$
Universal restriction	$\forall r.C$	$\{x \mid x \in \Delta^I \wedge \forall y(r(x,y) \in r^I \rightarrow y \in C^I)\}$
Existential restriction	$\exists r.C$	$\{x \mid x \in \Delta^I \wedge \exists y(r(x,y) \in r^I \wedge y \in C^I)\}$

$\mathcal{ALC}$ Concepts

- We now know how to make concept (and role) descriptions, but how can we describe how concepts (and roles) relate to each other?
- E.g., "A Teacher is a type of Human" (specialization, inclusion) and "Human is equivalent to Person" (equivalence)
- DLs allow one to state these as axioms about concepts (and roles).

Description	Syntax	Semantics
General Concept Inclusion	$C \sqsubseteq D$	$C^I \subseteq D^I$
Concept Equivalence	$C \equiv D$	$C^I = D^I$

- $C \equiv D$ is basically a shorthand for stating both $C \sqsubseteq D$ and $D \sqsubseteq C$.
- Note that $\mathcal{ALC}$ does not provide role axioms. There are more expressive DLs that do. E.g., to state that $childOf \equiv parentOf^-$

A DL Knowledge Base (KB)

- A DL KB is a pair $< \mathcal{T}, \mathcal{A} >$, where
  - $\mathcal{T}$ is a set of terminological axioms (the Terminology Box, or TBox)
  - $\mathcal{A}$ is a set of assertional axioms (the Assertion Box, or ABox)
- A TBox can be compared with a schema
- The ABox can be compared with the data

- We will cover the most important aspects of KBs, but for details I refer to [1] and [2].

Example of a KB
Tbox
$Lecturer \equiv Person \sqcap \neg Student$ $Student \equiv Person \sqcap \exists attends.Lecture$
Abox
$Lecturer(christophe)$ $teaches(christophe, ois)$

The TBox

- A TBox is a finite set of General Concept Inclusions and Concept Equivalences.

- A definition is a concept equality where the left-hand side is atomic. They are useful to provide symbolic names to complex descriptions

$$CatPerson \equiv Person \sqcap \exists owns.Cat$$

- What if we cannot define a concept completely?
  - Concept inclusion allows us to model the necessary conditions using specialization. For instance, what makes a cat a cat? We might have trouble defining something as follows: $Cat \equiv Animal \sqcap \cdots$ and thus "merely" require that all Cats are animals.

$$Cat \sqsubseteq Animal$$

The TBox

- A Generalized Terminology is a TBox where:
  - The LHS of all axioms are an atomic concept, and
  - Each atomic concept only appears once on the LHS of an axiom

- A Regular Terminology is a Generalized Terminology only containing concept equivalences.

- A generalized terminology $\mathcal{T}$ can be transformed into a regular terminology $\mathcal{T}'$ by "rewriting" concept inclusions with the following trick: $Cat \sqsubseteq Animal$ in $\mathcal{T}$ becomes $Cat \equiv Cat' \sqcap Animal$ where Cat' is a concept that represents the qualities that distinguish a cat from other animals (even though we do not know what).

The ABox

- A set $\mathcal{A}$ of assertions is called an Abox
- Describes the facts specific to an application domain
  - Cat(victor)
  - Person(christophe)
  - owns(christophe, victor)
- Given interpretation $I = (\Delta^I, .^I)$
  - $C(a)$ is satisfied by I iff $a^I \in C^I$
  - $r(a, b)$ is satisfied by I iff $(a^I, b^I) \in r^I$
- An Abox $\mathcal{A}$ is said to be satisfied with respect to an interpretation I iff every assertion in $\mathcal{A}$ is satisfied. This is written as $I \vDash \mathcal{A}$.

The ABox

- There are two common assumptions for the Abox

- The Unique Name Assumption (UNA) infers that individual names in I are distinct. In other words: $a^I \neq b^I$

- The Open World Assumption (OWA) presumes that the information in the ABox is incomplete. I.e., assertions in the ABox represent statements that are true in all models, but not that all unknown information not occurring in the models are not true. → "What is not known, is not necessarily false."

