

#### **INFR11215 Knowledge Graphs**

# **Tractable Schema Reasoning**

#### Jeff Z. Pan



http://knowledge-representation.org/j.z.pan/

[Reading: Baader et al., Chapter 6]



### **Lecture Outline**

- Motivation
- Overview of EL and reasoning
- Detailed Discussions on EL and reasoning
- Practical

#### **Motivations**



EDINBURGH



- Web Ontology Language (OWL)
  - OWL v2 family
    - OWL 2 DL
    - OWL 2 EL, OWL 2 QL, OWL 2 RL
- ALC not a good starting point
  - its foundation FL<sub>0</sub> (⊓ and ∀) is not a good foundation
  - subsumption with GCI is EXPTimecomplete
  - EL is PTime-complete
    - TBox reasoning
    - ABox reasoning
    - Query answering

#### **Motivations**



EDINBURGH



#### • SNOMED Clinical Terms is

- Probably the single most comprehensive clinical terminology
- Licensed for national use throughout the UK and the US
- Content that covers most clinical concepts
- A terminology model that supports retrieval of alternative representations of similar information
- SNOMED CT is an EL ontology
  - clear performance difference between
     EL algorithm and algorithms for ALCextended logics



#### **Lecture Outline**

- Motivation
- Overview of EL and reasoning
- Detailed Discussions on EL and reasoning
- Practical



### What is EL



- EL Class Description
  - existential restriction: ∃r.C
  - conjunction: C  $\square$  D
  - the top class: ⊤
  - not including:
    - value restriction:  $\forall r.C$
    - ・ disjunction: C 凵 D
    - the bottom class:  $\perp$
- EL Axioms
  - GCI: C  $\sqsubseteq$  D

Knowledge Graphs Jeff Z. Pan



# **Class Satisfiability Checking in EL**



- Every EL class is satisfiable
  - Why?
  - Class satisfiability checking is not an interesting problem
- Challenge
  - Subsumption checking in EL is non-trivial, as it cannot be reduced to class unsatisfiability
  - Why?
  - 0 |= C  $\sqsubseteq$  D iff C  $\sqcap$  ¬D is unsatisfiable

# **Subsumption Checking in EL**



HE UNIVERS EDINBURGH

- Subsumption checking in EL (with general Tbox) is PTime-complete
  - For FL<sub>0</sub>, it is EXPTime-complete
- Usually this is done in a batch mode: classification
  - A TBox reasoning service that computes subsumption relation among all named classes
- Given an EL TBox T, signature Sig (T) contains all class and property names used in T



