

# Description Logics – Querying

Petr Křemen

`petr.kremen@fel.cvut.cz`

December 13, 2018



# Outline

- 1 What if OWL is not enough?
- 2 Complex Queries
  - Evaluation of Conjunctive Queries in  $\mathcal{ALC}$
- 3 Modeling Error Explanation
  - Black-box methods
  - Algorithms based on CS-trees
  - Algorithm based on Reiter's Algorithm
  - Algorithm based on Reiter's Algorithm



# Problems

- What if OWL is not enough?
- What if more complex queries than consistency checking are necessary?
- What to do if an ontology is inconsistent?



- 1 What if OWL is not enough?
- 2 Complex Queries
  - Evaluation of Conjunctive Queries in  $\mathcal{ALC}$
- 3 Modeling Error Explanation
  - Black-box methods
  - Algorithms based on CS-trees
  - Algorithm based on Reiter's Algorithm
  - Algorithm based on Reiter's Algorithm

# What if OWL is not enough?



# SROIQ (OWL) Revision

$Man \sqsubseteq Person$

$Man \sqsubseteq \neg Woman$

$Man \sqcap \exists hasChild \cdot Man \sqsubseteq FatherOfSons$

$hasSon \sqsubseteq hasChild$

$hasParent \circ hasBrother \sqsubseteq hasUncle$

$trans(hasDescendant)$

$sym(hasSpouse)$

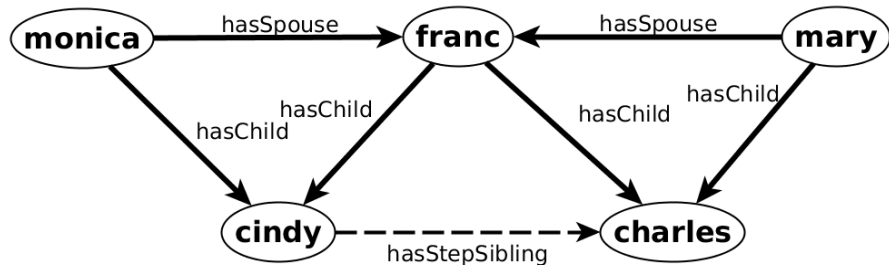
$fun(hasMother)$

$hasWife \sqsubseteq hasHusband^{-}$

## How to express hasStepSibling?



# How to express hasStepSibling?



## How to express `hasStepSibling`?

$$\begin{aligned}
 & \text{hasSpouse}(?m1, ?f), \text{hasSpouse}(?m2, ?f), \\
 & \text{hasChild}(?m1, ?c1), \text{hasChild}(?m2, ?c2), \\
 & \text{hasChild}(?f, ?c1), \text{hasChild}(?f, ?c2), ?c1 \neq ?c2 \\
 & \qquad \qquad \qquad \rightarrow \text{hasStepSibling}(?c1, ?c2)
 \end{aligned}$$


# OWL2-DL + rules undecidable

... unless variables in rules are restricted to match named individuals only.

## DL-safe Rules

A rule is DL-safe, if its variables are *distinguished*, i.e. they can only match **named individuals** in the ontology. Consistency checking of OWL2-DL + DL-safe rules is decidable.





1 What if OWL is not enough?

2 **Complex Queries**

- Evaluation of Conjunctive Queries in  $\mathcal{ALC}$

3 Modeling Error Explanation

- Black-box methods
- Algorithms based on CS-trees
- Algorithm based on Reiter's Algorithm
- Algorithm based on Reiter's Algorithm

# Complex Queries



## What if we need to answer a complex query?

- Consistency checking is not enough. What if we would like to ask more, e.g. ... **How many czech writers died in the Czech Republic according to DBPedia ?**

```

PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dcterms: <http://purl.org/dc/terms/>
SELECT COUNT (?x)
{
  ?x    dbo:deathPlace    dbr:Czech_Republic ;
        dcterms:subject  dbr:Category:Czech_writers
}

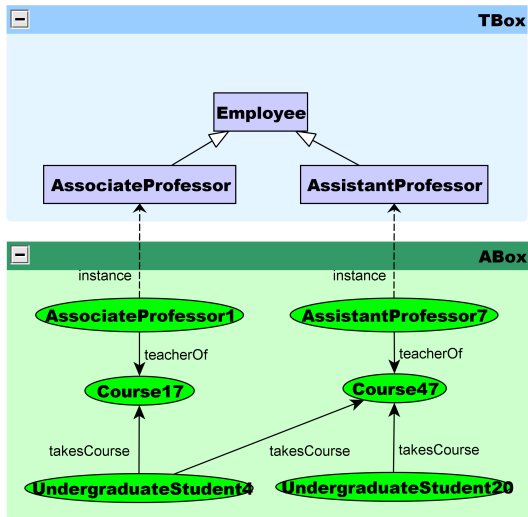
```

at the following endpoint:

