Outline

Outline

Outline

Computational complexity

[J.E. Hopcroft, 2007; Papadimitriou, 1994]
Computational complexity theory aims at understanding how difficult it is to solve specific problems.

- A problem is considered as an (in general infinite) set of instances of the problem, each encoded in some meaningful (i.e., compact) way.
- Standard complexity theory deals with decision problems: i.e., problems that admit a yes/no answer.
- Algorithm that solves a decision problem:
    - input: an instance of the problem
    - output: yes or no
- The difficulty (complexity) is measured in terms of the amount of resources (time, space) that the algorithm needs to solve the problem.
  $\rightsquigarrow$ complexity of the algorithm, or upper bound
- To measure the complexity of the problem, we consider the best possible algorithm that solves it.
  $\rightsquigarrow$ lower bound

Computational complexity

- Worst-case complexity analysis: the complexity is measured in terms of a (complexity) function f:
  - argument: the size n of an instance of the problem (i.e., the length of its encoding)
  - result: the amount $f(n)$ of time/space needed in the worst-case to solve an instance of size n
- The asymptotic behaviour of the complexity function when n grows is considered.
- To abstract away from contingent issues (e.g., programming language, processor speed, etc.), we refer to an abstract computing model: Turing Machines (TMs).

Complexity classes

To achieve robustness wrt. encoding issues, usually one does not consider specific complexity functions f, but rather families $\mathcal{C}$ of complexity functions, giving rise to complexity classes.

Definition (Time/space complexity class $\mathcal{C}$)

A time/space complexity class $\mathcal{C}$ is the set of all problems P such that an instance of P of size n can be solved in time/space at most $C(n)$.

Note: Consider a (decision) problem P, and an encoding of the instances of P into strings over some alphabet Σ.
Once we fix such an encoding, the problem actually corresponds to a language L_P, namely the set of strings encoding those instances of the problem for which the answer is yes.
Hence, in the technical sense, a complexity class is actually a set of languages.

Outline

Reductions

To establish lower bounds on the complexity of problems, we make use of the notion of reduction:

Definition (Reduction)

A reduction from a problem P_1 to a problem P_2 is a function R (the reduction) from instances of P_1 to instances of P_2 such that:

1. R is efficiently computable (typically in logarithmic space), and
2. An instance I of P_1 has answer yes iff $R(I)$ has answer yes.

P_1 reduces to P_2 if there is a reduction R from P_1 to P_2.

Intuition: If P_1 reduces to P_2, then P2 is at least as difficult as P_1, since we can solve an instance I of P_1 by reducing it to the instance $R(I)$ of P_2 and then solve $R(I)$.

Hardness and completeness

Definition (Hardness)

A problem P is hard for a complexity class $\mathcal{C}$ if every problem in $\mathcal{C}$ can be reduced to P.

Definition (Completeness)

A problem P is complete for a complexity class $\mathcal{C}$ if

1. it is hard for $\mathcal{C}$, and
2. it belongs to $\mathcal{C}$

Intuitively, a problem that is complete for $\mathcal{C}$ is among the hardest problems in $\mathcal{C}$.

Outline

Tractability and intractability: PTime and NP

Definition (PTime)

Set of problems solvable in polynomial time by a deterministic TM.

- These problems are considered tractable, i.e., solvable for large inputs.
- Is a robust class (PTime computations compose).

Definition (NP)

Set of problems solvable in polynomial time by a non-deterministic TM.

- These problems are believed intractable, i.e., unsolvable for large inputs.
- The best known algorithms actually require exponential time.
- Corresponds to a large class of practical problems, for which the following type of algorithm can be used:
  1. Non-deterministically guess a possible solution of polynomial size.
  2. Check in polynomial time that the guessed solutions is good.

Complement of problemms in NP: coNP

Definition (coNP)

Set of problems whose complement is in NP, i.e., problems for which determining whether an instance admits a no answer is in NP.

- For problems whose complexity is characterized in terms of a non-deterministic Turing machine, solving it and solving its complement might be different
- A yes answer is returned if there exists a non-deterministic computation path of the TM that leads to acceptance
- A no answer requires that all no non-deterministic computation paths of the TM lead to rejection

coNP is believed to be different from both NP and PTime

Complexity classes above NP

Definition (PSpace)

Set of problems solvable in polynomial space by a deterministic TM.

