Reasoning and Query Answering in Description Logics

Magdalena Ortiz, Mantas Šimkus

Vienna University of Technology

8th Reasoning Web Summer School, 3–8 September 2012

Ontologies

An ontology is a conceptual description of a domain

- that can be expressed in different formalisms
  - The OWL Web Ontology Languages
  - Description Logics
  - F-Logic, RDFS
  - Datalog and related rule-based formalisms
- and can be used for a vast range of purposes
  - in the Semantic Web, to allow automated agents to understand shared web resources
  - in Google's new *Knowledge Graph*, for improved Web search and more informative results
  - in medical and life sciences, to support the effective clinical recording of data in order to improve patient care
  - in organizations, to provide a coherent and unified conceptual view of possibly distributed, redundant, and incoherent data sources, and to allow access to them
  - ...

Ontologies, Data, and Description Logics

In this tutorial, we focus on ontologies expressed in
Description Logics (DLs)
and on their application for data access

The tutorial has two parts:

1. A brief introduction to DLs
   - DL basics
   - reasoning problems
   - computational aspects
2. An overview of the setting where DLs are used for data access
   - the query answering problem in DLs
   - reasoning techniques
   - computational aspects

Outline

DLs as a Family of Logics

Most DLs are fragments of classical first order logic (FOL), which usually

- are function-free and use only two variables (and maybe some additions like counting quantifiers)
- allow only the restricted guarded quantification
- are closely related to modal logic and extensions

In contrast to FOL, Description Logics:

- are decidable
- their syntax is specially well-suited for describing structured knowledge
  - no explicit variables
  - Representation at the *predicate level*
- may provide 'abbreviations' for common KR constructs cumbersome to write in FOL

DLs as Computational Logics

- Description Logics are a hierarchy of decidable logics with increasing expressive power and computational complexity

  DLs range from

  - Lightweight DLs that support efficient inference but have quite limited expressiveness
  - Very expressive DLs that allow for a flexible representation of very complex domains, at the price of higher complexity of inference

  A hallmark of DLs is the study of the trade-off between expressive power and computational complexity

- This supports a problem-oriented choice of the logic!

Vocabulary

We start from a *vocabulary* with three kinds of elements:

concept names: atomic classes, unary predicates

female, student, course, frog

role names: atomic relations, binary predicates

hasChild, likes, isEnrolledIn, hasColor

individuals: constants

zeus, heracles, kermit, cecilia

Concept and role constructors

Using the available constructors, we build more complex concept and roles

- Concept constructors:

$\mathsf{female} \sqcap \mathsf{child}$	$\mathsf{plane} \sqcup \mathsf{bird}$
$(\mathsf{fruit} \sqcup \mathsf{vegetable}) \sqcap \neg\mathsf{rotten}$	$\geqslant 2\,\mathsf{hasChild}.\mathsf{female}$
$\mathsf{frog} \sqcap \forall \mathsf{hasColor}.\mathsf{green}$	$\exists \mathsf{hasParent}.\{\mathrm{zeus}\}$

- Role constructors:

$\mathsf{isEnrolledIn} \cup \mathsf{attends}$	$\mathsf{isRelatedTo} \cap \neg\mathsf{likes}$
$(\mathsf{hasParent} \cup \mathsf{hasChild})^*$	

The set of available constructors is different in each DL

The ABox

In the ABox we write

- Concept membership assertions, like Hero(perseus)
  (Often written perseus : Hero)
- Role membership assertions, like hasParent(perseus, zeus)
  (Often written (perseus, zeus) : hasParent)

Hence, an ABox may look like

$$\begin{array}{l} \mathsf{hasParent}(\mathrm{heracles}, \mathrm{zeus}) \\ \mathsf{hasParent}(\mathrm{heracles}, \mathrm{alcmene}) \\ \mathsf{hasParent}(\mathrm{perseus}, \mathrm{zeus}) \\ \mathsf{Deity}(\mathrm{zeus}) \\ \mathsf{Hero}(\mathrm{perseus}) \end{array}$$

- Intuitively, it lists facts that are known to be true
- Can be seen as a partial description of the world

The TBox

The TBox is a set of terminological axioms that state how concept or roles are related to each other

- Two main kinds of terminological axioms:
  - General concept inclusions (GCIs): $C \sqsubseteq D$
  - Definitions: $A \equiv D$, where A is a concept name
    It can be seen as a shortcut for $A \sqsubseteq D$ and $D \sqsubseteq A$
- Sometimes, distinction between:

Terminology
- Set of definitions
- For every atomic concept A, there is only one definition whose left hand side is A

General TBox
- Set of GCIs
- Every set of definitions can be written as a set of GCIs

The TBox (cont'd)

We usually consider general TBoxes, which may look as follows

$$\begin{array}{rcl} \mathsf{Hero} & \sqsubseteq & \exists \mathsf{hasAncestor}.\mathsf{Deity} \\ \mathsf{Deity} & \sqsubseteq & \forall \mathsf{hasAncestor}.\mathsf{Deity} \\ \forall \mathsf{hasParent}.\mathsf{Mortal} & \sqsubseteq & \mathsf{Mortal} \\ \top & \sqsubseteq & \mathsf{Mortal} \sqcup \mathsf{Deity} \sqcup \mathsf{Hero} \\ \top & \sqsubseteq & \exists \mathsf{hasParent}.\mathsf{Male} \sqcap \exists \mathsf{hasParent}.\mathsf{Female} \end{array}$$

$\top$ is a special concept that informally means 'everybody' (more later)

- Intuitively, it describes constraints on every object
- It can imply the existence of more objects

The Meaning of KBs (semantics)

The semantics is given in terms of interpretations, similar to the ones used in FOL

An interpretation has:

1. A non-empty domain
2. An interpretation function
   - It gives meaning to the basic symbols in the vocabulary
   - It is extended to complex concept and roles, following the rules that define the different constructors

An interpretation is called a model if it satisfies all the assertions in the ABox and all the axioms in the TBox

Syntax of $\mathcal{ALC}$ Concepts

Our basic vocabulary contains:

- a countable set N_C of concepts names,
- a countable set N_R of role names (or just roles), and
- a countable set N_I of individual names (or just individuals)

$\mathcal{ALC}$ concepts are defined inductively:

- Every concept name $A \in \mathsf{N}_C$ is a concept
- $\top$ and $\bot$ are concepts
- If C is a concept, then $\neg C$ is a concept
- If C_1 and C_2 are concepts, then $C_1 \sqcap C_2$ and $C_1 \sqcup C_2$ are concepts
- If $R \in \mathsf{N}_R$ is a role and C is a concept, then $\forall R.C$ and $\exists R.C$ are concepts

Note: In $\mathcal{ALC}$ we only have *concept constructors*

Syntax - $\mathcal{ALC}$ Knowledge Bases

An $\mathcal{ALC}$ Knowledge Base is a pair $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ where:

- The TBox $\mathcal{T}$ is a finite set of GCIs ($C_1 \sqsubseteq C_2$), and
- The ABox $\mathcal{A}$ is a finite set of (concept and role) membership assertions ($C(a), R(a,b)$)

$$\mathcal{T} = \{ \quad \top \sqsubseteq \mathsf{Mortal} \sqcup \mathsf{Deity} \sqcup \mathsf{Hero}, \quad \top \sqsubseteq \exists \mathsf{hasParent}.\mathsf{Male} \sqcap \exists \mathsf{hasParent}.\mathsf{Female}, \quad \forall \mathsf{hasParent}.\mathsf{Mortal} \sqsubseteq \mathsf{Mortal}, \quad \mathsf{Hero} \sqsubseteq \exists \mathsf{hasAncestor}.\mathsf{Deity}, \quad \mathsf{Deity} \sqsubseteq \forall \mathsf{hasAncestor}.\mathsf{Deity} \quad \}$$

$$\mathcal{A} = \{ \quad \mathsf{hasParent}(\mathrm{heracles}, \mathrm{zeus}), \quad \mathsf{hasParent}(\mathrm{heracles}, \mathrm{alcmene}), \quad \mathsf{hasParent}(\mathrm{perseus}, \mathrm{zeus}), \quad \mathsf{Deity}(\mathrm{zeus}), \quad \mathsf{Hero}(\mathrm{perseus}) \quad \}$$

The Semantics of $\mathcal{ALC}$

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

- a non-empty set $\Delta^{\mathcal{I}}$ called domain
- an interpretation function $\cdot^{\mathcal{I}}$

The interpretation function $\cdot^{\mathcal{I}}$ maps

- every concept C to a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, i.e., $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
- every role R to a set $R^{\mathcal{I}}$ of pairs of elements from $\Delta^{\mathcal{I}}$, i.e., a *binary relation* $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
- every individual a to an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$

Example - Interpretation of atomic symbols

1. We consider a domain: $\{z, p, m, a, s, h, n\}$
2. Each individuals is interpreted as one element

   $\mathrm{zeus}^{\mathcal{I}} = z \quad \mathrm{perseus}^{\mathcal{I}} = p \quad \mathrm{alcmene}^{\mathcal{I}} = a \ \ldots$
3. Concept names are interpreted as sets of elements