<http://dbpedia-live.openlinksw.com/sparql/>

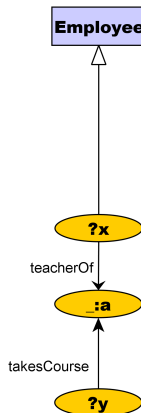


# Conjunctive Queries

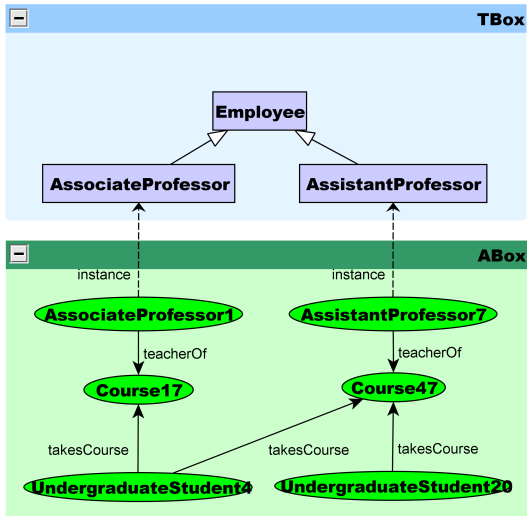


## Conjunctive ABox Queries.

"Get all teachers and their students."



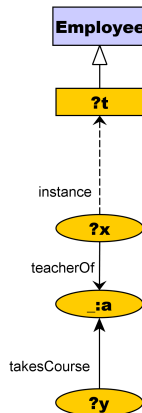
# Metaqueries



## Mixed TBox/ABox Queries.

"Get all teachers and their students."

"... together with the type of the teachers."



# Query Types

**Conjunctive (ABox) queries** – queries asking for individual tuples complying with a graph-like pattern.



## Query Types

**Conjunctive (ABox) queries** – queries asking for individual tuples complying with a graph-like pattern.

### Example

“Find all mothers and their daughters having at least one brother.” :

$$Q(?x, ?z) \leftarrow \text{Woman}(?x), \text{hasChild}(?x, ?y), \text{hasChild}(?x, ?z), \\ \text{Man}(?y), \text{Woman}(?z)$$



## Query Types

**Conjunctive (ABox) queries** – queries asking for individual tuples complying with a graph-like pattern.

### Example

“Find all mothers and their daughters having at least one brother.” :

$$Q(?x, ?z) \leftarrow \text{Woman}(?x), \text{hasChild}(?x, ?y), \text{hasChild}(?x, ?z), \\ \text{Man}(?y), \text{Woman}(?z)$$

**Metaqueries** – queries asking for individual/concept/role tuples. There are several languages for metaqueries, e.g. SPARQL-DL, OWL-SAIQL, etc.



## Query Types

**Conjunctive (ABox) queries** – queries asking for individual tuples complying with a graph-like pattern.

### Example

“Find all mothers and their daughters having at least one brother.” :

$$Q(?x, ?z) \leftarrow \text{Woman}(?x), \text{hasChild}(?x, ?y), \text{hasChild}(?x, ?z), \\ \text{Man}(?y), \text{Woman}(?z)$$

**Metaqueries** – queries asking for individual/concept/role tuples. There are several languages for metaqueries, e.g. SPARQL-DL, OWL-SAIQL, etc.

### Example

“Find all people together with their type.” in SPARQL-DL:

$$Q(?x, ?c) \leftarrow \text{TYPE}(?x, ?c), \text{SUBCLASSOF}(?c, \text{Person})$$



## Conjunctive (ABox) queries

Conjunctive (ABox) queries are analogous to database SELECT-PROJECT-JOIN queries.

### Conjunctive Query

$$Q(?x_1, \dots, ?x_D) \leftarrow t_1, \dots, t_T,$$

where each  $t_i$  is either

- $C(y_k)$  (where  $C$  is a concept)
- $R(y_k, y_l)$  (where  $R$  is a role)

and  $y_i$  is either (i) an individual, or (ii) variable from a new set  $V$  (variables will be differentiated from individuals by the prefix “?”). We need all  $?x_i$  to be present also in one of  $t_i$ .



## Conjunctive ABox Queries – Semantics

- Conjunctive queries of the form  $Q()$  are called *boolean* – such queries only test existence of a relational structure in each model  $\mathcal{I}$  of the ontology  $\mathcal{K}$ .
- Consider any interpretation  $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ . *Evaluation*  $\eta$  is a function from the set of individuals and variables into  $\Delta^{\mathcal{I}}$  that coincides with  $\mathcal{I}$  on individuals.
- Then  $\mathcal{I} \models_{\eta} Q()$ , iff
  - $\eta(y_k) \in C^{\mathcal{I}}$  for each atom  $C(y_k)$  from  $Q()$  and
  - $\langle \eta(y_k), \eta(y_l) \rangle \in R^{\mathcal{I}}$  for each atom  $R(y_k, y_l)$  from  $Q()$
- Interpretation  $\mathcal{I}$  is a model of  $Q()$ , iff  $\mathcal{I} \models_{\eta} Q()$  for some  $\eta$ .
- Next,  $\mathcal{K} \models Q()$  ( $Q()$  is satisfiable in  $\mathcal{K}$ ) iff  $\mathcal{I} \models Q()$  whenever  $\mathcal{I} \models \mathcal{K}$



## Conjunctive ABox Queries – Variables

- Queries without variables are not practically interesting. For queries with variables we define semantics as follows. An N-tuple  $\langle i_1, \dots, i_n \rangle$  is a *solution* to  $Q(?x_1, \dots, ?x_n)$  in theory  $\mathcal{K}$ , whenever  $\mathcal{K} \models Q'()$ , for a boolean query  $Q'$  obtained from  $Q$  by replacing all occurrences of  $?x_1$  in all  $t_k$  by an individual  $i_1$ , etc.