- Polynomial space is “not really good”, since these problems may require exponential time.
- These problems are considered to be more difficult than NP problems.
- Practical algorithms and heuristics work less well than for NP problems.

Definition (ExpTime)

Set of problems solvable in exponential time by a deterministic TM.

- This is the first provably intractable complexity class.
- These problems are considered to be very difficult.

Definition (NExpTime)

Set of problems solvable in exponential time by a non-deterministic TM.

Complexity classes below PTime

Definition (LogSpace and NLogSpace)

Set of problems solvable in logarithmic space by a (non-)deterministic TM.

- Note: when measuring the space complexity, the size of the input does not count, and only the working memory (TM tape) is considered.
- Note 2: logarithmic space computations compose (this is not trivial).
- Correspond to reachability in undirected and directed graphs, respectively.

Definition (AC^0)

Set of problems solvable in constant time using a polynomial number of processors.

- These problems are solvable efficiently even for very large inputs.
- Corresponds to the complexity of model checking a fixed FO formula when the input is the model only.

Relationship between the complexity classes

The following relationships are known:

$$AC^0 \subsetneq LogSpace \subseteq NLogSpace \subseteq PTime \subseteq$$

$$\subseteq NP \subseteq PSpace \subseteq$$

$$\subseteq ExpTime \subseteq NExpTime$$

Moreover, we know that:

$$PTime \subsetneq ExpTime$$

Outline

Outline

What are Description Logics?

Description Logics (DLs) [Baader et al., 2003] are logics specifically designed to represent and reason on structured knowledge.

The domain of interest is composed of objects and is structured into:

- concepts, which correspond to classes, and denote sets of objects
- roles, which correspond to (binary) relationships, and denote binary relations on objects

The knowledge is asserted through so-called assertions, i.e., logical axioms.

Origins of Description Logics

Description Logics stem from early days knowledge representation formalisms (late '70s, early '80s):

- Semantic Networks: graph-based formalism, used to represent the meaning of sentences.
- Frame Systems: frames used to represent prototypical situations, antecedents of object-oriented formalisms.

Problems: no clear semantics, reasoning not well understood.

Description Logics (a.k.a. Concept Languages, Terminological Languages) developed starting in the mid '80s, with the aim of providing semantics and inference techniques to knowledge representation systems.

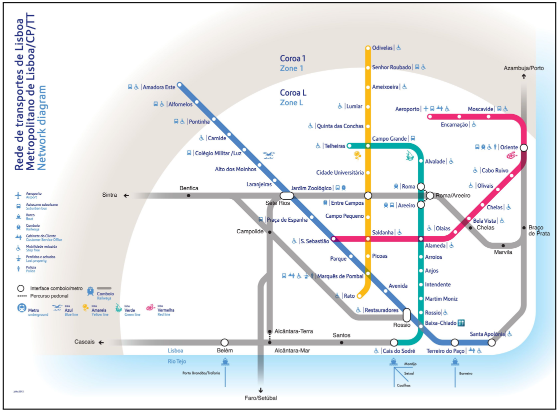
What are Description Logics about?

Abstractly, DLs allow one to predicate about labeled directed graphs:

- Vertexes represents real world objects.
- Vertexes's labels represents qualities of objects.
- Edges represents relations between (pairs of) objects.
- Edges' labels represents the types of relations between objects.

Every fragment of the world that can be abstractly represented in terms of a labeled directed graph is a good candidate for being represented by DLs.

What are Description Logics about? - Example 1



Exercise

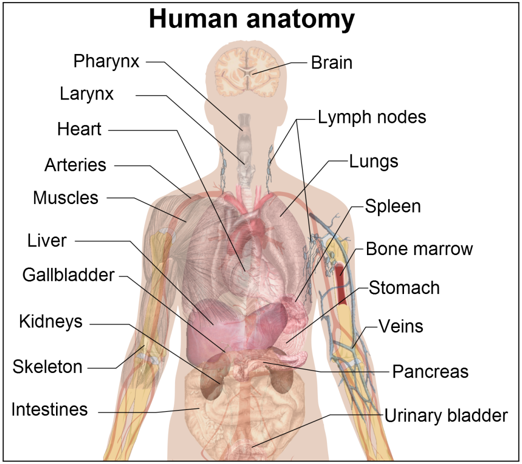
Represent Metro lines in Lisbon in a labelled directed graph.