🝕 pizza.owl (http://www.co-ode.org/ontologies/pizza/2005/10/1	8/pizza.owl) - [C:\Users\Jeff\Documents\My Work\Onto\pizza-v051018.owl]			
File Edit Ontologies Reasoner Tools Refactor Tabs V	/iew Window Help			
A				
Active Ontology Entities Classes Object Properties Data P	Individuals OWI Viz DL Query			
Henre Onlogy Linues Classes Object reported Data				
Asserted class hierarchy Inferred class hierarchy	Class Annotations Class Usage	meos		
▶ ● Nothing	comment	080		
CheeseyVegetableTopping	"A class to demonstrate mistakes made with setting a property domain. The property has Topping has a domain of Pizza. This			
OmainConcept     Gountry	means that the reasoner can inter that all individuals using the has lopping property must be of type Pizza. Because of the restriction on this class, all members of IceCream must use the hasTopping property, and therefore must also be members of			
▼ ● Pizza	Pizza. However, Pizza and IceCream are disjoint, so this causes an inconsistency. If they were not disjoint, IceCream would be			
► 🖯 CheeseyPizza	Intered to be a subclass of Pizza. @en	000		
► UnterestingPizza	"Sorvete"@pt	000		
OnvegetarianPizza				
RealItalianPizza	Baggrintion: JosCraam			
	Description, decream			
SpicyPizza = SpicyPizzaEquivaler SpicyPizzaEquivalent = SpicyPizzaEquivalent	Equivalent classes 🕀			
►	Equivalent classes 🕘	0		
<ul> <li>SpicyPizza = SpicyPizzaEquivaler</li> <li>SpicyPizzaEquivalent = SpicyPizz</li> <li>VegetarianPizza</li> <li>PizzaBase</li> </ul>	Equivalent classes 😜	0		
<ul> <li>SpicyPizza ≡ SpicyPizzaEquivaler</li> <li>SpicyPizzaEquivalent ≡ SpicyPizz</li> <li>VegetarianPizza</li> <li>PizzaBase</li> <li>DeepPanBase</li> <li>ThinAndCrispyBase</li> </ul>	Equivalent classes 🚱  Nothing  Superclasses 💽	0		
<ul> <li>SpicyPizza = SpicyPizzaEquivaler</li> <li>SpicyPizzaEquivalent = SpicyPizz</li> <li>VegetarianPizza</li> <li>PizzaBase</li> <li>DeepPanBase</li> <li>ThinAndCrispyBase</li> <li>PizzaTopping</li> </ul>	Equivalent classes	000		
<ul> <li>SpicyPizza = SpicyPizzaEquivaler</li> <li>SpicyPizzaEquivalent = SpicyPizz</li> <li>VegetarianPizza</li> <li>PizzaBase</li> <li>DeepPanBase</li> <li>ThinAndCrispyBase</li> <li>PizzaTopping</li> <li>FishTopping</li> </ul>	Equivalent classes  Nothing Superclasses  DomainConcept hasTopping some FruitTopping	0 0 0 0 0 0 0 0 0		
<ul> <li>SpicyPizza = SpicyPizzaEquivaler</li> <li>SpicyPizzaEquivalent = SpicyPizz</li> <li>VegetarianPizza</li> <li>PizzaBase</li> <li>DeepPanBase</li> <li>ThinAndCrispyBase</li> <li>PizzaTopping</li> <li>FishTopping</li> <li>MeatTopping</li> <li>SpicyTopping</li> </ul>	Equivalent classes  Nothing Superclasses  DomainConcept hasTopping some FruitTopping Inferred anonymous superclasses	0 0 0 0 0 0 0 0		
<ul> <li>SpicyPizza = SpicyPizzaEquivaler</li> <li>SpicyPizzaEquivalent = SpicyPizz</li> <li>VegetarianPizza</li> <li>PizzaBase</li> <li>DeepPanBase</li> <li>ThinAndCrispyBase</li> <li>PizzaTopping</li> <li>FishTopping</li> <li>MeatTopping</li> <li>SpicyTopping</li> <li>VegetarianTopping</li> <li>VegetarianTopping</li> </ul>	Equivalent classes	0 0 2 2 2 3		
<ul> <li>SpicyPizza = SpicyPizzaEquivaler</li> <li>SpicyPizzaEquivalent = SpicyPizz</li> <li>VegetarianPizza</li> <li>PizzaBase</li> <li>DeepPanBase</li> <li>ThinAndCrispyBase</li> <li>PizzaTopping</li> <li>FishTopping</li> <li>MeatTopping</li> <li>SpicyTopping</li> <li>VegetarianTopping</li> <li>VegetarianTopping</li> </ul>	Equivalent classes  Nothing Superclasses  DomainConcept hasTopping some FruitTopping Inferred anonymous superclasses Members	0 0 0 0 0 0 0 0		
<ul> <li>SpicyPizza = SpicyPizzaEquivaler</li> <li>SpicyPizzaEquivalent = SpicyPizz</li> <li>VegetarianPizza</li> <li>PizzaBase</li> <li>DeepPanBase</li> <li>ThinAndCrispyBase</li> <li>Pizza Topping</li> <li>Fish Topping</li> <li>SpicyTopping</li> <li>SpicyTopping</li> <li>VegetarianTopping</li> <li>VegetarianTopping</li> <li>ValuePartition</li> <li>Spiciness</li> </ul>	Equivalent classes  Nothing Superclasses DomainConcept hasTopping some FruitTopping Inferred anonymous superclasses Members	©×• ©×•		
<ul> <li>SpicyPizza = SpicyPizzaEquivaler</li> <li>SpicyPizzaEquivalent = SpicyPizz</li> <li>VegetarianPizza</li> <li>PizzaBase</li> <li>DeepPanBase</li> <li>ThinAndCrispyBase</li> <li>PizzaTopping</li> <li>FishTopping</li> <li>SpicyTopping</li> <li>VegetarianTopping</li> <li>VegetarianTopping</li> <li>VegetarianTopping</li> <li>SpicyTopping</li> <li>Spiciness</li> <li>Hot</li> </ul>	Equivalent classes	© © © © © © © © © ©		
<ul> <li>SpicyPizza = SpicyPizzaEquivaler</li> <li>SpicyPizzaEquivalent = SpicyPizz</li> <li>VegetarianPizza</li> <li>PizzaBase</li> <li>DeepPanBase</li> <li>ThinAndCrispyBase</li> <li>PizzaTopping</li> <li>FishTopping</li> <li>MeatTopping</li> <li>SpicyTopping</li> <li>VegetarianTopping</li> <li>VegetarianTopping</li> <li>Spiciness</li> <li>Hot</li> <li>Medium</li> </ul>	Equivalent classes    Equivalent classes  Nothing  Superclasses  DomainConcept  has Topping some FruitTopping  Inferred anonymous superclasses  Members  Disjoint classes  Pizza Topping  Pizza	©×• ©×• ©×•		
<ul> <li>SpicyPizza = SpicyPizzaEquivaler</li> <li>SpicyPizzaEquivalent = SpicyPizz</li> <li>VegetarianPizza</li> <li>PizzaBase</li> <li>DeepPanBase</li> <li>ThinAndCrispyBase</li> <li>PizzaTopping</li> <li>FishTopping</li> <li>SpicyTopping</li> <li>VegetarianTopping</li> <li>VegetarianTopping</li> <li>Spiciness</li> <li>Hot</li> <li>Medium</li> <li>Mild</li> </ul>	Equivalent classes  Equivalent classes Equivalent classes Superclasses DomainConcept Commons superclasses Inferred anonymous superclasses Members Disjoint classes Disjoint classes PizzaTopping Pizza PizzaBase	©		