## Conjunctive ABox Queries – Variables

- Queries without variables are not practically interesting. For queries with variables we define semantics as follows. An N-tuple  $\langle i_1, \dots, i_n \rangle$  is a *solution* to  $Q(?x_1, \dots, ?x_n)$  in theory  $\mathcal{K}$ , whenever  $\mathcal{K} \models Q'()$ , for a boolean query  $Q'$  obtained from  $Q$  by replacing all occurrences of  $?x_k$  in all  $t_k$  by an individual  $i_1$ , etc.
- In conjunctive queries two types of variables can be defined:



## Conjunctive ABox Queries – Variables

- Queries without variables are not practically interesting. For queries with variables we define semantics as follows. An  $N$ -tuple  $\langle i_1, \dots, i_n \rangle$  is a *solution* to  $Q(?x_1, \dots, ?x_n)$  in theory  $\mathcal{K}$ , whenever  $\mathcal{K} \models Q'()$ , for a boolean query  $Q'$  obtained from  $Q$  by replacing all occurrences of  $?x_1$  in all  $t_k$  by an individual  $i_1$ , etc.
- In conjunctive queries two types of variables can be defined:
  - distinguished** occur in the query head as well as body, e.g.  $?x, ?z$  in the previous example. These variables are evaluated as domain elements that are necessarily interpretations of some individual from  $\mathcal{K}$ . That individual is the binding to the distinguished variable in the query result.



## Conjunctive ABox Queries – Variables

- Queries without variables are not practically interesting. For queries with variables we define semantics as follows. An N-tuple  $\langle i_1, \dots, i_n \rangle$  is a *solution* to  $Q(?x_1, \dots, ?x_n)$  in theory  $\mathcal{K}$ , whenever  $\mathcal{K} \models Q'()$ , for a boolean query  $Q'$  obtained from  $Q$  by replacing all occurrences of  $?x_1$  in all  $t_k$  by an individual  $i_1$ , etc.
- In conjunctive queries two types of variables can be defined:
  - distinguished** occur in the query head as well as body, e.g.  $?x, ?z$  in the previous example. These variables are evaluated as domain elements that are necessarily interpretations of some individual from  $\mathcal{K}$ . That individual is the binding to the distinguished variable in the query result.
  - undistinguished** occur only in the query body, e.g.  $?y$  in the previous example. Their can be interpreted as any domain elements.



# Conjunctive Queries – Examples

## Example

Let's have a theory  $\mathcal{K}_4 = (\emptyset, \{(\exists R_1 \cdot C_1)(i_1), R_2(i_1, i_2), C_2(i_2)\})$ .

- Does  $\mathcal{K} \models Q_1()$  hold for  $Q_1() \leftarrow R_1(?x_1, ?x_2)$  ?
- What are the solutions of the query  $Q_2(?x_1) \leftarrow R_1(?x_1, ?x_2)$  for  $\mathcal{K}$  ?
- What are the solutions of the query  $Q_3(?x_1, ?x_2) \leftarrow R_1(?x_1, ?x_2)$  for  $\mathcal{K}$  ?



# Evaluation of Conjunctive Queries in $\mathcal{ALC}$

1 What if OWL is not enough?

2 **Complex Queries**

- Evaluation of Conjunctive Queries in  $\mathcal{ALC}$

3 **Modeling Error Explanation**

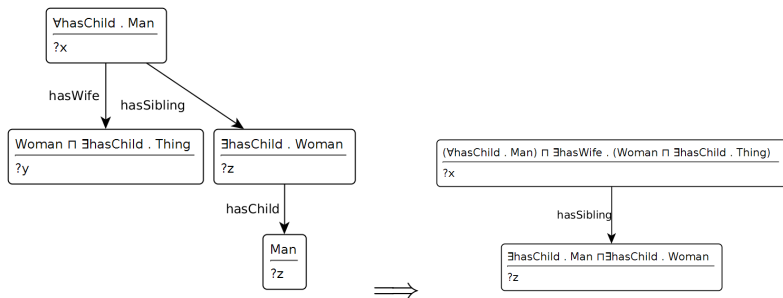
- Black-box methods
- Algorithms based on CS-trees
- Algorithm based on Reiter's Algorithm
- Algorithm based on Reiter's Algorithm





# Satisfiability of $\mathcal{ALC}$ Boolean Queries

- Satisfiability of the boolean query  $Q()$  having a tree shape can be checked by means of the **rolling-up technique**.