   $\mathsf{Hero}^{\mathcal{I}} = \{p, h\} \quad \mathsf{Deity}^{\mathcal{I}} = \{z\} \quad \mathsf{Mortal}^{\mathcal{I}} = \{s, a\} \ \ldots$
4. Roles are interpreted as sets of pairs

   $$\mathsf{hasParent}^{\mathcal{I}} = \{(h,z), (h,a), (p,z)\}$$
   $$\mathsf{loves}^{\mathcal{I}} = \{(m,n)\} \ \ldots$$

The Semantics of $\mathcal{ALC}$ (cont'd)

The interpretation function is extended to all concepts:

Constructor	Syntax	Semantics
top/verum	$\top$	$\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$
bottom/falsum	$\bot$	$\bot^{\mathcal{I}} = \emptyset$
negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjunction	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
disjunction	$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
universal rest.	$\forall R.C$	$\{d_1 \mid \forall d_2 \in \Delta^{\mathcal{I}}.(R^{\mathcal{I}}(d_1, d_2) \to d_2 \in C^{\mathcal{I}})\}$
existential rest.	$\exists R.C$	$\{d_1 \mid \exists d_2 \in \Delta^{\mathcal{I}}.(R^{\mathcal{I}}(d_1, d_2) \land d_2 \in C^{\mathcal{I}})\}$

Example - Interpretation of complex concepts

1. We consider a domain: $\{z, p, m, a, s, h, n\}$
2. Each individuals is interpreted as one element:
   $\text{zeus}^{\mathcal{I}} = z \quad \text{perseus}^{\mathcal{I}} = p \quad \text{alcmene}^{\mathcal{I}} = a \ \ldots$
3. Concepts are interpreted as sets of elements.
   $\text{Hero}^{\mathcal{I}} = \{p, h\} \quad \text{Deity}^{\mathcal{I}} = \{z\} \quad \text{Mortal}^{\mathcal{I}} = \{s, a\} \ \ldots$
4. Roles are interpreted as sets of pairs
   $$\begin{aligned} \text{hasParent}^{\mathcal{I}} &= \{(h, z), (h, a), (p, z)\} \\ \text{loves}^{\mathcal{I}} &= \{(m, n)\} \ \ldots \end{aligned}$$
5. The atomic expressions fix the meaning of all the complex ones, e.g.,
   $$\begin{aligned} (\text{Hero} \sqcup \text{Mortal})^{\mathcal{I}} &= \{p, h, s, a\} \\ (\exists \text{hasParent}.\text{Mortal})^{\mathcal{I}} &= \{h\} \\ (\neg \text{Mortal})^{\mathcal{I}} &= \{z, p, m, h, n\} \\ (\forall \text{hasParent}.\text{Deity})^{\mathcal{I}} &= \{z, p, m, a, s, n\} \end{aligned}$$

TBox, ABox and KB Satisfaction

For an interpretation $\mathcal{I}$, we say that

- $\mathcal{I}$ satisfies a GCI $C_1 \sqsubseteq C_2$ if $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$
- $\mathcal{I}$ satisfies a TBox $\mathcal{T}$ if it satisfies every GCI in $\mathcal{T}$
- $\mathcal{I}$ satisfies a concept membership assertion $C(a)$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$
  $\mathcal{I}$ satisfies a role membership assertion $R(a, b)$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$
- $\mathcal{I}$ satisfies a ABox $\mathcal{A}$ if it satisfies every membership assertion in $\mathcal{A}$

An interpretation $\mathcal{I}$ is called a model of a knowledge base $(\mathcal{T}, \mathcal{A})$ if it satisfies the TBox $\mathcal{T}$ and the ABox $\mathcal{A}$

Example - TBox, ABox and KB Satisfaction

Let us take an interpretation $\mathcal{I}$ with domain $\Delta^{\mathcal{I}} = \{z, p, a, h\}$ and

$$\begin{aligned} \text{zeus}^{\mathcal{I}} &= z & \text{Hero}^{\mathcal{I}} &= \{p, h\} \\ \text{perseus}^{\mathcal{I}} &= p & \text{Deity}^{\mathcal{I}} &= \{z\} \\ \text{alcmene}^{\mathcal{I}} &= a & \text{Mortal}^{\mathcal{I}} &= \{a\} \\ \text{heracles}^{\mathcal{I}} &= h & \text{Male}^{\mathcal{I}} &= \{z, p, h\} \\ & & \text{Female}^{\mathcal{I}} &= \{a\} \\ & & \text{hasParent}^{\mathcal{I}} &= \{(h, z), (h, a), (p, z)\} \\ & & \text{hasAncestor}^{\mathcal{I}} &= \{(h, z), (h, a), (p, z)\} \end{aligned}$$

$\mathcal{I}$ satisfies the ABox: $\mathcal{A} = \{$ hasParent(heracles, zeus), hasParent(heracles, alcmene), hasParent(perseus, zeus), Deity(zeus), Hero(perseus) $\}$

Example - TBox, ABox and KB Satisfaction

Let us take an interpretation $\mathcal{I}$ with domain $\Delta^{\mathcal{I}} = \{z, p, a, h\}$ and

$$\begin{array}{rcrcl} \mathrm{zeus}^{\mathcal{I}} = z & & \mathsf{Hero}^{\mathcal{I}} & = & \{p, h\} \\ \mathrm{perseus}^{\mathcal{I}} = p & & \mathsf{Deity}^{\mathcal{I}} & = & \{z\} \\ \mathrm{alcmene}^{\mathcal{I}} = a & & \mathsf{Mortal}^{\mathcal{I}} & = & \{a\} \\ \mathrm{heracles}^{\mathcal{I}} = h & & \mathsf{Male}^{\mathcal{I}} & = & \{z, p, h\} \\ & & \mathsf{Female}^{\mathcal{I}} & = & \{a\} \\ & & \mathsf{hasParent}^{\mathcal{I}} & = & \{(h, z), (h, a), (p, z)\} \\ & & \mathsf{hasAncestor}^{\mathcal{I}} & = & \{(h, z), (h, a), (p, z)\} \end{array}$$

What about the TBox?

$$\mathcal{T} = \{ \begin{array}{r} \mathsf{Hero} \sqsubseteq \exists \mathsf{hasAncestor}.\mathsf{Deity}, \\ \mathsf{Deity} \sqsubseteq \forall \mathsf{hasAncestor}.\mathsf{Deity} \\ \forall \mathsf{hasParent}.\mathsf{Mortal} \sqsubseteq \mathsf{Mortal}\ , \\ \top \sqsubseteq \mathsf{Mortal} \sqcup \mathsf{Deity} \sqcup \mathsf{Hero}, \\ \top \sqsubseteq \exists \mathsf{hasParent}.\mathsf{Male} \sqcap \exists \mathsf{hasParent}.\mathsf{Female} \end{array} \},$$

Is it a model of the KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$?

Expressive DLs

The term expressive DLs is used informally
to refer to $\mathcal{ALC}$ and its extensions

The most common ways to obtain an extension of $\mathcal{ALC}$ are:

- Adding other concept constructors
  For example, number restrictions
- Adding role constructors
  For example, inverses
- Allowing, apart from GCIs, other kind of axioms in the TBox
  For example, inclusions between roles

Concept Constructors

Some common concept constructors are:

- Different kinds of number restrictions, which informally allow us to count the number of objects related by a certain role
- **Nominals**, aka *one-of* $\mathcal{O}$
- **Self concepts**, a more recent construct

Number restrictions

They can be of different kinds:

Qualified number restrictions	$\mathcal{Q}$	$\geqslant 2\,\mathsf{hasChild}.\mathsf{Male}$ $\leqslant 2\,\mathsf{hasChild}.\mathsf{Male}$
(Unqualified) number restrictions	$\mathcal{N}$	$\geqslant 3\,\mathsf{hasChild}$ $\leqslant 3\,\mathsf{hasChild}$ equiv. to $\geqslant 3\,\mathsf{hasChild}.\top$ $\leqslant 3\,\mathsf{hasChild}.\top$
Functionality restrictions	$\mathcal{F}$	$\leqslant 1\,\mathsf{hasFather}$ also written funct(hasFather)

Qualified Number Restrictions

Syntax If $n \geq 1$, R is a role and C is a concept, then $\leqslant n\, R.C$, $\geqslant n\, R.C$ are concepts

Semantics
$\geqslant n\, R.C = \{d_1 | \#(\{d_2 | (d_1, d_2) \in R^{\mathcal{I}} \wedge d_2 \in C^{\mathcal{I}}\}) \geq n\}$
$\leqslant n\, R.C = \{d_1 | \#(\{d_2 | (d_1, d_2) \in R^{\mathcal{I}} \wedge d_2 \in C^{\mathcal{I}}\}) \leq n\}$

Nominals $\mathcal{O}$

- allow us to build a concept from a set of individuals $\{a_1, \ldots, a_n\}$
- stands for the set of objects that interpret the individuals $a_1, \ldots, a_n$

For example, we can define:

$\{\text{Gaia}, \text{Chaos}, \text{Chronos}, \text{Ananke}\}$	the primordial gods
$\{\text{Austria}, \text{Belgium}, \ldots, \text{UK}\}$	the countries of the EU

Nominals

Syntax	If $a_1, \ldots, a_n$ are individuals, then $\{a_1, \ldots, a_n\}$ is a concept
Semantics	$\{a_1, \ldots, a_n\}^\mathcal{I} = \{a_1{}^\mathcal{I}, \ldots, a_n{}^\mathcal{I}\}$