Before we continue with reasoning

- First describe the relationship between DLs and FOL and we will see how we can transform formulas in DLs into formulas in FOL. This may be handy if you have seen semantic tableaus for FOL in your undergraduate. This part may be skipped if you merely want to get the gist of DLs.

- We will also describe the differences between databases and ontologies (and ontologies on the Semantic Web). This part is useful to understand some of the more "quirkier" aspects of DLs and its behavior when used on the Semantic Web.

Relationship with FOL

- Description logics are decidable fragments of FOL. Concept descriptions correspond to FOL with one free variable, which will be bound when used in a concept inclusion or equivalence statement [1].

- Translation of assertions in a DL into FOL formulas are provided by [1]. The translation of concept description C into a FOL formula with one free variable $\tau_x(C)$ is defined as follows:
  - $\tau_x(A) := A(x)$ for atomic concepts A
  - $\tau_x(C \sqcap D) := \tau_x(C) \wedge \tau_x(D)$
  - $\tau_x(C \sqcup D) := \tau_x(C) \vee \tau_x(D)$
  - $\tau_x(\neg C) := \neg\tau_x(C)$
  - $\tau_x(\forall r.C) := \forall y(r(x,y) \rightarrow \tau_y(C))$, where variable $y \neq x$
  - $\tau_x(\exists r.C) := \exists y(r(x,y) \wedge \tau_y(C))$ where variable $y \neq x$

- Given a TBox $\mathcal{T}$ with concept-inclusions(*), the translation $\tau(\mathcal{T})$ of $\mathcal{T}$ is given by:

$$\tau(\mathcal{T}) := \bigwedge_{C \sqsubseteq D \in \mathcal{T}} \forall x(\tau_x(C) \rightarrow \tau_x(D))$$

(*) Assuming we have rewritten all equivalences into two concept-inclusions

Difference between databases and ontologies

- Database:
  - Closed World Assumption (CWA). Missing information is treated as false
  - Unique Name Assumption (UNA). Each individual has a single, unique name
  - Schema contains constraints that define legal database populations

- Ontologies:
  - No CWA, but OWA. Missing information treated as unknown
  - DLs **usually** adopt a UNA. Each individual has a single, unique name.
  - OWL does not adopt UNA. Each individual can have multiple names.
  - Axioms are used to infer implicit information from explicit information.

- Why would OWL not adopt UNA?

Difference between databases and ontologies

- Assume the following facts are in our KB:
  - Cat(Victor)
  - Cat(Bettina)
  - Cat(Gaston)
  - hasPet(Christophe, Victor)
  - hasPet(Christophe, Bettina)

- Is Gaston Christophe's pet?
  - Database: No.
  - Ontology: Open World Assumption leads to "we do not know". It was not explicitly stated that Gaston was NOT Christophe's Pet.

Difference between databases and ontologies

- Assume the following facts are in our KB:
  - Cat(Victor)
  - Cat(Bettina)
  - Cat(Gaston)
  - hasPet(Christophe, Victor)
  - hasPet(Christophe, Bettina)

- How many pets does Christophe have?
  - Database: 2
  - Ontology (with UNA): At least 2
  - Ontology (without UNA): At least 1 – The individual names "Victor" and "Bettina" may refer to the same cat. It was not explicitly stated that they referred to different things.

Difference between databases and ontologies

- Assume the following facts are in our KB:
  - Cat(Victor)
  - Cat(Bettina)
  - Cat(Gaston)
  - hasPet(Christophe, Victor)
  - hasPet(Christophe, Bettina)
  - **Victor ≠ Bettina** cfr. owl:differentFrom in the next lecture

- How many pets does Christophe have?
  - Database: 2
  - Ontology (with UNA): At least 2
  - Ontology (without UNA): At least 2

Difference between databases and ontologies

- Assume the following facts are in our KB:
  - Cat(Victor)
  - Cat(Bettina)
  - Cat(Gaston)
  - hasPet(Christophe, Victor)
  - hasPet(Christophe, Bettina)
  - Victor ≠ Bettina
  - **$(\forall \text{hasPet}.(\{\text{Victor, Bettina}\}))(\text{Christophe})$**