What are Description Logics about? - Example 2

Exercise

Represent some aspects of Facebook as a labelled directed graph.

What are Description Logics about? - Example 3



Exercise

Represent some aspects of human anatomy as a labelled directed graph.

What are Description Logics about? - Example 4

Exercise

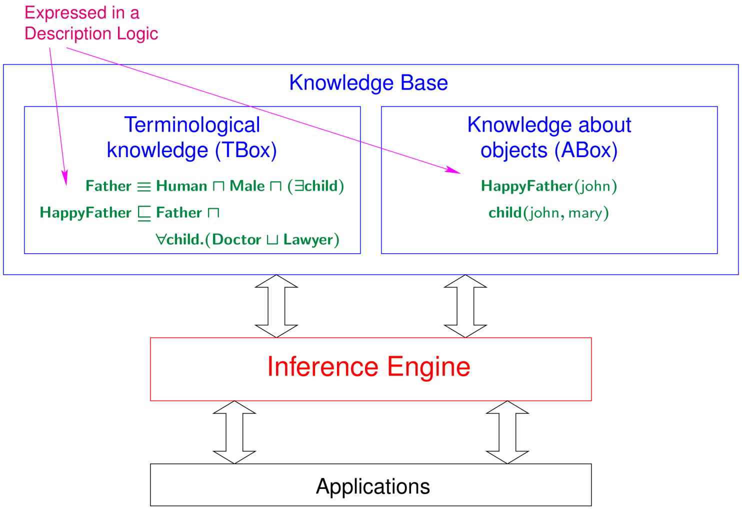
Represent some aspects of document classification as a labelled directed graph.

Ingredients of a Description Logic

A DL is characterized by:

1. A description language: how to form concepts and roles
   $Human \sqcap Male \sqcap \exists hasChild \sqcap \forall hasChild.(Doctor \sqcup Lawyer)$
2. A mechanism to specify knowledge about concepts and roles (i.e., a TBox)
   $\mathcal{T} = \{Father \equiv Human \sqcap Male \sqcap \exists hasChild,$
   $HappyFather \sqsubseteq Father \sqcap \forall hasChild.(Doctor \sqcup Lawyer)\}$
3. A mechanism to specify properties of objects (i.e., an ABox)
   $\mathcal{A} = \{HappyFather(john), hasChild(john, mary)\}$
4. A set of inference services: how to reason on a given KB
   $\mathcal{T} \models HappyFather \sqsubseteq \exists hasChild.(Doctor \sqcup Lawyer)$
   $\mathcal{T} \cup \mathcal{A} \models (Doctor \sqcup Lawyer)(mary)$

Architecture of a Description Logic system



Expressed in a Description Logic
Knowledge Base
Terminological knowledge (TBox)
$\textbf{Father} \equiv \textbf{Human} \sqcap \textbf{Male} \sqcap (\exists\textbf{child})$
$\textbf{HappyFather} \sqsubseteq \textbf{Father} \sqcap \forall\textbf{child}.(\textbf{Doctor} \sqcup \textbf{Lawyer})$
Knowledge about objects (ABox)
$\textbf{HappyFather}(\text{john})$
$\textbf{child}(\text{john}, \text{mary})$
Inference Engine
Applications

Description language

A description language provides the means for defining:

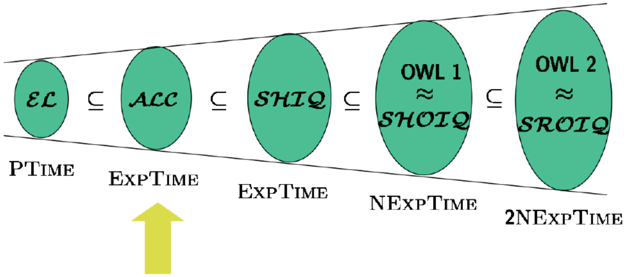
- concepts, corresponding to classes: interpreted as sets of objects;
- roles, corresponding to relationships: interpreted as binary relations on objects.