Self concepts

- allow us to build a concept $\exists R.$Self from a role R
- stands for the set of objects that are related via R to itself

For example, we can define:

$\exists$loves.Self	narcissists
$\exists$hasAncestor.Self	individuals that are their own ancestors

Self

Syntax	If R is a role, then $\exists R.$Self is a concept
Semantics	$\exists R.\text{Self}^\mathcal{I} = \{d \mid (d, d) \in R^\mathcal{I}\}$

Some Role Constructors

- The inverse $\mathcal{I}$ is the most popular role constructor

  hasParent$^-$

- Sometimes, Boolean role constructors are considered

Union (role disjunction)	hasFriend $\cup$ hasRelative
Intersection (role conjunction)	hasFriend $\cap$ hasRelative
Negation	$\neg$hasFriend
Difference	hasRelative $\setminus$ hasFriend

  - $\mathcal{B}$ stands for intersection, union, and negation
  - b stands for intersection, union, and difference

- Other constructors

Role composition		hasParent $\circ$ hasSibling
Regular expressions	*reg*	hasParent* $\circ$ (hasParent$^-$)*

Summary of Main constructors

	Constructor	Syntax	Semantics
Concept constructors		(R is a role, C a concept, a_i individuals, $n \in \mathbb{N}$)	
$\mathcal{N}$	NR	$\geq n\, R$	$\{d_1 \mid \#(\{d_2 \mid (d_1, d_2) \in R^\mathcal{I}\}) \geq n\}$
		$\leq n\, R$	$\{d_1 \mid \#(\{d_2 \mid (d_1, d_2) \in R^\mathcal{I}\}) \leq n\}$
$\mathcal{Q}$	QNR	$\geqslant n\, R.C$	$\{d_1 \mid \#(\{d_2 \mid (d_1, d_2) \in R^\mathcal{I} \wedge d_2 \in C^\mathcal{I}\}) \geq n\}$
		$\leqslant n\, R.C$	$\{d_1 \mid \#(\{d_2 \mid (d_1, d_2) \in R^\mathcal{I} \wedge d_2 \in C^\mathcal{I}\}) \leq n\}$
$\mathcal{O}$	nominals	$\{a_1, \ldots, a_n\}$	$\{a_1^\mathcal{I}, \ldots, a_n^\mathcal{I}\}$
	self	$\exists R.$Self	$\{d \mid (d, d) \in R^\mathcal{I}\}$
Role constructors			(R, R_i, are roles)
$\mathcal{I}$	inverse	R^-	$\{(d_2, d_1) \mid (d_1, d_2) \in R^\mathcal{I}\}$
(*reg*)	composition	$R_1 \circ R_2$	$\{(d_1, d_3) \mid (d_1, d_2) \in R_1^\mathcal{I} \wedge (d_2, d_3) \in R_2^\mathcal{I}\}$
(*reg*)	refl. trans. closure	R^*	$(\Delta^\mathcal{I} \times \Delta^\mathcal{I}) \cup \{(d_1, d_n) \mid (d_i, d_{i+1}) \in R^\mathcal{I}, 1 \leq i < n\}$
$(\mathcal{B}, b)$	intersection	$R_1 \cap R_2$	$R_1^\mathcal{I} \cap R_2^\mathcal{I}$
$(\mathcal{B}, b, reg)$	union	$R_1 \cup R_2$	$R_1^\mathcal{I} \cup R_2^\mathcal{I}$
$(\mathcal{B})$	negation	$\neg R$	$\Delta^\mathcal{I} \times \Delta^\mathcal{I} \setminus R^\mathcal{I}$
(b)	difference	$R_1 \setminus R_2$	$R_1^\mathcal{I} \setminus R_2^\mathcal{I}$

Role Axioms

We can also extend $\mathcal{ALC}$ by allowing terminological axioms that refer to roles and their relations

- Role inclusions $\mathcal{H}$ are expressions of the form

$$R \sqsubseteq S$$

  for roles R and S

  A set of role inclusions is called a role hierarchy or RBox

- Transitivity axioms are expressions of the form

$$\mathsf{trans}(R)$$

  for a role R, asserting that R is transitive.

  The extension of $\mathcal{ALC}$ with transitivity axioms is denoted $\mathcal{S}$

Role Axioms (cont'd)

Semantically, in a model of a KB

- For every role inclusion $R \sqsubseteq S$, $R^\mathcal{I} \subseteq S^\mathcal{I}$ must hold
- For each trans(R), $R^\mathcal{I}$ must be transitively closed: if $(d_1, d_2) \in R^\mathcal{I}$ and $(d_2, d_3) \in R^\mathcal{I}$, then $(d_1, d_3) \in R^\mathcal{I}$.

In $\mathcal{SH}$ we can express, for example:
hasParent $\sqsubseteq$ hasAncestor trans(hasAncestor)

Some Expressive DLs

Some examples of expressive DLs are:

$\mathcal{ALCHOIQ}b$	$\mathcal{SHOINB}$	$\mathcal{SHOIQ}$
$\mathcal{SHIQ}$	$\mathcal{SHOQ}$	$\mathcal{SHIO}$
$\mathcal{ALC}_{reg}$	$\mathcal{ALCI}_{reg}$	$\mathcal{ALCIF}_{reg}$

- $\mathcal{SHIQ}$ and $\mathcal{SHOIQ}$ are closely related to the OWL languages (more later)
- Widely studied, supported by many existing reasoners

The $\mathcal{SR}$ family

The new OWL 2 standard is based on the $\mathcal{SR}$ family of DLs, which is an extension of the $\mathcal{SH}$ family.

- It supports Self concepts
- It has complex role inclusion axioms:

$$\begin{aligned} \mathsf{hasParent} &\sqsubseteq \mathsf{hasAncestor} \\ \mathsf{hasAncestor} \circ \mathsf{hasAncestor} &\sqsubseteq \mathsf{hasAncestor} \\ \mathsf{hasParent} \circ \mathsf{hasSibling} &\sqsubseteq \mathsf{hasUncle} \end{aligned}$$

  The implications between roles must satisfy certain syntactic restrictions

  - Strong restrictions on cyclic dependencies
  - Witnessed by an order on the roles
  - Ensures that the role inclusions form a regular grammar

The $\mathcal{SR}$ family (cont'd)

- It allows for special axioms to impose properties on roles

Reflexivity	Ref(R)
Irreflexivity	Irr(R)
Disjointness	Disj(R, S)
Symmetry	Sym(R)

- In the $\mathcal{SR}$ family, KBs are defined as a triple $(\mathcal{T}, \mathcal{R}, \mathcal{A})$, where $\mathcal{R}$ is an RBox that contains all the role axioms
- The most prominent $\mathcal{SR}$ logics are $\mathcal{SRIQ}$ and $\mathcal{SROIQ}$

The DLs underlying OWL

The OWL standards proposed by W3C are based on these logics:

OWL Variant	DL counterpart
OWL 1 - Lite	$\mathcal{SHIF}$
OWL 1 - DL	$\mathcal{SHOIQ}$
OWL 2	$\mathcal{SROIQ}$

- Additionally, the OWL standards support *data types*, which are captured by concrete domains in DLs.
  We do not consider them in this course.

Terminological Reasoning Services

Traditional reasoning services are tailored for knowledge engineering.

1. Concept (or class) subsumption
   Input: concepts C and D (and possibly a TBox $\mathcal{T}$ / a KB $\mathcal{K}$)
   Problem: Is C subsumed by D (w.r.t. $\mathcal{T}$ / $\mathcal{K}$)?
   - C is subsumed by D (w.r.t. $\mathcal{T}$ / $\mathcal{K}$) if $C^\mathcal{I} \subseteq D^\mathcal{I}$ in every interpretation $\mathcal{I}$ (that is a model of $\mathcal{T}$ / $\mathcal{K}$)

   Does my ontology ensure that everyone who has a parent that is not mortal is either a hero or a deity?

2. Concept satisfiability
   Input: concept C (and possibly a TBox $\mathcal{T}$ / a KB $\mathcal{K}$)
   Problem: Is C satisfiable (w.r.t. $\mathcal{T}$ / $\mathcal{K}$)?
   - C is satisfiable (w.r.t. $\mathcal{T}$ / $\mathcal{K}$) if $C^\mathcal{I} \neq \emptyset$ for some interpretation $\mathcal{I}$ (that is a model of T)

   Is it possible that someone has more than two parents?

Terminological Reasoning Services (cont'd)

3. Concept disjointness
   Input: concepts C and D (and possibly a TBox $\mathcal{T}$ / a KB $\mathcal{K}$)
   Problem: Are C and D disjoint (w.r.t. $\mathcal{T}$ / $\mathcal{K}$)?
   - C and D are disjoint (w.r.t. $\mathcal{T}$ / $\mathcal{K}$) if $C^\mathcal{I} \cap D^\mathcal{I} = \emptyset$ in every interpretation $\mathcal{I}$ (that is a model of T / $\mathcal{K}$)

   Does my ontology ensure that everyone who is a hero does not have two mortal parents?