## **Normalisation**

 $\bullet$ 

Idea



EDINBURGH

- Simplify the axioms into some certain form so that reasoning algorithms can take advantage of it
- example: NNF (negated normal form)
- Normal forms for EL
  - $A \sqsubseteq B$
  - $A1 \sqcap A2 \sqsubseteq B$
  - A ⊑ ∃r.B
  - ∃r.A ⊑ B
  - where A, A1, A2, B are either named class in Sig(T) or the top class ⊤

Knowledge Graphs Jeff Z. Pan





EDINBURGH

 $\begin{array}{cccccccc} \mathsf{NF0} & \widehat{D} \sqsubseteq \widehat{E} & \longrightarrow & \widehat{D} \sqsubseteq A, & A \sqsubseteq \widehat{E} \\ \mathsf{NF1}_r & C \sqcap \widehat{D} \sqsubseteq B & \longrightarrow & \widehat{D} \sqsubseteq A, & C \sqcap A \sqsubseteq B \\ \mathsf{NF1}_\ell & \widehat{D} \sqcap C \sqsubseteq B & \longrightarrow & \widehat{D} \sqsubseteq A, & A \sqcap C \sqsubseteq B \\ \mathsf{NF2} & \exists r. \widehat{D} \sqsubseteq B & \longrightarrow & \widehat{D} \sqsubseteq A, & \exists r. A \sqsubseteq B \\ \mathsf{NF3} & B \sqsubseteq \exists r. \widehat{D} & \longrightarrow & A \sqsubseteq \widehat{D}, & B \sqsubseteq \exists r. A \\ \mathsf{NF4} & B \sqsubseteq D \sqcap E & \longrightarrow & B \sqsubseteq D, & B \sqsubseteq E \\ \text{where } C, D, E \text{ denote arbitrary } \mathcal{EL} \text{ concepts,} \\ \widehat{a} = \widehat{a} = \mathcal{A} = \mathcal{A} = \mathcal{A} = \mathcal{A} = \mathcal{A} \\ \end{array}$ 

 $\widehat{D}, \widehat{E}$  denote  $\mathcal{EL}$  concepts that are neither concept names nor  $\top$ , *B* is a concept name, and

A is a new concept name.