With some abuse of notation. One can define a class as a set of individuals. See allValuesFrom (enabled by $\mathcal{O}$)

- How many pets does Christophe have?
  - Database: 2
  - Ontology (with UNA): 2
  - Ontology (without UNA): 2

Reasoning

- Why do we need reasoning?
  - Ensure that a knowledge base (KB) is meaningful, correct, minimally redundant
  - Answer queries over ontology classes and instances

- DL systems allows one not only to describe a universe of discourse, but also to reason about a knowledge base $\mathcal{K} =< \mathcal{T}, \mathcal{A} >$
  - A knowledge base $\mathcal{K}$ is said to be satisfied by an interpretation I if and only if I satisfies both $\mathcal{T}$ and $\mathcal{A}$.
  - A knowledge base $\mathcal{K}$ is consistent iff there exists a model for $\mathcal{K}$
  - Testing the satisfiability of a knowledge base can be done by checking whether $\mathcal{K} \not\models \bot$ holds.

Reasoning Services

Knowledge Base
TBox
$CatPerson \equiv Person \sqcap \exists owns.Cat$ $Cat \sqsubseteq Animal$ $DogPerson \equiv Person \sqcap \neg CatPerson$
ABox
$Person(christophe)$ $Cat(victor)$ $owns(christophe, victor)$

Service	***Formal***	***Example***
Concept Satisfiability	$\mathcal{K} \not\models C \sqsubseteq \bot$	$CatPerson \sqcap \neg Person$
Subsumption	$\mathcal{K} \models C \sqsubseteq D$	$CatPerson \sqsubseteq Person$
KB Satisfiability	$\mathcal{K} \not\models \bot$	$\neg CatPerson(christophe)$
Instance Checking	$\mathcal{K} \models C(a)$	$Cat(victor)$
Retrieval	$\{a \mid \mathcal{K} \models C(a)\}$	$Animal$ => {victor}

How?

- Most reasoners are based on the tableau algorithm (or semantic tableaux) to solve the satisfiability problem.
- These are sound, complete, and terminating (EXPTIME).
- Problems can be transformed into other types of problems to avail of this technique.

Reasoning Services

Many reasoning problems can be reduced to other problems:

1. $C \equiv_{\mathcal{K}} D$ iff $C \sqsubseteq_{\mathcal{K}} D$ and $D \sqsubseteq_{\mathcal{K}} C$
2. $C \sqsubseteq_{\mathcal{K}} D$ iff $C \equiv_{\mathcal{K}} C \sqcap D$
3. $C \sqsubseteq_{\mathcal{K}} D$ iff $C \sqcap \neg D$ is unsatisfiable w.r.t. $\mathcal{K}$
4. C is satisfiable w.r.t. $\mathcal{K}$ iff $C \not\equiv_{\mathcal{K}} \bot$
5. C is satisfiable w.r.t. $\mathcal{T}$ iff $< \mathcal{T}, \{C(a)\} >$ is consistent
6. $\mathcal{K}$ is consistent iff a is not an instance of $\bot$ w.r.t. $\mathcal{K}$
7. a is an instance of C w.r.t. $\mathcal{K}$ iff $< \mathcal{T}, \mathcal{A} \cup \{\neg C(a)\} >$ is inconsistent