For other services we give an informal description:

4. Least common subsumer: Given concepts C and D, find the most specific concept that subsumes them both
5. Classification: Find all subsumptions between the concept names occurring in a given ontology

These services are usually called terminological reasoning services

Reasoning about Instances

Other services focus on the individuals occurring in the ABox.

6. Instance checking
   Input: individual a, concept C, and a KB $\mathcal{K}$
   Problem: Is a an instance of C in $\mathcal{K}$?
   - a is an instance of C in $\mathcal{K}$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{K}$

   Is Heracles a hero who has a mortal mother?

7. Instance retrieval
   Input: concept C, KB $\mathcal{K}$
   Problem: List all instances of C in $\mathcal{K}$

   Retrieve all heroes that have a mortal mother.

Note: problems 1–3 and 6 are decision problems, their answer is yes/no.

Knowledge Base Satisfiability

Finally, one of the most important services:

KB satisfiability

Input: a KB $\mathcal{K}$
Problem: Is $\mathcal{K}$ *satisfiable*, that is, does there exists a model of $\mathcal{K}$?

- Does our KB make sense? Are there contradictions in it?
- Closest to reasoning in traditional FOL
- We can often reduce other decision problems to KB (un)satisfiability

C is subsumed by D w.r.t. $(\mathcal{T}, \mathcal{A})$ iff $(\mathcal{T}, \mathcal{A} \cup \{C \sqcap \neg D(b)\})$ is unsatisfiable
C is satisfiable w.r.t. $(\mathcal{T}, \mathcal{A})$ iff $(\mathcal{T}, \mathcal{A} \cup \{C(b)\})$ is satisfiable
C and D are disjoint w.r.t. $(\mathcal{T}, \mathcal{A})$ iff $(\mathcal{T}, \mathcal{A} \cup \{C \sqcap D(b)\})$ is unsatisfiable
a is an instance of C in $(\mathcal{T}, \mathcal{A})$ iff $(\mathcal{T}, \mathcal{A} \cup \{\neg C(a)\})$ is unsatisfiable
where b is a fresh individual not occurring in $\mathcal{T}$ or $\mathcal{A}$.

➤ the reductions may require constructors (negation, conjunction) which may not be available in some logics

Outline

How hard is it to reason in expressive DLs?

For expressive DLs

- standard reasoning problems easily reduce to KB (un)satisfiability
- and have the same complexity
  - ➤ In $\mathcal{ALC}$ and some extensions, reasoning about concepts only is easier (unless PSpace = ExpTime)

To decide satisfiability of $\mathcal{K}$, algorithms search for (a representation of) a model of $\mathcal{K}$

- The larger this model representation may be, the harder the problem
- For most expressive DLs, it may be of at least single exponential size

Complexity of Reasoning in DLs

	Concept satisfiability	KB satisfiability
$\mathcal{ALC}$, $\mathcal{ALCIQ}$	PSpace-complete	ExpTime-complete
$\mathcal{SH}, \mathcal{SHIQ}$	ExpTime-complete	
$\mathcal{SHOIQ}$	NExpTime-complete	
$\mathcal{SRIQ}$	2ExpTime-complete	
$\mathcal{SROIQ}$	2NExpTime-complete	

➤Despite their high complexity, most of these DLs are supported by efficient reasoners

Some notes on complexity

- Reasoning about KBs with acyclic TBoxes usually has the same complexity as reasoning about concepts only
- In all DLs that contain (or can express) $\mathcal{SH}$, reasoning about a single concept is already as hard as reasoning about arbitrary KBs
- The combination of inverses $\mathcal{I}$, nominals $\mathcal{O}$ and counting ($\mathcal{Q}$, $\mathcal{N}$ or $\mathcal{F}$) results in more complicated models
- In the $\mathcal{SR}$ family, complex role inclusions $R_1 \circ \ldots R_n \sqsubseteq R$ make model representations exponentially larger

From lightweight to expressive DLs, and back

- In the early years of DLs, researchers struggled to find suitable tractable DLs
- However, the minimal combinations of constructors considered desirable (e.g., $\sqcap + \forall$) made reasoning NP-hard
- With the appearance of the FaCT system in the 1990s, efficient reasoning with (ExpTime) hard DLs seemed possible
- This lead to the development of increasingly expressive logics

$$\mathcal{ALC} \leadsto \mathcal{SHIQ} \leadsto \text{OWL 1 } (\mathcal{SHOIQ}) \leadsto \text{OWL 2 } (\mathcal{SROIQ})$$

- But with this transition, the promise of efficiency on natural inputs became increasingly untrue
- In some applications this complexity is unacceptable

Lightweight DLs

Lightweight DLs

- For many applications, scalable, lightweight DLs are enough
- Existential restrictions are crucial (universal ones not always)
- Research increasingly focused on these DLs in the last years

The most prominent examples are

$\mathcal{EL}$ and *DL-Lite*

We also mention Horn DLs which are more expressive but preserve some of their positive features

Outline

The Basic $\mathcal{EL}$

Essentially, $\mathcal{EL}$ is a half of $\mathcal{ALC}$:

- It supports existential restrictions $\exists R.C$, but no universal ones
- It supports conjunction $C \sqcap D$, but no disjunction
- Of course, it does not allow for negation
  - but we can use $\bot$ to express a restricted form of negation

$\mathcal{EL}$ concepts are defined inductively as follows

$$C, D \longrightarrow A \mid \top \mid C \sqcap D \mid \exists R.C$$

where $A \in \mathsf{N}_C$ is a concept name and $R \in \mathsf{N}_R$ is a role.

Motivation and Applications

In many applications existential restrictions and conjunction seem to play a central role.

- Many medical and Life Sciences ontologies rely on this kind of axioms:

ViralPneumonia	$\sqsubseteq$	∃CausitiveAgent.Virus
ViralPneumonia	$\sqsubseteq$	InfectiousPneumonia
InfectiousPneumonia	$\sqsubseteq$	Pneumonia ⊓ InfectiousDisease
Pneumonia	$\sqsubseteq$	∃AssociatedMorphology.Inflammation
Pneumonia	$\sqsubseteq$	∃FindingSite.Lung

Motivation and Applications (cont'd)

- SNOMED CT (Systematized Nomenclature of Medicine – Clinical Terms) is written in (a minor extension of) $\mathcal{EL}$
- So are
  - large fragments of the GALEN ontology (Generalized Architecture for Languages, Encyclopaedias and Nomenclatures in medicine), another very important medical ontology `http://www.openclinical.org/prj_galen.html`
  - the Gene Ontology, and ontology for biology with the aim of "standardizing the representation of gene and gene product attributes across species and databases" `http://www.geneontology.org/`
  - etc.

Satisfiability in $\mathcal{EL}$

In the basic $\mathcal{EL}$ (i.e. without $\bot$)

- Satisfiability (w.r.t. a TBox / KB) is trivial
  - There is no way to express contradictions
  - Every concept C is satisfiable (w.r.t. every TBox / every KB)
- Algorithms focus on deciding subsumption
  - We can build a canonical model that witnesses all subsumptions
  - The model can be built in polynomial time

If we allow the use of $\bot$, satisfiability is not trivial but can also be decided in polynomial time using the canonical model

Theorem

Satisfiability and subsumption (w.r.t. a TBox/KB) in $\mathcal{EL}^\bot$ are P-complete

Other polynomial Extensions of $\mathcal{EL}$

Additionally to $\bot$, we can also add the following to $\mathcal{EL}$:

- Nominals $\{a\}$
- Domain and range restrictions $\top \sqsubseteq \forall R^-.C$, $\top \sqsubseteq \forall R.C$
- Complex role inclusions $R_1 \circ \ldots \circ R_n \sqsubseteq R$

We can adapt the canonical model construction to accommodate these features, and reasoning is still feasible in polynomial time

Roughly, this results in the DL called $\mathcal{EL}^{++}$

ExpTime-hard Extensions of $\mathcal{EL}$

In other extensions of $\mathcal{EL}$, reasoning (w.r.t. arbitrary TBoxes) becomes ExpTime-hard:

- $\mathcal{ELU}^\bot$ that extends $\mathcal{EL}^\bot$ with disjunction
  - We can reduce concept satisfiability w.r.t. to a TBox in $\mathcal{ALC}$ to TBox satisfiability in $\mathcal{ELU}^\bot$
- $\mathcal{ELU}$ that extends $\mathcal{EL}$ with disjunction
  - We can reduce concept satisfiability w.r.t. to a TBox in $\mathcal{ALC}$ to the same problem in $\mathcal{ELU}^\bot$
- $\mathcal{EL}^\forall$ that extends $\mathcal{EL}$ with value (or universal) restrictions $\forall R.C$
  - We can reduce concept satisfiability w.r.t. to a TBox in $\mathcal{ELU}$ to the same problem in $\mathcal{EL}^\forall$

There is no known extension of $\mathcal{EL}$ between P and ExpTime

The Basic $DL\text{-}Lite$

In $DL\text{-}Lite$, we distinguish between two kinds of concepts

1. Basic concepts B, with the following syntax:

$$B \longrightarrow A \mid \exists R \mid \exists R^-$$

   where $\exists R$ is an alternative syntax for $\exists R.\top$

2. (General) concepts C, which additionally allow for negation and conjunction

$$C \longrightarrow B \mid \neg B \mid C_1 \sqcap C_2$$

GCIs are a bit *asymmetric* and allow general concepts only on the r.h.s.

$$B \sqsubseteq C$$

Motivation and Applications

DL-Lite was specially tailored in such a way that:

- traditional reasoning problems are all solvable in polynomial time
- the data described by the ontology can be queried efficiently
  - it has very low computational complexity
  - it can be achieved by relying on existing database technologies

  (more later)
- it can express basic data and conceptual modeling formalisms, like ER-diagrams and UML class diagrams
  - among other advantages, this allows for formal reasoning in these formalisms, and for studying their complexity

Motivation and Applications (cont'd)

The application of *DL-Lite* has been specially successful in areas like:

- ontology based data access
- information and data integration
- conceptual modeling

and similar data-oriented fields.