To define concepts and roles:

- We start from a (finite) alphabet of atomic concepts and atomic roles, i.e., simply names for concept and roles.
- Then, by applying specific constructors, we can build complex concepts and roles, starting from the atomic ones.

A description language is characterized by the set of constructs that are available for that.

Many description logics



Formal semantics of a description language

The formal semantics of DLs is given in terms of interpretations.

Definition (Interpretation)

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

- a nonempty set $\Delta^{\mathcal{I}}$, called the interpretation domain (of $\mathcal{I}$)
- an interpretation function $\cdot^{\mathcal{I}}$, which maps
  - each atomic concept A to a subset $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$
  - each atomic role P to a subset $P^{\mathcal{I}}$ of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$

The interpretation function is extended to complex concepts and roles according to their syntactic structure.

$\mathcal{AL}$ Concept constructors

Construct	Syntax	Example	Semantics
atomic concept	A	$Doctor$	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
atomic role	P	$hasChild$	$P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
atomic negation	$\neg A$	$\neg Doctor$	$\Delta^{\mathcal{I}} \backslash A^{\mathcal{I}}$
conjunction	$C \sqcap D$	$Human \sqcap Male$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
(unqual.) exist. res.	$\exists R$	$\exists hasChild$	$\{o \mid \exists o'.(o,o') \in R^{\mathcal{I}}\}$
value restriction	$\forall R.C$	$\forall hasChild.Male$	$\{o \mid \forall o'.(o,o') \in R^{\mathcal{I}} \to o' \in C^{\mathcal{I}}\}$
bottom	$\bot$		$\emptyset$

(C, D denote arbitrary concepts and R an arbitrary role)
The above constructs form the basic language $\mathcal{AL}$ of the family of $\mathcal{AL}$ languages.

Additional concept and role constructors

Construct	$\mathcal{AL}$	Syntax	Semantics
disjunction	$\mathcal{U}$	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
top		$\top$	$\Delta^{\mathcal{I}}$
qual. exist. res.	$\mathcal{E}$	$\exists R.C$	$\{o \mid \exists o'.(o,o') \in R^{\mathcal{I}} \wedge o' \in C^{\mathcal{I}}\}$
(full) negation	$\mathcal{C}$	$\neg C$	$\Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$
number	$\mathcal{N}$	$(\geq kR)$	$\{o \mid \sharp\{o' \mid (o,o') \in R^{\mathcal{I}}\} \geq k\}$
restriction		$(\leq kR)$	$\{o \mid \sharp\{o' \mid (o,o') \in R^{\mathcal{I}}\} \leq k\}$
qual. number	$\mathcal{Q}$	$(\geq kR.C)$	$\{o \mid \sharp\{o' \mid (o,o') \in R^{\mathcal{I}} \wedge o' \in C^{\mathcal{I}}\} \geq k\}$
restriction		$(\leq kR.C)$	$\{o \mid \sharp\{o' \mid (o,o') \in R^{\mathcal{I}} \wedge o' \in C^{\mathcal{I}}\} \leq k\}$
inverse role	$\mathcal{I}$	R^{-}	$\{(o,o') \mid (o',o) \in R^{\mathcal{I}}\}$
role closure	reg	R^{*}	$(R^{\mathcal{I}})^{*}$

Note: Many different DL constructs and their combinations have been investigated.

Further examples of DL constructs

- Disjunction: $\forall hasChild.(Doctor \sqcup Lawyer)$
- Qualified existential restriction: $\exists hasChild.Doctor$
- Full negation: $\neg(Doctor \sqcup Lawyer)$
- Number restrictions: $(\geq 2\ hasChild) \sqcup (\leq 1\ sibling)$
- Qualified number restrictions: $(\geq 2\ hasChild.Doctor)$
- Inverse role: $\forall hasChild^{-}.Doctor$
- Reflexive-transitive role closure: $\exists hasChild^{*}.Doctor$

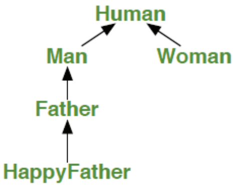
Reasoning on concept expressions

An interpretation $\mathcal{I}$ is a model of a concept C if $C^{\mathcal{I}} \neq \emptyset$.