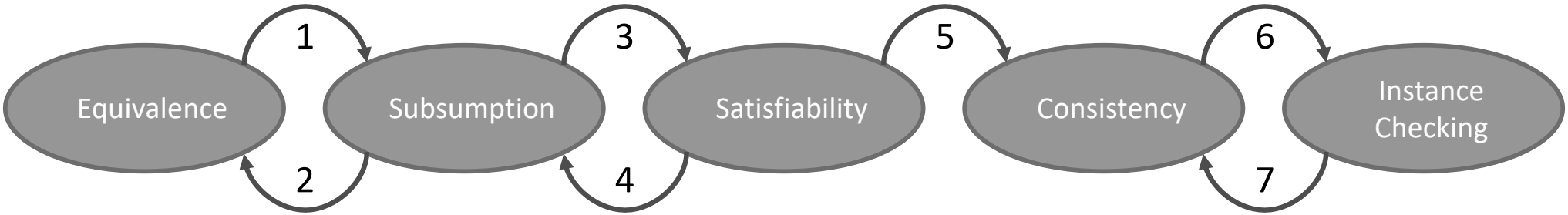


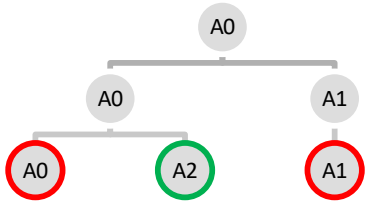
Tableau Algorithm for ABox Consistency

- Try to generate a finite model for the input ABox $\mathcal{A}_0$
  - Apply tableau rules on ABox , which may result in new ABoxes
  - Termination: when no rules can be applied to any ABox
  - Answer:
    - Consistent if there is an open and complete ABox
    - Inconsistent if all ABoxes are closed (not open)

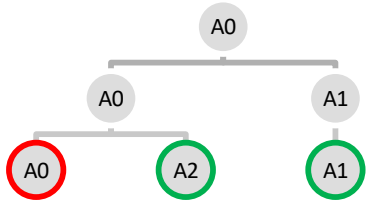
- An obvious contradiction is of the form $C(a)$ and $\neg C(a)$
- $\mathcal{A}$ is open when there is no obvious contradiction in $\mathcal{A}$
- $\mathcal{A}$ is complete when no more rules can be applied on $\mathcal{A}$

Tableau Algorithm for ABox Consistency

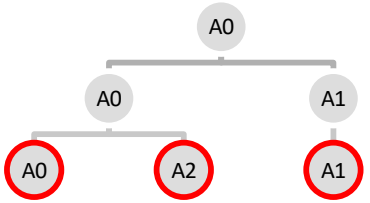
- Assume we start from an ABox $\mathcal{A}_0$
  - We apply transformation rules until either an ABox contains an obvious contradiction, or an ABox is open and complete.
  - Red circles indicate an ABox with an obvious contradiction
  - Green circles indicate an ABox that is open and complete
  - Are the following KBs consistent?



There is one open and complete ABox. There is a model for the KB and the KB is therefore consistent.



There is one open and complete ABox. There is a model for the KB and the KB is therefore consistent.



There is no open and complete ABox. All ABoxes contain contradictions. There is no model for KB and the KB is therefore inconsistent.

TBox Reasoning

- What if we only have a TBox?
- Remember we only deal with Generalized Terminologies:
  - The LHS of all axioms are an atomic concept, and
  - Each atomic concept only appears once on the LHS of an axiom
- We will also avoid cyclic terminologies ("recursive definitions") for this lecture, this allows us to "reduce" TBox reasoning to ABox reasoning [2].
- For example.

Concept Satisfiability

- What if we only have a TBox and we want to test whether a concept can have instances (is satisfiable)?

TBox
$CatPerson \equiv Person \sqcap \exists owns.Cat$
$Cat \sqsubseteq Animal$
$DogPerson \equiv Person \sqcap \neg CatPerson$

Can the concept Cat have instances?

- In order to test the satisfiability of C, the algorithm starts with the ABox $\mathcal{A}_0 = \{C(x)\}$, and apply transformation rules to the ABox until no more rules apply.
  - If one of the complete ABoxes obtained this way does not contain a contradiction, then $\mathcal{A}_0$ is consistent (and thus C is satisfiable).
  - If all the ABoxes obtained this way do contain a contradiction, then $\mathcal{A}_0$ is inconsistent (and thus C is unsatisfiable).

Satisfiability

- An interpretation *I* satisfies a TBox $\mathcal{T}$ if it satisfies every axiom in $\mathcal{T}$:
  - For every $C \sqsubseteq D$ you have $C^I \subseteq D^I$
- An interpretation *I* satisfies an ABox $\mathcal{A}$ if it satisfies every assertion in $\mathcal{A}$:
  - For every $C(a)$ in $\mathcal{A}$, $a^I \in C^I$ (and thus also in Δ^I)
  - For every *R(a, b)* in $\mathcal{A}$, $(a^I, b^I) \in R^I$
- An interpretation I satisfies a KB if it satisfies both the TBox and the ABox. A KB is said to be satisfiable if there exists at least one non-empty interpretation (and domain) satisfying that KB.