Model construction in *DL-Lite*

- Similarly to $\mathcal{EL}$, a satisfiable *DL-Lite* concept/KB has a canonical model that allows to solve standard reasoning tasks
- The canonical model can be built using a DB-like chase procedure as known from databases
- Moreover, most problems can be solved without actually constructing the canonical model

Reasoning in *DL-Lite*

- Unsatisfiability in *DL-Lite* can only arise due to some $C \sqsubseteq \neg D$ implied by the TBox that is violated in the ABox
  - To check satisfiability, we only need to derive all the $C \sqsubseteq \neg D$ that follow form the TBox and check them
  - This can be done in polynomial time
- Subsumption is reducible to KB unsatisfiability

$$\langle \mathcal{T}, \mathcal{A} \rangle \models C \sqsubseteq D \quad \text{iff} \quad \langle \mathcal{T}', \mathcal{A}' \rangle \text{ is unsatisfiable}$$

where $\mathcal{T}' = \mathcal{T} \cup \{A \sqsubseteq C, A \sqsubseteq \neg D\}$ and $\mathcal{A}' = \mathcal{A} \cup \{A(d)\}$ for fresh A and d

Combined vs. Data Complexity

- So far, our complexity considerations have assumed combined complexity
  - 'standard' measure of complexity
  - takes into account the size of the full input, including TBox and ABox
- For certain settings, more fine-grained notions of complexity known from databases are more adequate
- When the ABox may contain big amounts of data and its much larger than the terminological component, we focus on data complexity

Definition (Data complexity)

Data complexity is the complexity of reasoning w.r.t. to an input ABox, where the terminological component (TBox, concepts) is assumed to be fixed

Data Complexity in DLs

- All expressive DLs are intractable in data complexity
  - practically all of them are NP-complete (for satisfiability)
- $\mathcal{EL}$ is P-complete in data complexity

A crucial difference between $\mathcal{EL}$ and $DL\text{-}Lite$ is that DL-Lite has lower data complexity

Data Complexity in $DL\text{-}Lite$

Theorem

The data complexity of reasoning in $DL\text{-}Lite$ is not higher than that of evaluating an SQL query over a database

- $DL\text{-}Lite$ has very low complexity
  - feasible in logarithmic space, and inside a (highly parallelizable) complexity class called AC_0
- Any reasoning problem over a $DL\text{-}Lite$ KB can be reduced to evaluating an SQL query over a database corresponding to the ABox
  - particularly appealing if we indeed have a very large and dynamic ABox
  - the implementation of this idea has made $DL\text{-}Lite$ a very popular formalism

Extensions of $DL\text{-}Lite$

There are many well known extensions of $DL\text{-}Lite$ that preserve its nice computational features, for example:

- In $DL\text{-}Lite_{\mathcal{F}}$ the TBox may include functionality assertions funct(R), funct(R^{-})
- In $DL\text{-}Lite_{\mathcal{R}}$ we have role inclusions, also of the form $R \sqsubseteq \neg S$ (sometimes called $DL\text{-}Lite^{\mathcal{H}}$)
- $DLR\text{-}Lite$, and the respective $_{\mathcal{F}}$ and $_{\mathcal{R}}$ extensions, allow for predicates of arity higher than 2

Many other extensions are defined in a 'less standard' way (e.g., $DL\text{-}Lite_{\mathsf{horn}}$, $DL\text{-}Lite_{\mathsf{krom}}$)

Beyond LogSpace

It is well known that, essentially, adding any other DL construct to $DL\text{-}Lite$ increases the data complexity beyond logarithmic space.

For example,

- adding concepts of the form $\exists R.A$ on the l.h.s. of GCIs, $\forall R.A$ on the r.h.s. or $\exists R^{-}.A$ on the l.h.s. makes reasoning NLogSpace-hard
- It we additionally allow conjunction on the l.h.s. reasoning becomes PTime hard (like in $\mathcal{EL}$)
- Concept negation, concept disjunction, or concepts of the form $\forall R.A$ on the l.h.s.,make reasoning already NP-hard

Lightweight Profiles for OWL 2

The new OWL 2 standard has profiles that are intended to support scalable reasoning:

- OWL EL is based on $\mathcal{EL}^{++}$
- OWL QL is based on $DL\text{-}Lite$

Don't miss the the tutorial by Markus Krötzsch!

Horn fragments of other DLs

We know that most extensions of $\mathcal{EL}$ and $DL\text{-}Lite$ lead to increased complexity of reasoning, but . . .

> are some of the positive features of these DLs preserved in more expressive logics?

Fortunately, yes:

- Horn fragments of DLs are obtained by restricting the syntax of expressive DLs in such a way that disjunction can not be expressed
- They fall inside the (well-known) Horn fragment of FOL
- This is usually enough to ensure the existence of one canonical model that suffices for all reasoning problems, as in $\mathcal{EL}$ and $DL\text{-}Lite$

Complexity of Horn DLs

- The data complexity of reasoning in Horn-DLs is usually PTime-complete
  - This holds for Horn-$\mathcal{SHIQ}$, Horn-$\mathcal{SHOIQ}$, Horn-$\mathcal{SRIQ}$, Horn-$\mathcal{SROIQ}$
- The combined complexity is not much lower than that of the non-Horn variant
  - Horn-$\mathcal{SHIQ}$ and Horn-$\mathcal{SHOIQ}$ are ExpTime-complete
  - Horn-$\mathcal{SRIQ}$ and Horn-$\mathcal{SROIQ}$ are 2ExpTime-complete

Roughly, this is because the (representation of) the canonical model may be as large and complex as in the non-Horn case

> Horn DLs allow to reason efficiently in the presence of large amounts of data

Outline

1. Motivation
2. Introduction to DLs
2.1 The language of DLs
2.2 $\mathcal{ALC}$ and its extensions
2.3 Reasoning Services
2.4 Complexity of reasoning

So far:

- Ontologies describe relevant terms and their relations

> An inflammation of the lungs is a pneumonia
> Streptococcus is a type of Gram positive Bacteria
> Hypertension is a synonym for high blood pressure

- DL ABoxes store data
  ➤more important: actual data sources can be viewed as/mapped to ABoxes

patient(4971.120462)	hasFinding$(4971.120462, \mathrm{f14})$
inflammation$(\mathrm{f14})$	hasLocation$(\mathrm{f14}, \mathrm{lung})$
hasCausitiveAgent$(\mathrm{f14}, \mathrm{strepPn})$	strepBacteria$(\mathrm{strepPn})$

Queries in DLs

- The user can pose queries over the vocabulary of the ontology, the system performs reasoning to return all answers

> • *Retrieve antibiotics that can be used to treat Gram-positive bacterial pneumonia*
>
> • *Determine whether patient 6771.120884 has a close relative that is allergic to penicillin*
>
> • *Retrieve all patients diagnosed with bacterial pneumonia that have an antibiotic allergy, or have a direct relative that has an antibiotic allergy*

Queries in DLs (cont'd)

- Sometimes query answering is reducible to instance checking

> **For example,**
>
> The query
>
> *Determine whether patient 6771.120884 has a close relative that is allergic to penicillin*
>
> reduces to checking whether
>
> $\mathcal{K} \models \mathsf{Patient} \sqcap \exists \mathsf{hasRelative}.(\exists \mathsf{hasAllergy}.\mathsf{Penicillin})(6771.120884)$

- But this holds only for the very simple queries that can be written as concepts

Queries in DLs (cont'd)

For example,

The (similar) query

Determine whether there exists a common allergy for some relative of patient 6771.120884

amounts to the FOL formula

$$\text{Patient}(6771.120884) \wedge \exists x, y. \big(\text{hasRelative}(6771.120884, x) \wedge \\ \wedge \text{hasAllergy}(x, y) \wedge \text{hasAllergy}(6771.120884, y)\big)$$

which is not equivalent to any DL concept

- DL expressions are poor query languages!

Query Languages (cont'd)

We want to access and process data using database inspired query languages that allow to flexibly select and join pieces of information

- Such queries are, in general, not expressible in DLs
- The existing algorithms and complexity results do not apply for them

We need new reasoners, new reasoning techniques, new algorithms, and new complexity bounds

We briefly discuss some of them in the rest of this tutorial

Outline

Databases and DL Ontologies

- A TBox is similar to a conceptual schema, but while the latter is only important in the design phase, the TBox will still be relevant when actually answering queries
- It expresses constraints on the schema, but the semantics is different from traditional DB constraints

  $\rightsquigarrow$ the constraints need not be satisfied by the ABox (database) at run time!