Basic reasoning tasks

1. Concept satisfiability: does C admit a model?
2. Concept subsumption $C \sqsubseteq D$: does $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ hold for all interpretations $\mathcal{I}$?

Subsumption is used to build the concept hierarchy:



Exercise

Show that if DL is propositionally closed, then (1) and (2) are mutually reducible.

Complexity of reasoning on concept expressions

Complexity of concept satisfiability [Donini et al., 1997]

$\mathcal{AL}, \mathcal{ALN}$	PTime
$\mathcal{ALU}, \mathcal{ALUN}$	NP-complete
$\mathcal{ALE}$	coNP-complete
$\mathcal{ALC}, \mathcal{ALCN}, \mathcal{ALCI}, \mathcal{ALCQI}$	PSpace-complete

- Two sources of complexity:
  - union ($\mathcal{U}$) of type NP,
  - (qualified) existential quantification ($\mathcal{E}$) of type coNP.
- When they are combined, the complexity jumps to PSpace.
- Number restrictions ($\mathcal{N}$) do not add to the complexity.

Outline

Structural properties vs. asserted properties

We have seen how to build complex concept and roles expressions, which allow one to denote classes with a complex structure.

However, in order to represent real world domains, one needs the ability to assert properties of classes and relationships between them (e.g., as done in UML class diagrams).

The assertion of properties is done in DLs by means of an ontology (or knowledge base).

Description Logics ontology

Definition (Description Logics ontology (or knowledge base))

A Description Logics ontology (or knowledge base) is a pair $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox.

Definition (Description Logics TBox)

A Description Logic TBox consists of a set of assertions on concepts and roles:

- Inclusion assertions on concepts: $C_1 \sqsubseteq C_2$
- Inclusion assertions on roles: $R_1 \sqsubseteq R_2$
- Property assertions on (atomic) roles:
  - $(\textbf{transitive} \quad P)$ $(\textbf{symmetric} \quad P)$ $(\textbf{domain} \quad P \quad C)$
  - $(\textbf{functional} \quad P)$ $(\textbf{reflexive} \quad P)$ $(\textbf{range} \quad P \quad C) \ldots$

Definition (Description Logics ABox)

A Description Logics ABox consists of a set of assertions on individuals: (we use c_i to denote individuals)

- Membership assertions for concepts: $A(c)$
- Membership assertions for roles: $P(c_1, c_2)$
- Equality and distinctness assertions: $c_1 \approx c_2, \quad c_1 \not\approx c2$

Description Logics ontology - Example

Note: We use $C_1 \equiv C_2$ as an abbreviation for $C_1 \sqsubseteq C_2, \quad C_2 \sqsubseteq C_1$.

Example (TBox assertions)

- Inclusion assertions on concepts:

$$\begin{array}{rcl} Father & \equiv & Human \sqcap Male \sqcap \exists hasChild \\ HappyFather & \sqsubseteq & Father \sqcap \forall hasChild.(Doctor \sqcup Lawyer \sqcup HappyPerson) \\ HappyAnc & \sqsubseteq & \forall descendant.HappyFather \\ Teacher & \sqsubseteq & \neg Doctor \sqcap \neg Lawyer \end{array}$$

- Inclusion assertions on roles:
  $hasChild \sqsubseteq descendant \qquad hasFather \sqsubseteq hasChild^{-}$
- Property assertions on roles:
  (**transitive** descendant),(**reflexive** descendant),(**functional** hasFather)

Example (ABox membership assertions)

- $Teacher(mary), \quad hasFather(mary; john), \quad HappyAnc(john)$

Semantics of a Description Logics ontology

The semantics is given by specifying when an interpretation $\mathcal{I}$ satisfies an assertion α, denoted $\mathcal{I} \models \alpha$.

Definition (Satisfiability of TBox Assertions)

- $\mathcal{I} \models C_1 \sqsubseteq C_2$ if $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$
- $\mathcal{I} \models R_1 \sqsubseteq R_2$ if $R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$
- $\mathcal{I} \models (\mathbf{prop}\ P)$ if $P^{\mathcal{I}}$ is a relation that has the property **prop**.

(Note: domain and range assertions can be expressed by means of concept inclusion assertions.)