- Note: it may very well be that your KB is satisfiable, but concepts in the KB are not. Concept satisfiability checking is a separate task. We will cover such an example later.

Negation Normal Forms

- Before we start applying the Tableau Algorithm, we normalize all concept descriptions into Negation Normal Forms (NNF).
- The goal of NNFs is to "rewrite" and "push" negations in formulas so that the only occur in front of atomic concepts.
  - $\neg(C \sqcap D) \equiv (\neg C \sqcup \neg D)$
  - $\neg(C \sqcup D) \equiv (\neg C \sqcap \neg D)$
  - $\neg\exists r.C \equiv \forall r.\neg C$
  - $\neg\forall r.C \equiv \exists r.\neg C$
  - $\neg\neg C \equiv C$

- What about concept equivalence and concept inclusion? Remember that:
  - Concept equivalence can be rewritten as two concept inclusions
  - $\mathcal{ALC}$ is equivalent with $\mathcal{ALEU}$, allowing is to rewrite $C \sqsubseteq D$ as $(\neg C \sqcup D)$
  - This allows us to only avail of rules for $\sqcap$,$\sqcup$, $\forall$, and $\exists$

Transformation Rules

⊓-rule	If $\mathcal{A}$ contains $(C \sqcap D)(x)$, but it does not contain both $C(x)$ and D(x), then $\mathcal{A} = \mathcal{A} \cup \{C(x), D(x)\}$
⊔-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 $\mathcal{A}' = \mathcal{A} \cup \{D(x)\}$
∃-rule	If $\mathcal{A}$ contains $(\exists R.C)(x)$, but there is no individual z such that $C(z)$ and $R(x,z)$ are in $\mathcal{A}$, then $\mathcal{A} = \mathcal{A} \cup \{C(y), R(x,y)\}$ where y is an individual name not occurring in $\mathcal{A}$
∀-rule	If $\mathcal{A}$ contains $(\forall R.C)(x)$ and $R(x,y)$, but it does not contain $C(y)$, then $\mathcal{A} = \mathcal{A} \cup \{C(y)\}$

Transformation Rules

$\sqsubseteq$-rule	Either we transform the $\sqsubseteq$ into a $\equiv$ using an "artificial" concept and use the transformation rules below, OR: **For every individual a in $\mathcal{A}$**: if $C \sqsubseteq D$ is in $\mathcal{T}$ and $(\neg C \sqcup D)(a)$ is not in $\mathcal{A}$, then $\mathcal{A} = \mathcal{A} \cup \{(\neg C \sqcup D)(a)\}$
	The following rules are for acyclic terminologies with only atomic concepts on the LHS.
$\equiv$-rule	If $\mathcal{A}$ contains $A \equiv C$, where A is atomic, and $\mathcal{A}$ contains $A(x)$ and $\mathcal{A}$ does not contain $C(x)$, then $\mathcal{A} = \mathcal{A} \cup \{C(x)\}$
$\equiv$-rule	If $\mathcal{A}$ contains $A \equiv C$, where A is atomic, and $\mathcal{A}$ contains $\neg A(x)$ and $\mathcal{A}$ does not contain $\neg C(x)$, then $\mathcal{A} = \mathcal{A} \cup \{\neg C(x)\}$

Examples

Is the concept $\exists hasFriend.Person$ satisfiable?

1. The concept is in NNF
2. Prepare the ABox $\mathcal{A}_0$ and instantiate the concept with individual x.
3. Apply transformation rules.

$\mathcal{A}_0$	$(\exists hasFriend.Person)(x)$
	↓ ∃-rule
$\mathcal{A}_0$	$(\exists hasFriend.Person)(x)$ $hasFriend(x, y)$ $Person(y)$

No more transformation rules can be applied. All ABoxes are open and complete. The concept is thus satisfiable.

Examples

Is the concept $\neg\exists attends.Course \sqcap \neg\forall attends.(\neg Course)$ satisfiable?