## **Example: Normalisation**

NF0	$\widehat{D} \sqsubseteq \widehat{E}$	$\longrightarrow$	$\widehat{D} \sqsubseteq A,$	$A \sqsubseteq \widehat{E}$
$NF1_r$	$C\sqcap \widehat{D}\sqsubseteq B$	$\longrightarrow$	$\widehat{D} \sqsubseteq A,$	$C\sqcap A\sqsubseteq B$
$NF1_\ell$	$\widehat{D}\sqcap C\sqsubseteq B$	$\longrightarrow$	$\widehat{D} \sqsubseteq A,$	$A\sqcap C\sqsubseteq B$
NF2	$\exists r. \widehat{D} \sqsubseteq B$	$\longrightarrow$	$\widehat{D} \sqsubseteq A,$	$\exists r.A \sqsubseteq B$
NF3	$B \sqsubseteq \exists r. \widehat{D}$	$\longrightarrow$	$A \sqsubseteq \widehat{D},$	$B \sqsubseteq \exists r.A$
NF4	$B\sqsubseteq D\sqcap E$	$\longrightarrow$	$B \sqsubseteq D$ ,	$B \sqsubseteq E$

- Input axiom
  - $\exists r.A \sqcap \exists r.\exists s.A \sqsubseteq A \sqcap B$
- Normalisation
  - 1.  $\exists r.A \sqcap \exists r.\exists s.A \sqsubseteq A0, A0 \sqsubseteq A \sqcap B (NF0)$
  - 2.  $\exists r.A \sqsubseteq A1, A1 \sqcap \exists r.\exists s.A \sqsubseteq A0 (NF1I)$
  - 3.  $\exists r. \exists s. A \sqsubseteq A2, A1 \sqcap A2 \sqsubseteq A0 (NF1r)$
  - 4. ∃s.A ⊑ A3, ∃r.A3 ⊑ A2 (NF2)
  - 5.  $A0 \sqsubseteq A, A0 \sqsubseteq B (NF4)$



## **Conservative Extension**



- Given two EL TBoxes T1 and T2, T2 is a conservative extension of T1 if
  - Sig(T1) ⊆Sig (T2)
  - every model of T2 is a model of T1
  - for every model I1 of T1, there exists a model I2 of T2 such as I1 and I2 coincide on sig(T1) U T, i.e.,
    - $\Delta^{II} = \Delta^{I2}$
    - $A^{I1} = A^{I2}$  for every named class in  $A \in Sig(T1)$ , and
    - $r^{I1} = r^{I2}$  for every named property in  $r \in Sig(T1)$



## **Conservative Extension and EL**



 Given two EL TBoxes T1 and T2, such that T2 is a conservative extension of T1, and C, D are EL class descriptions containing only class and property names from Sig(T1)

- Then T1 ⊨ C  $\sqsubseteq$  D iff T2 ⊨ C  $\sqsubseteq$  D

- Given two EL TBoxes T1 and T2, such that T2 is the normalised TBox obtained from T1
  - Then T2 is a conservative extension of T1
  - T2 is linear in the size of T1

## **Classification Procedure**



EDINBURGH

- We assume that the input TBox axioms are all in normal form
  - The overall number of the normalised GCIs is polynomial in the size of the TBox
- Idea
  - start from the inputs GCIs and add implied GCIs using classification rules



• To get the concrete we need to



- Rule application
  - T' start as the TBox
  - If axioms appear on top of the line are in T', then add the axioms below into T' (unless they are already in)

$$CR1 \quad \overline{A \sqsubseteq A} \qquad CR2 \quad \overline{A \sqsubseteq \top}$$

$$CR3 \quad \frac{A_1 \sqsubseteq A_2 \quad A_2 \sqsubseteq A_3}{A_1 \sqsubseteq A_3} \qquad CR4 \quad \frac{A \sqsubseteq A_1 \quad A \sqsubseteq A_2 \quad A_1 \sqcap A_2 \sqsubseteq B}{A \sqsubseteq B}$$

$$CR5 \quad \frac{A \sqsubseteq \exists r.A_1 \quad A_1 \sqsubseteq B_1 \quad \exists r.B_1 \sqsubseteq B}{A \sqsubseteq B}$$

[credit: F Baader]