Definition (Satisfiability of ABox Assertions)

We first need to extend the interpretation function $\cdot^{\mathcal{I}}$, so that it maps each individual c to an element $c^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$.

- $\mathcal{I} \models A(c)$ if $c^{\mathcal{I}} \in A^{\mathcal{I}}$.
- $\mathcal{I} \models P(c_1, c_2)$ if $(c_1^{\mathcal{I}}, c_2^{\mathcal{I}}) \in P^{\mathcal{I}}$
- $\mathcal{I} \models c_1 \approx c_2$ if $c_1^{\mathcal{I}} = c_2^{\mathcal{I}}$
- $\mathcal{I} \models c_1 \not\approx c_2$ if $c_1^{\mathcal{I}} \neq c_2^{\mathcal{I}}$

Model of a Description Logics ontology

Definition (Model)

An interpretation $\mathcal{I}$ is a model of:

- an assertion α, if it satisfies α.
- a TBox $\mathcal{T}$, if it satisfies all assertions in $\mathcal{T}$.
- an ABox $\mathcal{A}$, if it satisfies all assertions in $\mathcal{A}$.
- an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ if it is a model of both $\mathcal{T}$ and $\mathcal{A}$.

Note: We use $\mathcal{I} \models \beta$ to denote that interpretation $\mathcal{I}$ is a model of β (where β stands for an assertion, TBox, ABox, or ontology).

Interpretation of individuals

We may make some assumptions on how individuals are interpreted.

Definition (Unique name assumption (UNA))

When c_1 and c_2 are two individuals such that $c_1 \neq c_2$, then $c_1^{\mathcal{I}} \neq c_2^{\mathcal{I}}$.

Note: When the UNA holds, equality and distinctness assertions are meaningless.

Definition (Standard name assumption (SNA))

The UNA holds, and moreover individuals are interpreted in the same way in all interpretations.
Hence, we may assume that $\Delta^{\mathcal{I}}$ contains the set of individuals, and that for each interpretation $\mathcal{I}$, we have that $c^{\mathcal{I}} = c$ (then, c is called a standard name).

Outline

Logical implication

The fundamental reasoning service from which all other ones can be easily derived is . . .

Definition (Logical implication)

An ontology $\mathcal{O}$ logically implies an assertion α , written $\mathcal{O} \models \alpha$ if α is satisfied by all models of $\mathcal{O}$.

We can provide an analogous definition for a TBox $\mathcal{T}$ instead of an ontology $\mathcal{O}$.

TBox reasoning

- TBox Satisfiability: $\mathcal{T}$ is satisfiable if it admits at least one model.
- Concept Satisfiability: C is satisfiable wrt. $\mathcal{T}$ if there is a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}}$ is not empty, i.e., $\mathcal{T} \not\models C \equiv \bot$
- Subsumption: C_1 is subsumed by C_2 wrt. $\mathcal{T}$ if for every model $\mathcal{I}$ of $\mathcal{T}$ we have $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$, i.e., $\mathcal{T} \models C_1 \sqsubseteq C_2$
- Equivalence: C_1 and C_2 are equivalent wrt. $\mathcal{T}$ if for every model $\mathcal{I}$ of $\mathcal{T}$ we have $C_1^{\mathcal{I}} = C_2^{\mathcal{I}}$, i.e., $\mathcal{T} \models C_1 \equiv C_2$
- Disjointness: C_1 and C_2 are disjoint wrt. $\mathcal{T}$ if for every model $\mathcal{I}$ of $\mathcal{T}$ we have $C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} = \emptyset$, i.e., $\mathcal{T} \models C_1 \sqcap C_2 \equiv \bot$
- Functionality implication: A functionality assertion $(\textbf{funct } R)$ is logically implied by $\mathcal{T}$ if for every model $\mathcal{I}$ of $\mathcal{T}$, we have that $(o, o_1) \in R^{\mathcal{I}}$ and $(o, o_2) \in R^{\mathcal{I}}$ implies $o_1 = o_2$, i.e., $\mathcal{T} \models (\textbf{funct } R)$

Note: Analogous definitions hold for role satisfiability, subsumption, equivalence, and disjointness.