1. The concept is NOT in NNF
   - $\neg\exists attends.Course \sqcap \neg\forall attends.(\neg Course)$
   - $\forall attends.\neg Course \sqcap \neg\forall attends.(\neg Course)$
   - $\forall attends.\neg Course \sqcap \exists attends.(\neg\neg Course)$
   - $\forall attends.\neg Course \sqcap \exists attends.Course$
2. Prepare the ABox $\mathcal{A}_0$ and instantiate the concept with individual x.
3. Apply transformation rules.

Examples

$\mathcal{A}_0$	$(\forall attends.\neg Course \sqcap \exists attends.Course)(x)$
	↓ ⊓-rule
$\mathcal{A}_0$	$(\forall attends.\neg Course \sqcap \exists attends.Course)(x)$ $(\forall attends.\neg Course)(x)$ $(\exists attends.Course)(x)$
	↓ ∃-rule
$\mathcal{A}_0$	$(\forall attends.\neg Course \sqcap \exists attends.Course)(x)$ $(\forall attends.\neg Course)(x)$ $(\exists attends.Course)(x)$ $attends(x,y)$ $Course(y)$
	↓ ∀-rule
$\mathcal{A}_0$	$(\forall attends.\neg Course \sqcap \exists attends.Course)(x)$ $(\forall attends.\neg Course)(x)$ $(\exists attends.Course)(x)$ $attends(x,y)$ $Course(y)$ $\neg Course(y)$

Even though this ABox is complete, we could terminate upon finding an obvious contradiction. All ABoxes contain contradictions, therefore the concept is not satisfiable.

Examples

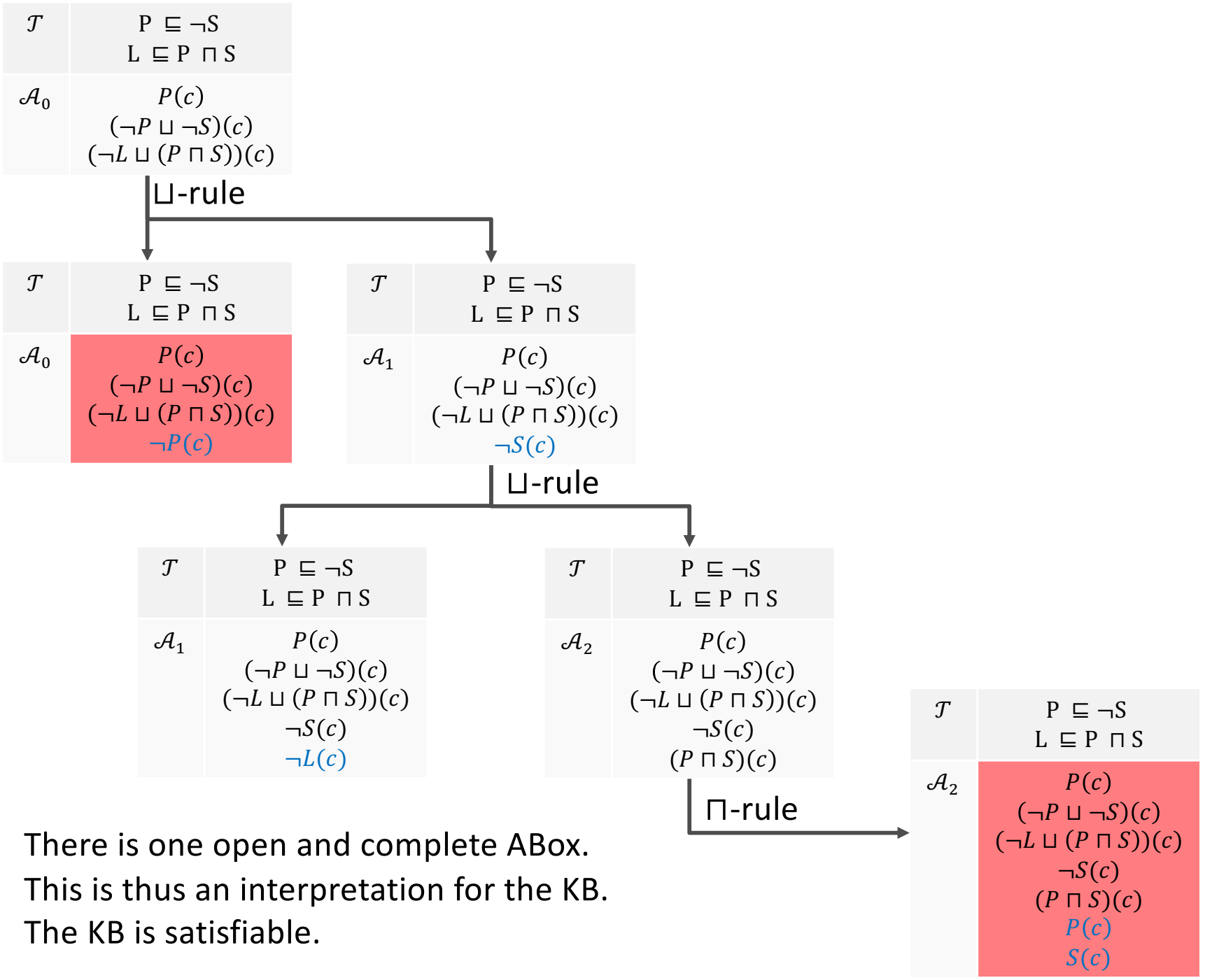
Is the following KB satisfiable?