DBs and Complete information

In traditional databases it is assumed that information is complete

For example, if the train departure table contains only

Destination	Departure
⋮	⋮
Innsbruck	08:10
Innsbruck	10:10
Innsbruck	12:10
Innsbruck	14:10
Innsbruck	17:10
⋮	⋮

then we know that there is no train to Innsbruck departing at 9:05

KBs and Incomplete information

In contrast, in a DL KB we do not assume that information is complete

For example, if we only have

$$\mathsf{Person}(Andrea) \qquad \mathsf{Female}(Maria)$$

then it does not imply that Maria is not a person, not that Andrea is neither male nor female.

In fact, incomplete information results in different models that have to be taken into account when answering queries

If in the example above we also have

$$\mathsf{person} \sqsubseteq \mathsf{male} \sqcup \mathsf{female}$$

then we will have at least two models: one where Andrea is male, and one where Andrea is female.

Open vs. Closed World Assumption

Formally, we have

- In Databases we make the closed word assumption (CWA): the facts that are not known to be true are considered false
- In contrast in DLs, like in standard first order logic, we make the open word assumption (OWA): a fact whose truth we know nothing about can be either true or false

This semantic difference has a huge impact on query answering!

Query answering in DLs vs. query answering in DBs

- A DB is a relational structure
- A KB represents a set of relational structure, its models

Given a KB $\mathcal{K}$ and a query q, we are interested in the certain answers to q over $\mathcal{K}$, i.e., in the answers that occur in every model of $\mathcal{K}$.

Closely related to query answering in incomplete databases

- In Databases, the query is evaluated over one structure
  $\leadsto$ model checking is computationally easy
- In DLs and incomplete DBs, the query is answered over many structures
  $\leadsto$ logical consequence is computationally costly

Querying Knowledge Bases

The most important reasoning problem is query answering

Query Answering

Given a KB $\mathcal{K}$ and a query q, compute the tuples of individuals that are an answer for q in every model of $\mathcal{K}$

- We will define the notion of answer formally once we have formally defined the query language
- We sometimes consider queries with no answer variables, for which the answer is **true** if the query is true in all models, or **false** otherwise

Querying Knowledge Bases - Example

TBox $\mathcal{T}$:

$$\begin{aligned} \exists \mathsf{hasFather}.\top &\sqsubseteq \mathsf{Person} \\ \exists \mathsf{hasFather}^-.\top &\sqsubseteq \mathsf{Person} \\ \mathsf{Person} &\sqsubseteq \exists \mathsf{hasFather} \end{aligned}$$

ABox $\mathcal{A}$: $\mathsf{Person}(\mathrm{john})$, $\mathsf{Person}(\mathrm{nick})$, $\mathsf{Person}(\mathrm{toni})$, $\mathsf{hasFather}(\mathrm{john{,}nick})$, $\mathsf{hasFather}(\mathrm{nick{,}toni})$

Queries:

$$\begin{aligned} q_1(x,y) &\leftarrow \mathsf{hasFather}(x,y) \\ q_2(x) &\leftarrow \exists y.\,\mathsf{hasFather}(x,y) \\ q_3(x) &\leftarrow \exists y_1,y_2,y_3.\,\mathsf{hasFather}(x,y_1) \land \mathsf{hasFather}(y_1,y_2) \\ &\quad \land \mathsf{hasFather}(y_2,y_3) \\ q_4(x,y_3) &\leftarrow \exists y_1,y_2.\,\mathsf{hasFather}(x,y_1) \land \mathsf{hasFather}(y_1,y_2) \\ &\quad \land \mathsf{hasFather}(y_2,y_3) \end{aligned}$$

Certain answers:

$$\begin{aligned} \mathsf{cert}(q_1,\langle\mathcal{T},\mathcal{A}\rangle) &= \{\ (\mathrm{john{,}nick}),\ (\mathrm{nick{,}toni})\ \} \\ \mathsf{cert}(q_2,\langle\mathcal{T},\mathcal{A}\rangle) &= \{\ \mathrm{john},\ \mathrm{nick},\ \mathrm{toni}\ \} \\ \mathsf{cert}(q_3,\langle\mathcal{T},\mathcal{A}\rangle) &= \{\ \mathrm{john},\ \mathrm{nick},\ \mathrm{toni}\ \} \\ \mathsf{cert}(q_4,\langle\mathcal{T},\mathcal{A}\rangle) &= \{\ \} \end{aligned}$$

Choosing a query language

We want to access and process data using database inspired query languages that allow to flexibly select and join pieces of information

Which is the best query language?

Some candidates:

- DL expressions: concepts and roles
  - They allow us to do simple instance queries
  - but as we have discussed, have very limited expressive power
- Formulas in FOL
  - A natural candidate - recall examples above
  - but they are not decidable
  - ⇝ answering yes/no queries over an empty KB amounts to deciding FOL validity

Choosing a query language (cont'd)

A good alternative:

- Conjunctive Queries (CQs)
  - A special kind of positive existential FOL formulas
  - Equivalent to the plain Select-Project-Join fragment of SQL
  - Very popular in databases, standard language in many areas
  - Around 90% of the queries in actual applications fall in this fragment
  - Positive computational features
  - All the mentioned examples are CQs

They have been extensively studied for a wide range of DLs

An Example Conjunctive Query

$$q(x) \leftarrow \text{Hero}(x), \text{hasMother}(x, v_1), \text{hasAncestor}(v_1, v_2), \text{Deity}(v_2)$$

Or, using standard FOL syntax:

$$q(x) \leftarrow \exists v_1, v_2.\text{Hero}(x) \wedge \text{hasMother}(x, v_1) \wedge \text{hasAncestor}(v_1, v_2) \wedge \text{Deity}(v_2)$$

The query asks for the heroes x that have a divine ancestor on the maternal side

(we use red to highlight the answer variables)

Other query languages that have been studied for some DLs

Unions of Conjunctive queries (UCQs): disjunctions of CQs

$$q(x) \leftarrow \{\text{Hero}(x), \text{hasMother}(x, v_1), \text{hasAncestor}(v_1, v_2), \text{Deity}(v_2)\} \cup \{\text{hasWife}(x, v_1), \text{Deity}(v_1)\}$$

Or, in FOL syntax

$$q(x) \leftarrow (\exists v_1, v_2.\text{Hero}(x) \wedge \text{hasMother}(x, v_1) \wedge \text{hasAncestor}(v_1, v_2) \wedge \text{Deity}(v_2)) \vee (\exists v_1.\text{hasWife}(x, v_1) \wedge \text{Deity}(v_1))$$

heroes that have a divine ancestor on the maternal side or are married with a goddess

They are also very popular, and they preserve many of the good computational properties of CQs

Other query languages (cont'd)

- Positive queries: positive FOL formulas

$$q(x_1, x_2) \leftarrow \exists v.\ \text{hasRelative}(x_1, x_2) \wedge \text{hasChild}(x_1, v) \wedge \text{hasChild}(x_2, v) \wedge \text{Male}(x_1) \wedge \text{Female}(x_2) \wedge (\text{Mortal}(x_1) \vee \text{Mortal}(x_2))$$

pairs of individuals who are relatives, have a common child v, and at least one of them is mortal

They have the same expressiveness as UCQs, but they are more succinct

Other query languages (cont'd)

- Queries that allow to express more complex navigations on the database, in the style of XPath

$$q(x_1, x_2) \leftarrow \text{hasParent}^* \circ \text{hasParent}^{-*}(x_1, x_2)$$

pairs of individuals who are relatives

- Combinations of the above

$$q_1(x_1, x_2) \leftarrow \exists v.\ \text{hasParent}^* \circ \text{hasParent}^{-*}(x_1, x_2) \wedge \text{hasChild}(x_1, v) \wedge \text{hasChild}(x_2, v) \wedge \text{Male}(x_1) \wedge \text{Female}(x_2) \wedge (\text{Mortal}(x_1) \vee \text{Mortal}(x_2))$$

pairs of individuals who are relatives, have a common child v, and at least one of them is mortal

Outline

Conjunctive Queries

Conjunctive Query (CQ)

A *conjunctive query* is a formula of the form

$$q(\vec{t}) = \exists \vec{v}. A_1(\vec{v}_1) \wedge \ldots \wedge A_n(\vec{v}_n)$$

where

- $\vec{t}$ and $\vec{v}$ are lists of constants and variables,
- the A_i are concepts/roles,
- the $\vec{v}_i$ are lists of arguments of matching arity,
- and $\vec{v}_i \subseteq \vec{t} \cup \vec{v}$ for each i.