## Rolling-up Technique

- Each two atoms  $C_1(y_k)$  and  $C_2(y_k)$  can be replaced by a single query atom of the form  $(C_1 \sqcap C_2)(y_k)$ .



# Rolling-up Technique

- Each two atoms  $C_1(y_k)$  and  $C_2(y_k)$  can be replaced by a single query atom of the form  $(C_1 \sqcap C_2)(y_k)$ .
- Each query atom of the form  $R(y_k, y_l)$  can be replaced by the term  $(\exists R \cdot X)(y_k)$ , if  $y_l$  occurs in at most one other query atom of the form  $C(y_l)$  (if there is no  $C(y_l)$  atom in the query, consider w.l.o.g. that  $C$  is  $\top$ ).  $X$  equals to



## Rolling-up Technique

- Each two atoms  $C_1(y_k)$  and  $C_2(y_k)$  can be replaced by a single query atom of the form  $(C_1 \sqcap C_2)(y_k)$ .
- Each query atom of the form  $R(y_k, y_l)$  can be replaced by the term  $(\exists R \cdot X)(y_k)$ , if  $y_l$  occurs in at most one other query atom of the form  $C(y_l)$  (if there is no  $C(y_l)$  atom in the query, consider w.l.o.g. that  $C$  is  $\top$ ).  $X$  equals to
  - (i)  $C$ , whenever  $y_l$  is a variable,



## Rolling-up Technique

- Each two atoms  $C_1(y_k)$  and  $C_2(y_k)$  can be replaced by a single query atom of the form  $(C_1 \sqcap C_2)(y_k)$ .
- Each query atom of the form  $R(y_k, y_l)$  can be replaced by the term  $(\exists R \cdot X)(y_k)$ , if  $y_l$  occurs in at most one other query atom of the form  $C(y_l)$  (if there is no  $C(y_l)$  atom in the query, consider w.l.o.g. that  $C$  is  $\top$ ).  $X$  equals to
  - (i)  $C$ , whenever  $y_l$  is a variable,
  - (ii)  $C \sqcap Y_l$ , whenever  $y_l$  is an individual.  $Y_l$  is a *representative concept* of individual  $y_l$  occurring neither in  $\mathcal{K}$  nor in  $Q$ . For each  $y_l$  it is necessary to extend ABox of  $\mathcal{K}$  with concept assertion  $Y_l(y_l)$ .



## Satisfiability of $\mathcal{ALC}$ Boolean Queries (2)

... after rolling-up the query we obtain the query  $Q'() \leftarrow C(y)$ , that is satisfied in  $\mathcal{K}$ , iff  $Q()$  is satisfied in  $\mathcal{K}$ :

- **If  $y$  is an individual, then  $Q'()$  is satisfied, whenever  $\mathcal{K} \models C(y)$  (i.e.  $\mathcal{K} \cup \{(\neg C)(y)\}$  is inconsistent)**

### Example

Consider a query  $Q_4() \leftarrow R_1(?x_1, ?x_2), R_2(?x_1, ?x_3), C_2(?x_3)$ . This query can be rolled-up into the query  $Q'_4 \leftarrow (\exists R_1 \cdot \top \sqcap \exists R_2 \cdot C_2)(?x_1)$ . This query is satisfiable in  $\mathcal{K}_4$ , as  $\mathcal{K}_4 \cup \{(\exists R_1 \cdot \top \sqcap \exists R_2 \cdot C_2) \sqsubseteq \perp\}$  is inconsistent.



## Satisfiability of $\mathcal{ALC}$ Boolean Queries (2)

... after rolling-up the query we obtain the query  $Q'() \leftarrow C(y)$ , that is satisfied in  $\mathcal{K}$ , iff  $Q()$  is satisfied in  $\mathcal{K}$ :

- **If  $y$  is an individual, then  $Q'()$  is satisfied, whenever  $\mathcal{K} \models C(y)$  (i.e.  $\mathcal{K} \cup \{(\neg C)(y)\}$  is inconsistent)**
- **If  $y$  is a variable, then  $Q'()$  is satisfied, whenever  $\mathcal{K} \cup \{C \sqsubseteq \perp\}$  is inconsistent. Why ?**

### Example

Consider a query  $Q_4() \leftarrow R_1(?x_1, ?x_2), R_2(?x_1, ?x_3), C_2(?x_3)$ . This query can be rolled-up into the query  $Q'_4 \leftarrow (\exists R_1 \cdot \top \sqcap \exists R_2 \cdot C_2)(?x_1)$ . This query is satisfiable in  $\mathcal{K}_4$ , as  $\mathcal{K}_4 \cup \{(\exists R_1 \cdot \top \sqcap \exists R_2 \cdot C_2) \sqsubseteq \perp\}$  is inconsistent.



## Satisfiability of Boolean Queries in $\mathcal{ALC}$ (3)

... and what to do with queries with distinguished variables ?