Reasoning over an ontology

- Ontology Satisfiability: Verify whether an ontology $\mathcal{O}$ is satisfiable, i.e., whether $\mathcal{O}$ admits at least one model.
- Concept Instance Checking: Verify whether an individual c is an instance of a concept C in every model of $\mathcal{O}$, i.e., whether $\mathcal{O} \models C(c)$.
- Role Instance Checking: Verify whether a pair (c_1, c_2) of individuals is an instance of a role R in every model of $\mathcal{O}$, i.e., whether $\mathcal{O} \models R(c_1, c_2)$.

Reasoning in Description Logics - Example

Example (TBox)

- Inclusion assertions on concepts:

$$\begin{array}{rcl} Father & \equiv & Human \sqcap Male \sqcap \exists hasChild \\ HappyFather & \sqsubseteq & Father \sqcap \forall hasChild.(Doctor \sqcup Lawyer \sqcup HappyPerson) \\ HappyAnc & \sqsubseteq & \forall descendant.HappyFather \\ Teacher & \sqsubseteq & \neg Doctor \sqcap \neg Lawyer \end{array}$$

- Inclusion assertions on roles:
  $hasChild \sqsubseteq descendant$ $hasFather \sqsubseteq hasChild^-$
- Property assertions on roles:
  (**transitive** descendant), (**reflexive** descendant), (**functional** hasFather)

The above TBox logically implies: $HappyAnc \sqsubseteq Father.$

Example (ABox)

- Membership assertions:
  $Teacher(mary)$, $hasFather(mary; john)$, $HappyAnc(john)$

The above TBox and ABox logically imply: $HappyPerson(mary)$

Relationship among TBox reasoning tasks

The TBox reasoning tasks are mutually reducible to each other, provided the description language is propositionally closed:

Theorem (TBox satisfiability to concept satisfiability to concept non-subsumption)

$\mathcal{T}$ *satisfiable* *iff* $\mathcal{T} \not\models \top \equiv \bot$ *iff* *not* $\mathcal{T} \models \top \sqsubseteq \bot$
(i.e., $\top$ *satisfiable w.r.t.* $\mathcal{T}$ *)*

Theorem (Concept subsumption to concept unsatisfiability)

$\mathcal{T} \models C_1 \sqsubseteq C_2$ *iff* $\mathcal{T} \models C_1 \sqcap \neg C_2 \equiv \bot$
(i.e., $\models C_1 \sqcap \neg C_2$ *unsatisfiable w.r.t.* $\mathcal{T}$ *)*

Theorem (Concept satisfiability to TBox satisfiability)

$\mathcal{T} \not\models C \equiv \bot$ *iff* $\mathcal{T} \cup \{\top \sqsubseteq \exists P_{new} \sqcap \forall P_{new}.C\}$ *satisfiable*
(where P_{new} *is a new atomic role)*

Relationship among reasoning tasks

TBox reasoning can be reduced to reasoning over an ontology:

Theorem (Concept satisfiability to ontology satisfiability)

C satisfiable wrt $\mathcal{T}$ iff $\langle \mathcal{T} \cup \{A_{new} \sqsubseteq C\}, \{A(c_{new})\}\rangle$ is satisfiable (where A_{new} is a new atomic concept and c_{new} is a new individual)

Exercise

Show mutual reductions between the remaining (TBox and ontology) reasoning tasks.

Internalization of the TBox

- In some (very expressive) DLs, it is possible to reduce reasoning wrt. a TBox to reasoning over concept expressions only, i.e., the whole TBox can be internalized into a single concept.
- Whether this is possible depends on the available role and concept constructors, and the details differ for each DL.

Complexity of reasoning over DL ontologies

Reasoning over DL ontologies is much more complex than reasoning over concept expressions:

Bad news:

- without restrictions on the form of TBox assertions, reasoning over DL ontologies is already ExpTime-hard, even for very simple DLs (see, e.g.,[Donini, 2003]).

Good news:

- We can add a lot of expressivity (i.e., essentially all DL constructs seen so far), while still staying within the ExpTime upper bound.
- There are DL reasoners that perform reasonably well in practice for such DLs (e.g., Konclude, Hermit, Racer, Pellet, Fact++, . . .)
- For certain application domains, well-designed fragments of DLs can be used for which reasoning is polynomial – OWL profiles.