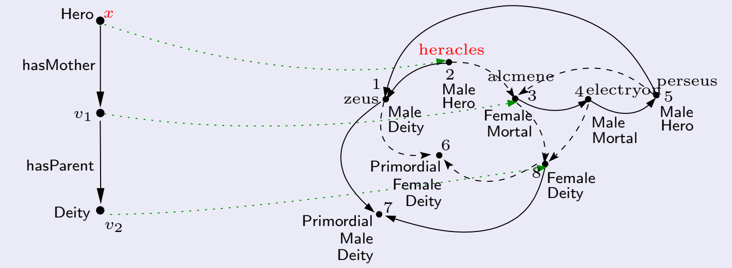
We often write conjunctive queries as lists (or even sets) of atoms

$$q(\vec{t}) = A_1(\vec{v}_1), \ldots, A_n(\vec{v}_n)$$

To define query answers, we use the notion of match

Query Match - Example

Intuitively, a match for a query q is an assignment of query variables to elements in an interpretation that makes q true. The answer to q is given by the image of the answer variables.



$$q_2(x) \leftarrow \text{Hero}(x), \text{hasMother}(x, v_1), \text{hasAncestor}(v_1, v_2), \text{Deity}(v_2)$$

Answer: heracles

Query Match, Query Answer

Formally:

Query Match

A *match for $q(\vec{t})$ in an interpretation $\mathcal{I}$* is a mapping from the variables and constants in q to $\Delta^{\mathcal{I}}$ such that

- $\pi(a) = a^{\mathcal{I}}$ for each individual a,
- $\pi(x) \in A^{\mathcal{I}}$ for each $A(x) \in q$, and
- $\langle \pi(x), \pi(y) \rangle \in r^{\mathcal{I}}$ for each $r(x, y) \in q$.

Query Answer

A tuple of individuals $\langle a_1, \ldots, a_n \rangle$ is called an *(certain) answer for $q(t_1, \ldots, t_n)$ over $\mathcal{K}$* if in every model $\mathcal{I}$ of $\mathcal{K}$ there is a match π for q such that $\pi(t_i) = a_i^{\mathcal{I}}$ for every i. We use $\text{cert}(q, \mathcal{K})$ to denote the set of certain answers for $q(t_1, \ldots, t_n)$ over $\mathcal{K}$.

Query Answering

Query answering consists on listing the answers to a query, i.e., it is an enumeration problem:

Definition (Query answering problem)

Given a KB $\mathcal{K}$ and a query q over $\mathcal{K}$, list all the tuples $\vec{c}$ of constants such that $\vec{c} \in \mathsf{cert}(q, \mathcal{K})$.

When studying the complexity of query answering, we need to consider the associated decision problem:

Definition (Recognition problem for query answering)

Given a KB $\mathcal{K}$, a query q over $\mathcal{K}$, and a tuple $\vec{c}$ of constants, check whether $\vec{c} \in \mathsf{cert}(q, \mathcal{K})$.

Entailment of Boolean Queries

Definition (Boolean query)

A query q over $\mathcal{K}$ that has no answer variables is called Boolean.

For a Boolean query q, the main reasoning task is deciding whether q evaluates to $\mathbf{true}$ in all models:

Definition (Query entailment problem)

For a Boolean query q and a KB $\mathcal{K}$, we write $\mathcal{K} \models q$ if there is a match for q in every model of $\mathcal{K}$.
Given a KB $\mathcal{K}$ and a Boolean query q over $\mathcal{K}$, the query entailment problem is to decide whether $\mathcal{K} \models q$.

Query Answering and Query Entailment

- The recognition problem for query answering reduces to the entailment problem for Boolean queries:
  - Simply instantiate the query with the input tuple and verify the entailment of the resulting Boolean query
- Many algorithms focus on query entailment only
- Query answering can then be achieved by calling the query entailment procedure for each possible tuple
(only constants occurring in the KB, thus finitely many tuples)
- In practice, of course, listing query answers should be done with smarter algorithms

Query answering in lightweight DLs

CQ entailment has been studied for many DLs.

For lightweight DLs like the DL-Lite and $\mathcal{EL}$ families, focus on:

- data complexity, which is usually tractable
- practical techniques for query answering with large amounts of data
  - query answering using existing technologies
  - in particular, using reductions into SQL and existing RDBMSs
  - or using other existing database technologies, such as Datalog engines

Recently, this kind of techniques have been explored for more expressive Horn DLs.

Query answering in expressive DLs

For expressive DLs that extend $\mathcal{ALC}$

- data complexity is typically coNP-complete
- the landscape for combined complexity of query answering is not so simple
- worst-case optimal algorithms are hard to come about
- until now, many decidability/complexity results obtained, but no practical algorithms implemented

Complexity of CQ entailment in DLs

	Combined complexity	Data complexity
Plain databases	NP-complete	in AC_0
DL-Lite	NP-complete	in AC_0
$\mathcal{EL}$	NP-complete	P-complete
Horn-$\mathcal{SHIQ}$	ExpTime-complete	P-complete
$\mathcal{SHIQ}$	2ExpTime-complete[(1)]	coNP-complete [(2)]
$\mathcal{SHOIQ}$	decidability open	

[(1)] CQ answering is already 2ExpTime-hard for $\mathcal{ALCI}$ and $\mathcal{SH}$. $\mathcal{SHOI}$ and $\mathcal{SHOQ}$ are also in 2ExpTime.

[(2)] Already for TBoxes with a single disjunction in fragments of $\mathcal{ALC}$

Query Answering vs. standard Reasoning

Instance checking	Combined complexity	Data complexity
DL-Lite	in P	in AC_0
$\mathcal{EL}$	P-complete	P-complete
Horn-$\mathcal{SHIQ}$	ExpTime-complete	P-complete
$\mathcal{SHIQ}$	ExpTime-complete	coNP-complete
$\mathcal{SHOIQ}$	NExpTime-complete	coNP-hard

Query answering	Combined complexity	Data complexity
DL-Lite	NP-complete	in AC_0
$\mathcal{EL}$	NP-complete	P-complete
Horn-$\mathcal{SHIQ}$	ExpTime-complete	P-complete
$\mathcal{SHIQ}$	2ExpTime-complete	coNP-complete
$\mathcal{SHOIQ}$	dec. open	coNP-hard

Query Answering with Relational Database Systems

- Existing Relational Database Systems seem the most promising approach for achieving scalability of query answering
- Main challenge to be overcome:

  $\leadsto$ How do we make a RDBMS aware of the TBox?

  - Option 1: incorporate the TBox into the query $\leadsto$ query rewriting
  - Option 2: incorporate the TBox into the ABox $\leadsto$ data completion

These two approaches are analogous to backward chaining and forward chaining in automated deduction and logic programming.

The Query Rewriting Approach

This approach was introduced by Calvanese et.al. for DL-Lite

> Idea: Given a KB $\langle \mathcal{T}, \mathcal{A} \rangle$ and a CQ q, obtain a FOL query $q_{\mathcal{T}}$ such that for every tuple $\vec{a}$ of constants,
>
> $\vec{a}$ is an answer for q over $\langle \mathcal{T}, \mathcal{A} \rangle$
>
> iff
>
> $\vec{a}$ is an answer for $q_{\mathcal{T}}$ over $\mathcal{A}$ (using the usual DB semantics)

This allows us to directly use off-the-shelf RDBMSs (FOL queries are equivalent to SQL over standard DBs):

- The ABox is stored directly as a database
- The query $q_{\mathcal{T}}$ is then evaluated over this DB
- Optimal from the data complexity point of view, since it really optimizes query answering w.r.t. the data size

The Query Rewriting Approach – Example 1

TBox $\mathcal{T}$:
$$\begin{array}{rcl} B' & \sqsubseteq & B \\ \exists S.\top & \sqsubseteq & A \end{array}$$

Query: $q \leftarrow A(x), R(x,y), B(y)$

The rewriting of q is the disjunction of:
$$\begin{array}{l} A(x), R(x,y), B(y); \\ A(x), R(x,y), B'(y); \\ S(x,z), R(x,y), B(y); \\ S(x,z), R(x,y), B'(y); \end{array}$$

- A CQ q is reformulated into a UCQ $q_{\mathcal{T}}$
- Intuitively, we exploit the GCIs to obtain new queries that can contribute to the answer

Query rewriting in DL-Lite

The rewriting algorithm is given as a set of rules that apply the GCIs in $\mathcal{T}$ (from right to left) to a given query:

$$\begin{array}{rccc}
A_1 \sqsubseteq A_2 & \ldots, A_2(x), \ldots & \rightsquigarrow & \ldots, A_1(x), \ldots \\
\exists P \sqsubseteq A & \ldots, A(x), \ldots & \rightsquigarrow & \ldots, P(x, _), \ldots \\
\exists P^- \sqsubseteq A & \ldots, A(x), \ldots & \rightsquigarrow & \ldots, P(_, x), \ldots \\
A \sqsubseteq \exists P & \ldots, P(x, _), \ldots & \rightsquigarrow & \ldots, A(x), \ldots \\
A \sqsubseteq \exists P^- & \ldots, P(_, x), \ldots & \rightsquigarrow & \ldots, A(x), \ldots \\
\exists P_1 \sqsubseteq \exists P_2 & \ldots, P_2(x, _), \ldots & \rightsquigarrow & \ldots, P_1(x, _), \ldots \\
P_1 \sqsubseteq P_2 & \ldots, P_2(x,y), \ldots & \rightsquigarrow & \ldots, P_1(x,y), \ldots \\
\ldots & & &
\end{array}$$

where _ denotes a fresh variable that appears only once

Roughly, we obtain the rewritten $q_{\mathcal{T}}$ by applying the rules and unifying variables in every possible way.