## **Example: Classification Rules**



- $\mathcal{T}_1 = \{ A \sqsubseteq \exists r.A, \\ \exists r.B \sqsubseteq B_1, \\ \top \sqsubseteq B, \\ A \sqsubseteq B_2, \\ B_1 \sqcap B_2 \sqsubseteq C \}$
- 1.  $A \sqsubseteq A, B \sqsubseteq B, B1 \sqsubseteq B1, B2 \sqsubseteq B2, C \sqsubseteq C (CR1)$
- 2.  $A \sqsubseteq T, B1 \sqsubseteq T, B2 \sqsubseteq T, C \sqsubseteq T, B \sqsubseteq T (CR2)$
- 3.  $A \sqsubseteq \top$ ,  $\top \sqsubseteq B \Rightarrow A \sqsubseteq B$  (CR3)
- 4. B1 ⊑ ⊤, ⊤ ⊑ B => B1 ⊑ B (CR3)
- 5.  $B2 \sqsubseteq \top$ ,  $\top \sqsubseteq B \Rightarrow B2 \sqsubseteq B$  (CR3)
- 6.  $C \sqsubseteq \top$ ,  $\top \sqsubseteq B \Rightarrow C \sqsubseteq B$  (CR3)
- 7.  $A \sqsubseteq \exists r.A, A \sqsubseteq B, \exists r.B \sqsubseteq B1 \Rightarrow A \sqsubseteq B1$  (CR5)
- 8.  $A \sqsubseteq B1, A \sqsubseteq B2, B1 \sqcap B2 \sqsubseteq C \Rightarrow A \sqsubseteq C (CR4)$





#### **Lecture Outline**

- Motivation
- Overview of EL and reasoning
- Detailed Discussions on EL and reasoning
- Practical



# **Subsumption Checking**

Subsumption checking between two class descirptions C ⊑ D can be reduced to that between two named classes A1 ⊑ A2



- More precisely
  - An EL TBox T |= C  $\sqsubseteq$  D iff T U {A1  $\sqsubseteq$  C, D  $\sqsubseteq$  A2} |= A1  $\sqsubseteq$  A2



## **Example: Subsumption Checking**

$$\begin{array}{c} \mathsf{CR1} & \overline{A \sqsubseteq A} & \mathsf{CR2} & \overline{A \sqsubseteq \top} \\ \mathsf{CR3} & \frac{A_1 \sqsubseteq A_2 & A_2 \sqsubseteq A_3}{A_1 \sqsubseteq A_3} & \mathsf{CR4} & \frac{A \sqsubseteq A_1 & A \sqsubseteq A_2 & A_1 \sqcap A_2 \sqsubseteq B}{A \sqsubseteq B} \\ \\ \mathsf{CR5} & \frac{A \sqsubseteq \exists r.A_1 & A_1 \sqsubseteq B_1 & \exists r.B_1 \sqsubseteq B}{A \sqsubseteq B} \end{array}$$

 $\mathcal{T}_1 = \{ A \sqsubseteq \exists r.A, \\ \exists r.B \sqsubseteq B_1, \\ \top \sqsubseteq B, \\ A \sqsubseteq B_2, \\ B_1 \sqcap B_2 \sqsubseteq C \}$ 

Question: Check if A  $\square$  C  $\sqsubseteq$   $\exists$ r.B holds

- 1. Extend the KB with  $\{A' \sqsubseteq A \sqcap C, \exists r.B \sqsubseteq B'\}$ , which is normalised as  $\{A' \sqsubseteq A, A' \sqsubseteq C, \exists r.B \sqsubseteq B'\}$  (NF4)
- 2.  $A \sqsubseteq A, B \sqsubseteq B, A' \sqsubseteq A', B' \sqsubseteq B', B1 \sqsubseteq B1, B2 \sqsubseteq B2, C \sqsubseteq C (CR1)$
- 3.  $A \sqsubseteq T, A' \sqsubseteq T B1 \sqsubseteq T, B2 \sqsubseteq T, C \sqsubseteq T, B \sqsubseteq T, B' \sqsubseteq T (CR2)$
- 4.  $A \sqsubseteq T, T \sqsubseteq B \Rightarrow A \sqsubseteq B (CR3)$
- 5.  $B1 \sqsubseteq T, T \sqsubseteq B \Rightarrow B1 \sqsubseteq B (CR3)$
- 6.  $B2 \sqsubseteq T, T \sqsubseteq B \Rightarrow B2 \sqsubseteq B (CR3)$
- 7.  $C \sqsubseteq T, T \sqsubseteq B \Rightarrow C \sqsubseteq B (CR3)$
- 8.  $A \sqsubseteq \exists r.A, A \sqsubseteq B, \exists r.B \sqsubseteq B1 \Rightarrow A \sqsubseteq B1 (CR5)$
- 9.  $A \sqsubseteq \exists r.A, A \sqsubseteq B, \exists r.B \sqsubseteq B' \Rightarrow A \sqsubseteq B'$  (CR5)
- 10. A'  $\sqsubseteq$ A, A  $\sqsubseteq$ B' =>A'  $\sqsubseteq$ B' (CR3)
- 11. Since A'  $\sqsubseteq$ B' holds, we have A  $\sqcap$  C  $\sqsubseteq \exists$ r.B