Notice that the concept Lecturer is not satisfiable. We are, however, going to check KB satisfiability. Is there a non-empty interpretation satisfying all axioms?

$\mathcal{T}$	$Professor \sqsubseteq \neg Student$ $Lecturer \sqsubseteq Professor \sqcap Student$
$\mathcal{A}_0$	$Professor(christophe)$

$\sqsubseteq$-rule to all individuals. Notice I abbreviated the symbols

$\mathcal{T}$	$\mathrm{P} \sqsubseteq \neg \mathrm{S}$ $\mathrm{L} \sqsubseteq \mathrm{P} \sqcap \mathrm{S}$
$\mathcal{A}_0$	$P(c)$ $(\neg P \sqcup \neg S)(c)$ $(\neg L \sqcup (P \sqcap S))(c)$



There is one open and complete ABox.

This is thus an interpretation for the KB.

The KB is satisfiable.

Examples

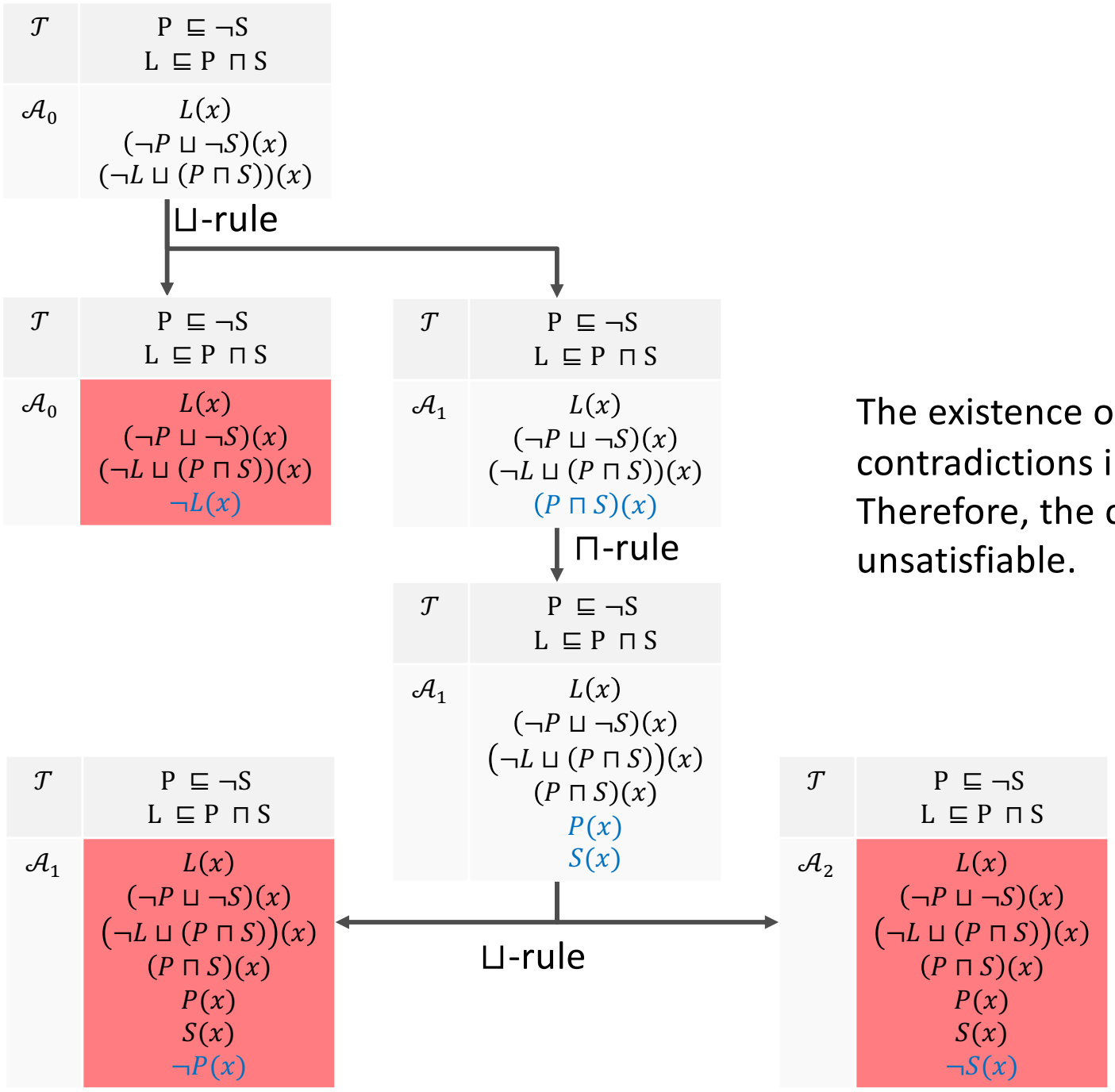
Is the concept Lecturer satisfiable?

The algorithm starts with the ABox $\mathcal{A}_0 = \{Lecturer(x)\}$, and apply the transformation rules to the ABox until no more rules apply.

$\mathcal{T}$	$Professor \sqsubseteq \neg Student$ $Lecturer \sqsubseteq Professor \sqcap Student$
$\mathcal{A}_0$	$Lecturer(x)$

$\sqsubseteq$-rule to all individuals.

$\mathcal{T}$	$\mathrm{P} \sqsubseteq \neg \mathrm{S}$ $\mathrm{L} \sqsubseteq \mathrm{P} \sqcap \mathrm{S}$
$\mathcal{A}_0$	$L(x)$ $(\neg P \sqcup \neg S)(x)$ $(\neg L \sqcup (P \sqcap S))(x)$



The existence of a Lecturer leads to contradictions in all ABoxes. Therefore, the concept Lecturer is unsatisfiable.

Concluding remarks

- DLs provide the foundation for OWL, which is why we covered DLs in this lecture.
- We introduced various important concepts (not to be confused with DL concepts ☺) that we will mention in the next lecture.
- We have only scratched the surface of DLs. If you are interested in this topic, I suggest starting with the references included in this presentation.

References

1. F. Baader, I. Horrocks, and U. Sattler. Description logics. Foundations of Artificial Intelligence, 3:135–179, 2008.

2. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. Patel-Schneider (eds.). The Description Logic Handbook. Theory, Implementation and Applications, Cambridge University Press, 2003.

3. Grigoris Antoniou, Frank van Harmelen: Web Ontology Language: OWL. Handbook on Ontologies 2009: 91-110

4. Web Ontology Language (OWL) 1.0
   - http://www.w3.org/2001/sw/WebOnt/

5. OWL 2.0
   - http://www.w3.org/TR/2009/WD-owl2-new-features-20090421/
   - https://www.w3.org/TR/owl2-syntax/