The limits of Query Rewriting

- Answering FOL queries in standard DBs is in AC_0 w.r.t. data complexity
- In query rewriting
    - The data, which is the only measured input, is not changed
    - The rewriting does not depend on the data
- Hence, the rewriting approach (into FOL queries) can only work for DLs whose data complexity is in AC_0
- That is, we can only use it for the DL-Lite family

$\mathcal{EL}$ and Query Rewriting

TBox $\mathcal{T}$: $\exists S.A \sqsubseteq A$

Query: $q \leftarrow A(x)$

The rewriting of q is the disjunction of:

$$A(x)$$
$$S(x,y_1), A(y_1)$$
$$S(x,y_1), S(y_1,y_2), A(y_2)$$
$$S(x,y_1), S(y_1,y_2), S(y_2,y_3), A(y_3)$$
$$\dots$$

This can not be written as a finite SQL query!

It can be written as $S^*(x,y), A(y)$, but transitive closure is not FOL-expressible

> $\mathcal{EL}$ is P-hard in data complexity, hence we can not use the Query Rewriting approach as defined above

Query Rewriting beyond DL Lite

> First-order rewritability fails for every DL beyond DL Lite

Some possible solutions:

1. Rewrite into query language that is more expressive than FOL
   - Every CQ over $\mathcal{EL}$ can be rewritten into a Datalog query

     The query above is equivalent to the Datalog query

     $$q(x) :- A(x)$$
     $$A(x) :- R(x,y), A(y)$$

   - Rewriting into Datalog works even for Horn-$\mathcal{SHIQ}$, the most expressive DL for which query answering has been implemented
2. Give up the *data independence* of the rewriting approach, and modify also the ABox (Lutz el.al. 08)

The Data completion approach – Naive attempt

Basic idea: add the TBox information to the ABox

- For each lightweight DL KB there is one canonical model that can be used for answering all queries
- If we represent that canonical model as a database, then we can simply pose queries to it
- But often this does not work:

  $\rightsquigarrow$ the canonical model for query answering may be infinite, even for DL Lite and $\mathcal{EL}$!

The Data completion approach

Question: Can we use the data completion approach?
Answer: Only partially - we need to combine it with, e.g., query rewriting

> **The combined approach (Lutz et al. 09)**
>
> Idea: Given a KB $\langle \mathcal{T}, \mathcal{A} \rangle$ and a CQ q, obtain a FOL query q' and an ABox $\mathcal{A}'$ such that for every tuple $\vec{a}$ of constants,
>
> $\vec{a}$ is an answer for q over $\langle \mathcal{T}, \mathcal{A} \rangle$
>
> iff
>
> $\vec{a}$ is an answer for q' over $\mathcal{A}'$ (using the usual DB semantics)

This requires the data to be modified

- Assumes a different setting (e.g., access to the data a priori, privacy)

The Combined approach – An overview

- Instead of the real canonical model, we realize in the database another representative model that reuses existentially quantified elements
- Reusing elements may introduce spurious query matches

  $\leadsto$ that is why we need to rewrite the query as well
- With suitable rewritings, we can obtain a query that has a match in the small model iff it has a match in the canonical one

This approach has been successfully applied to $\mathcal{EL}$ and DL-Lite

Open issues

Despite the success of query rewriting and the combined approach, many challenges remain:

- The rewritten queries are often hard to evaluate
  - they can be very large (exponential blow-up in query size)
  - their 'unnatural' structure may not be adequate for existing DBMS optimizations
- The data completion stage of the combined approach may not be applicable: no access to the data a priori, no right to modify it
- Even if it is applicable, it can be very expensive

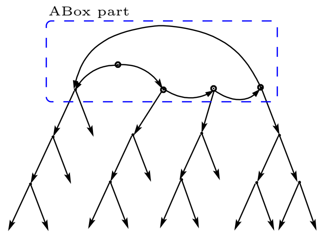
Many research efforts still aim at practicable and scalable query answering in lightweight DLs

Outline

Query Answering in Expressive DLs

Assume a given knowledge base $\mathcal{K}$ and a query q

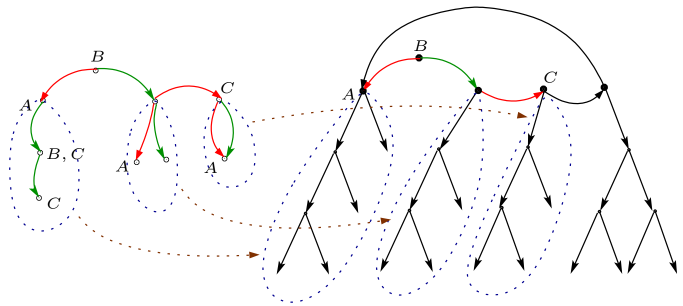
- We want to decide $\mathcal{K} \models q$
- This is equivalent to deciding whether there is a countermodel witnessing $\mathcal{K} \not\models q$ i.e. a model of $\mathcal{K}$ where there is no match for q
- For most DLs, one can show that if $\mathcal{K} \not\models q$, then there is a countermodel that has some kind of forest-shaped



Query Matches in Forest-shaped models

A match for q in a canonical model has two parts:

- a partial match into the A-Box part (roots)
- maps for subqueries inside the trees



Searching for Countermodels

- All existing algorithms search for a forest-shaped countermodel
- In many cases, this is done in three stages:
  1. Consider all partial matches of the query into the ABox part
  2. generate all combinations Q of subqueries that contain some subquery generated by a partial match
  3. For each Q, decide existence of a tree-shaped model part $\mathcal{I}$ with $\mathcal{I} \not\models Q$
- The last step focuses on trees only, and is often achieved by elaborate adaptations of TBox reasoning techniques

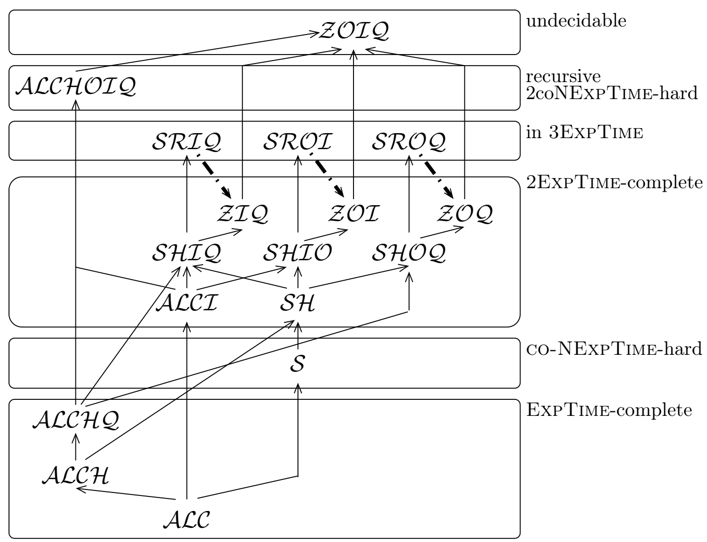
What makes query answering hard?

For most expressive DLs, query answering is very hard:

- There are exponentially many possible ABox parts, and exponentially many partial matches in each of them
- Q can be exponential in q
- A (subquery) can be matched to a tree in exponentially many different ways
- Deciding $\mathcal{I} \not\models Q$ inside a tree is exponentially harder than standard reasoning

Many algorithms for query answering in expressive DLs have been developed, but none of them seems implementable

Complexity of Query Answering in Expressive DLs



Overview of some Techniques for QA in Expressive DLs – Part 1

- Modified tableaux algorithms that take the query size into account when blocking
  - Often called n-blocking or CARIN blocking
  - First introduced for $\mathcal{ALCN}$ (in the context of a language called CARIN) (Levy and Rousset 98)
  - Has been extended to more expressive logics, but does not work for DLs with transitive roles

Overview of some Techniques for QA in Expressive DLs – Part 2

- Tuple-graph or rolling up techniques
  - Each way of mapping a subquery inside a tree can be expressed as a DL concept
  - Using this, query answering can be reduced to satisfiability testing
    - exponentially many satisfiability tests
    - each of them receives an exponentially larger KB as input
  - First introduced for a DL called $\mathcal{DLR}$ (Calvanese et.al. 98)
  - Yields optimal complexity bounds
  - Has been extended to other DLs like $\mathcal{ALCHQ}$, $\mathcal{SHIQ}$, and $\mathcal{SHOQ}$

Overview of some Techniques for QA in Expressive DLs – Part 3

- Automata on infinite trees reduce the existence of a countermodel to the emptiness test of a suitable automaton
  - Yield optimal bounds for combined complexity
  - They can handle both the ABox and the tree part, but are not optimal in data complexity
  - Can be combined with other techniques
  - Can accommodate many constructs
  - They have been used to obtain complexity bounds for the most expressive decidable DLs so far
- Knot-based techniques focus on the 'tree part' of the problem
  - Use simple, local representations of models
  - Allow to obtain optimal complexity bounds some hard cases

Summary

Query answering is a relatively new DL reasoning task that is gaining importance in many applications

- In general, query answering in DLs is harder and more involved than:
  - Query Answering in plain DBs
  - Traditional reasoning in DLs
- For lightweight DLs:
  - the complexity seems manageable
  - most successful approaches rely on relational DBs
  - in practice, scalability not so easy
  - many open challenges
- For expressive DLs:
  - the problem is usually very hard
  - many questions are still open
  - no practical algorithms available

Thanks!

Questions? Comments?