- Let's consider just queries that form “connected component” and contain for some variable  $y_k$  at least two query atoms of the form  $R_1(y_1, y_k)$  and  $R_2(y_2, y_k)$ .





## Satisfiability of Boolean Queries in $\mathcal{ALC}$ (3)

... and what to do with queries with distinguished variables ?

- Let's consider just queries that form “connected component” and contain for some variable  $y_k$  at least two query atoms of the form  $R_1(y_1, y_k)$  and  $R_2(y_2, y_k)$ .
- Question: *Why is it enough to take just one connected component?*



## Satisfiability of Boolean Queries in $\mathcal{ALC}$ (3)

... and what to do with queries with distinguished variables ?

- Let's consider just queries that form “connected component” and contain for some variable  $y_k$  at least two query atoms of the form  $R_1(y_1, y_k)$  and  $R_2(y_2, y_k)$ .
- Question: *Why is it enough to take just one connected component?*
- **Let's make use of the tree model property of  $\mathcal{ALC}$ . Each pair of atoms  $R_1(y_1, y_k)$  and  $R_2(y_2, y_k)$  can be satisfied only if  $y_k$  is interpreted as a domain element, that is an interpretation of an individual –  $y_k$  can be treated as distinguished. Why (see next slide) ?**



## Satisfiability of Boolean Queries in $\mathcal{ALC}$ (3)

... and what to do with queries with distinguished variables ?

- Let's consider just queries that form “connected component” and contain for some variable  $y_k$  at least two query atoms of the form  $R_1(y_1, y_k)$  and  $R_2(y_2, y_k)$ .
- Question: *Why is it enough to take just one connected component?*
- **Let's make use of the tree model property of  $\mathcal{ALC}$ . Each pair of atoms  $R_1(y_1, y_k)$  and  $R_2(y_2, y_k)$  can be satisfied only if  $y_k$  is interpreted as a domain element, that is an interpretation of an individual –  $y_k$  can be treated as distinguished. Why (see next slide) ?**
- For  $\mathit{SHOIN}$  and  $\mathit{SROIQ}$  there is no sound and complete decision procedure for general boolean queries.





## Queries with Distinguished Variables – naive pruning

Consider arbitrary query  $Q(?x_1, \dots, ?x_D)$ . How to evaluate it ?

- **naive way:** Replace each distinguished variable  $x_i$  with each individual occurring in  $\mathcal{K}$ . *Solutions* are those D-tuples  $\langle i_1, \dots, i_D \rangle$ , for which a boolean query created from  $Q$  by replacing each  $x_k$  with  $i_k$  is satisfiable.



## Queries with Distinguished Variables – naive pruning

Consider arbitrary query  $Q(?x_1, \dots, ?x_D)$ . How to evaluate it ?

- **naive way:** Replace each distinguished variable  $x_i$  with each individual occurring in  $\mathcal{K}$ . *Solutions* are those D-tuples  $\langle i_1, \dots, i_D \rangle$ , for which a boolean query created from  $Q$  by replacing each  $x_k$  with  $i_k$  is satisfiable.

Remind that  $\mathcal{K}_4 = (\emptyset, \{(\exists R_1 \cdot C_1)(i_1), R_2(i_1, i_2), C_2(i_2)\})$ . The query

$$Q_5(?x_1) \leftarrow R_1(?x_1, ?x_2), R_2(?x_1, ?x_3), C_2(?x_3)$$

has *solution*  $\langle i_1 \rangle$  as

$$Q'_5() \leftarrow R_1(i_1, ?x_2), R_2(i_1, ?x_3), C_2(?x_3)$$

can be rolled into  $Q''_5()$  for which  $\mathcal{K}_4 \models Q''_5$ :

$$Q''_5() \leftarrow (\exists R_1 \cdot \top \sqcap \exists R_2 \cdot C_2)(i_1)$$

## Queries with Distinguished Variables – naive pruning

... another example

The query

$$Q_6(?x_1, ?x_3) \leftarrow R_1(?x_1, ?x_2), R_2(?x_1, ?x_3), C_2(?x_3)$$

has *solution*  $\langle i_1, i_2 \rangle$  as

$$Q'_6() \leftarrow R_1(i_1, ?x_2), R_2(i_1, i_2), C_2(i_2)$$

can be rolled into  $Q''_6$  for which  $\mathcal{K}_4 \cup \{\mathbf{I}_2(i_2)\} \models Q''_6$ .

$$Q''_6() \leftarrow (\exists R_1 \cdot \top \sqcap \exists R_2 \cdot (C_2 \sqcap I_2))(i_1).$$

Similarly  $Q_7(?x_1, ?x_2) \leftarrow R_1(?x_1, ?x_2), R_2(?x_1, ?x_3), C_2(?x_3)$  has no solution.