Knowledge Graphs Jeff Z. Pan

## **EL Family**

•



ſHE UNIVERSITY ∂f EDINBURGH

- EL+ extends EL with
  - property chain inclusion: r1 o …o rk  $\sqsubseteq$  r
  - concrete domain (n-ary dataype predicate):
     D(f1,...fn)
- EL++ extends EL+ with
  - the bottom class:  $\perp$
  - norminal: {a}



### **Classification: OWL 2 EL vs OWL 2 DL**



- OWL 2 DL
  - subsumption checking is N2EXPTime-Complete
  - GCI-rule is expensive
  - many new optimisations but still challenging when there are large number of classes (SNOMED CT has over 300K)
- OWL 2 EL
  - Batch mode
  - Good base for approximation (such as those used by the TrOWL reasoner)

## **Conjunctive Queries**

EDINBURGH

- A conjunctive query  $q(\overrightarrow{x})$  has the form
  - $\exists y1, \dots, ym.(\alpha 1 \land \dots \land \alpha n)$ , where m>=0, n>=1
  - each atom αi is a concept atom A(x) or a property atom r(x,y)
  - y1,...,ym are called quantified variables
  - quantified variables that appear only in one atom are called unbounded variables
- CQs without constants are called pure CQs
- CQs can be reduced to pure CQs in polynomial time
- An FO query is called a Boolean query if its arity is 0.

Knowledge Graphs Jeff Z. Pan



# **Example: Conjunctive Queries**



- Assuming that we have three tables Professor, supervises and Student
- Return all pairs of supervisors and students
  - q1(x1,x2) = Professor(x1) ∧ supervise(x1,x2) ∧ Student(x2)
  - also written as q1(x1,x2) <- Professor(x1) ∧ supervise(x1,x2) ∧ Student(x2)
- Return all students whom are supervised by some professors
  - q2(x) = ∃y.Professor(y) ∧ supervises(y,x) ∧
     Student(x)



# **Ontology Based QA: Example 1**



We assume that each concept/relationship of the ontology is mapped directly to a database table.

But the database tables may be **incompletely specified**, or even missing for some concepts/relationships.

```
DB: Coordinator \supseteq { serge, marie }

Project \supseteq { webdam, diadem }

worksFor \supseteq { (serge,webdam), (georg,diadem) }

Query: q(x) \leftarrow Researcher(x)

Answer: { serge, marie, georg }

[credit: G Xiao] Knowledge Graphs

Jeff Z. Pan
```



# **Ontology Based QA : Example 2**



Each person has a father, who is a person.

DB: Person ⊇ { john, nick, toni }
hasFather ⊇ { (john,nick), (nick,toni) }

Queries:  $q_1(x, y) \leftarrow hasFather(x, y)$   $q_2(x) \leftarrow \exists y$ . hasFather(x, y)  $q_3(x) \leftarrow \exists y_1, y_2, y_3$ . hasFather $(x, y_1) \land hasFather(y_1, y_2) \land hasFather(y_2, y_3)$  $q_4(x, y_3) \leftarrow \exists y_1, y_2$ . hasFather $(x, y_1) \land hasFather(y_1, y_2) \land hasFather(y_2, y_3)$ 

Answers: to  $q_1$ : { (john,nick), (nick,toni) } to  $q_2$ : { john, nick, toni } to  $q_3$ : { john, nick, toni } to  $q_4$ : { } [credit: G Xiao]



## **Ontology Based QA : Example 3**





#### **Lecture Outline**

- Motivation: efficient and scalable reasoning
- Introduction: the EL description logic
- Focus: subsumption checking in EL
- Tutorial
  - Normailisation
  - Classification
  - Subsumption