## Queries with Distinguished Variables – iterative pruning

- ... a bit more clever strategy than replacing all variables: First, let's replace just the first variable  $?x_1$  with each individual from  $\mathcal{K}$ , resulting in  $Q_2$ . If the subquery of  $Q_2$  containing all query atoms from  $Q_2$  without distinguished variables is not a logical consequence of  $\mathcal{K}$ , then we do not need to test potential bindings for other variables.





## Queries with Distinguished Variables – iterative pruning

- ... a bit more clever strategy than replacing all variables: First, let's replace just the first variable  $?x_1$  with each individual from  $\mathcal{K}$ , resulting in  $Q_2$ . If the subquery of  $Q_2$  containing all query atoms from  $Q_2$  without distinguished variables is not a logical consequence of  $\mathcal{K}$ , then we do not need to test potential bindings for other variables.
- Many other optimizations are available.



## Queries with Distinguished Variables – iterative pruning

For the query  $Q_6(?x_1, ?x_3)$ , the naive strategy needs to check four different bindings (resulting in four tableau algorithm runs)

$$\langle i_1, i_1 \rangle,$$

$$\langle \mathbf{i}_1, \mathbf{i}_2 \rangle,$$

$$\langle i_2, i_1 \rangle,$$

$$\langle i_2, i_2 \rangle.$$

Out of them only  $\langle i_1, i_2 \rangle$  is a solution for  $Q_6$ . Consider only partial binding  $\langle i_2 \rangle$  for  $?x_1$ . Applying this binding to  $Q_6$  we get  $Q_7(?x_3) = R_1(i_2, ?x_2), R_2(i_2, ?x_3), C_2(?x_3)$ . Its distinguished-variable-free subquery is  $Q'_7() = R_1(i_2, ?x_2)$  and  $\mathcal{K}_4 \not\models Q'_7$ . Because of **monotonicity** of  $\mathcal{ALC}$ , we do not need to check the two bindings for  $?x_3$  in this case which saves us one tableau algorithm run.

# Modeling Error Explanation

Petr Křemen

[petr.kremen@fel.cvut.cz](mailto:petr.kremen@fel.cvut.cz)

December 13, 2018



# Outline

- 1 What if OWL is not enough?
- 2 Complex Queries
  - Evaluation of Conjunctive Queries in  $\mathcal{ALC}$
- 3 Modeling Error Explanation
  - Black-box methods
  - Algorithms based on CS-trees
  - Algorithm based on Reiter's Algorithm
  - Algorithm based on Reiter's Algorithm



- 1 What if OWL is not enough?
- 2 Complex Queries
  - Evaluation of Conjunctive Queries in  $\mathcal{ALC}$
- 3 Modeling Error Explanation
  - Black-box methods
  - Algorithms based on CS-trees
  - Algorithm based on Reiter's Algorithm
  - Algorithm based on Reiter's Algorithm

# Modeling Error Explanation



# Motivation

- When an inference engine claims inconsistency of an ( $\mathcal{ALC}$ ) theory/unsatisfiability of an ( $\mathcal{ALC}$ ) concept, **what can we do with it** ?



# Motivation

- When an inference engine claims inconsistency of an ( $\mathcal{ALC}$ ) theory/unsatisfiability of an ( $\mathcal{ALC}$ ) concept, **what can we do with it** ?
- We can start iterating through all axioms in the theory and look, “what went wrong”.



# Motivation

- When an inference engine claims inconsistency of an ( $\mathcal{ALC}$ ) theory/unsatisfiability of an ( $\mathcal{ALC}$ ) concept, **what can we do with it** ?
- We can start iterating through all axioms in the theory and look, “what went wrong”.
- ... but hardly in case we have **hundred thousand axioms**





# Motivation

- When an inference engine claims inconsistency of an ( $\mathcal{ALC}$ ) theory/unsatisfiability of an ( $\mathcal{ALC}$ ) concept, **what can we do with it** ?
- We can start iterating through all axioms in the theory and look, “what went wrong”.
- ... but hardly in case we have **hundred thousand axioms**
- A solution might be to ask the computer to *localize the axioms causing the problem for us*.





## MUPS – example

Minimal unsatisfiability preserving subterminology (MUPS) is a minimal set of axioms responsible for concept unsatisfiability.



## MUPS – example

Minimal unsatisfiability preserving subterminology (MUPS) is a minimal set of axioms responsible for concept unsatisfiability.



# MUPS – example

Minimal unsatisfiability preserving subterminology (MUPS) is a minimal set of axioms responsible for concept unsatisfiability.

## Example

Consider theory  $\mathcal{K}_5 = (\{\alpha_1, \alpha_2, \alpha_3\}, \emptyset)$

$\alpha_1$  :  $Person \sqsubseteq \exists hasParent \cdot (Man \sqcap Woman) \sqcap \forall hasParent \cdot \neg Person,$

$\alpha_2$  :  $Man \sqsubseteq \neg Woman,$

$\alpha_3$  :  $Man \sqcup Woman \sqsubseteq Person.$

Unsatisfiability of *Person* comes independently from two axiom sets (MUPSes), namely  $\{\alpha_1, \alpha_2\}$  and  $\{\alpha_1, \alpha_3\}$ . Check it yourself !



# MUPS

Currently two approaches exist for searching all MUPSeS for given concept:

**black-box methods** perform many satisfiability tests using existing inference engine.

- ☺ flexible and easily reusable for another (description) logic
- ☹ time consuming

**glass-box methods** all integrated into an existing reasoning (typically tableau) algorithm.

- ☺ efficient
- ☹ hardly reusable for another (description) logic.



## Glass-box methods

- For  $\mathcal{ALC}$  there exists a complete algorithm with the following idea:
  - tableau algorithm for  $\mathcal{ALC}$  is extended in such way that it “remembers which axioms were used during completion graph construction”.
  - for each completion graph containing a clash, the axioms that were used during its construction can be transformed into a MUPS.
- Unfortunately, complete glass-box methods do not exist for OWL-DL and OWL2-DL. The same idea (tracking axioms used during completion graph construction) can be used also for these logics, but only as a preprocessing reducing the set of axioms used by a black-box algorithm.



# Black-box methods

1 What if OWL is not enough?

2 Complex Queries

- Evaluation of Conjunctive Queries in  $\mathcal{ALC}$

3 Modeling Error Explanation

- **Black-box methods**
- Algorithms based on CS-trees
- Algorithm based on Reiter's Algorithm
- Algorithm based on Reiter's Algorithm





# Task formulation

- Let's have a *set of axioms*  $X$  of given DL and *reasoner*  $R$  for given DL. We want to find MUPSeS for :
  - 1 concept unsatisfiability, '
  - 2 theory (ontology) inconsistency,
  - 3 arbitrary entailment.
- It can be shown (see [0]) that w.l.o.g. we can deal only with *concept unsatisfiability*.
- **MUPS:** Let's denote  $MUPS(C, Y)$  a minimal subset  $MUPS(C, Y) \subseteq Y \subseteq X$  causing unsatisfiability of  $C$ .
- **Diagnose:** Let's denote  $DIAG(C, Y)$  a minimal subset  $DIAG(C, Y) \subseteq Y \subseteq X$ , such that if  $DIAG(C, Y)$  is removed from  $Y$ , the concept  $C$  becomes satisfiable.



## Task formulation (2)

- Let's focus on concept  $C$  unsatisfiability. Denote

$$R(C, Y) = \left\{ \begin{array}{ll} \text{true} & \text{iff } Y \not\models (C \sqsubseteq \perp) \\ \text{false} & \text{iff } Y \models (C \sqsubseteq \perp) \end{array} \right\}.$$

- There are many methods (see [0]). We introduce just two of them:
  - Algorithms based on CS-trees.
  - Algorithm for computing a single MUPS[0] + Reiter algorithm [0].



# Algorithms based on CS-trees

1 What if OWL is not enough?

2 Complex Queries

- Evaluation of Conjunctive Queries in  $\mathcal{ALC}$

3 Modeling Error Explanation

- Black-box methods
- **Algorithms based on CS-trees**
- Algorithm based on Reiter's Algorithm
- Algorithm based on Reiter's Algorithm



# CS-trees

- A naive solution: test for each set of axioms from  $\mathcal{T} \cup \mathcal{A}$  for  $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ , whether the set causes unsatisfiability – minimal sets of this form are MUPSeS.
- *Conflict-set trees (CS-trees)* systematize exploration of all these subsets of  $\mathcal{T} \cup \mathcal{A}$ . The main gist :

*If we found a set of axioms  $X$  that do not cause unsatisfiability of  $C$  (i.e.  $X \not\# C \sqsubseteq \perp$ ), then we know (and thus can avoid asking reasoner) that  $Y \not\# C \sqsubseteq \perp$  for each  $Y \subseteq X$ .*

- CS-tree is a representation of the state space, where each state  $s$  has the form  $(D, P)$ , where
  - $D$  is a set of axioms that *necessarily has to be part of all MUPSeS* found while exploring the subtree of  $s$ .
  - $P$  is a set of axioms that *might be part of some MUPSeS* found while exploring the subtree of  $s$ .

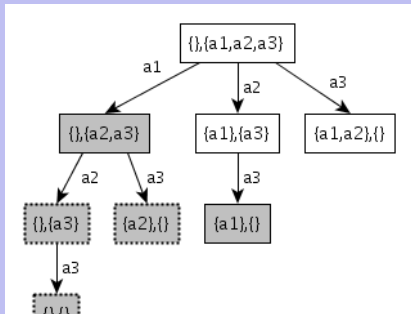


# CS-tree Exploration – Example

## Example

A CS-tree for unsatisfiability of *Person* (abbr.  $Pe$ , not to be mixed with the set  $P$ ) in  $\mathcal{K}_5 = \{\alpha_1, \alpha_2, \alpha_3\}$ :

$$\underbrace{Pe \sqsubseteq \exists hP \cdot (M \sqcap W) \sqcap \forall hP \cdot \neg Pe}_{\alpha_1}, \quad \underbrace{M \sqsubseteq \neg W}_{\alpha_2}, \quad \underbrace{M \sqcup W \sqsubseteq Pe}_{\alpha_3}.$$



In gray states, the concept *Person* is satisfiable ( $R(Pe, D \cup P) = true$ ). States with a dotted border are pruned by the algorithm.

## CS-tree Exploration

The following algorithm is exponential in the number of tableau algorithm runs.

- 1 (Init) The root of the tree is an initial state  $s_0 = (\emptyset, \mathcal{K})$  – a priori, we don't know any axiom being necessarily in a MUPS ( $D_{s_0} = \emptyset$ ), but potentially all axioms can be there ( $P_{s_0} = \mathcal{T} \cup \mathcal{A}$ ). Next, we define  $Z = (s_0)$  and  $R = \emptyset$



## CS-tree Exploration

The following algorithm is exponential in the number of tableau algorithm runs.

- 1 (Init) The root of the tree is an initial state  $s_0 = (\emptyset, \mathcal{K})$  – a priori, we don't know any axiom being necessarily in a MUPS ( $D_{s_0} = \emptyset$ ), but potentially all axioms can be there ( $P_{s_0} = \mathcal{T} \cup \mathcal{A}$ ). Next, we define  $Z = (s_0)$  and  $R = \emptyset$
- 2 (Depth First Search) If  $Z$  is empty, stop the exploration. Otherwise pop the first element  $s$  from  $Z$ .



## CS-tree Exploration

The following algorithm is exponential in the number of tableau algorithm runs.

- 1 (Init) The root of the tree is an initial state  $s_0 = (\emptyset, \mathcal{K})$  – a priori, we don't know any axiom being necessarily in a MUPS ( $D_{s_0} = \emptyset$ ), but potentially all axioms can be there ( $P_{s_0} = \mathcal{T} \cup \mathcal{A}$ ). Next, we define  $Z = (s_0)$  and  $R = \emptyset$
- 2 (Depth First Search) If  $Z$  is empty, stop the exploration. Otherwise pop the first element  $s$  from  $Z$ .
- 3 (Test) If  $R(C, D_s \cup P_s) = \text{true}$  then no subset of  $D_s \cup P_s$  can cause unsatisfiability – we continue with step 2.





## CS-tree Exploration

The following algorithm is exponential in the number of tableau algorithm runs.

- 1 (Init) The root of the tree is an initial state  $s_0 = (\emptyset, \mathcal{K})$  – a priori, we don't know any axiom being necessarily in a MUPS ( $D_{s_0} = \emptyset$ ), but potentially all axioms can be there ( $P_{s_0} = \mathcal{T} \cup \mathcal{A}$ ). Next, we define  $Z = (s_0)$  and  $R = \emptyset$
- 2 (Depth First Search) If  $Z$  is empty, stop the exploration. Otherwise pop the first element  $s$  from  $Z$ .
- 3 (Test) If  $R(C, D_s \cup P_s) = \text{true}$  then no subset of  $D_s \cup P_s$  can cause unsatisfiability – we continue with step 2.
- 4 (Finding an unsatisfiable set) We add  $D_s \cup P_s$  into  $R$  and remove from  $R$  all  $s' \in R$  such that  $D_s \cup P_s \subseteq s'$ . For  $P_s = \alpha_1, \dots, \alpha_N$  we push to  $Z$  a new state  $(D_s \cup \{\alpha_1, \dots, \alpha_{i-1}\}, P_s \setminus \{\alpha_1, \dots, \alpha_i\})$  – we continue with step 2.



## CS-tree Exploration (2)

- Soundness : Step 4 is important – here, we cover all possibilities. It always holds that  $D_s \cup P_s$  differs to  $D_{s'} \cup P_{s'}$  by just one element, where  $s'$  is a successor of  $s$ .



## CS-tree Exploration (2)

- Soundness : Step 4 is important – here, we cover all possibilities. It always holds that  $D_s \cup P_s$  differs to  $D_{s'} \cup P_{s'}$  by just one element, where  $s'$  is a successor of  $s$ .
- Finiteness : Set  $D_s \cup P_s$  is finite at the beginning and gets smaller with the tree depth. Furthermore, in step 4 we generate only finite number of states.



# Algorithm based on Reiter's Algorithm

1 What if OWL is not enough?

2 Complex Queries

- Evaluation of Conjunctive Queries in  $\mathcal{ALC}$

3 Modeling Error Explanation

- Black-box methods
- Algorithms based on CS-trees
- **Algorithm based on Reiter's Algorithm**
- Algorithm based on Reiter's Algorithm



## Another Approach – Reiter's Algorithm

There is an alternative to CS-trees:

- 1 Find a single (arbitrary) MUPS (*singleMUPS* in the next slides).
- 2 “remove the source of unsatisfiability provided by MUPS” (Reiter's algorithm in the next slides) from the set of axioms go explore the remaining axioms in the same manner.



# Algorithm based on Reiter's Algorithm

1 What if OWL is not enough?

2 Complex Queries

- Evaluation of Conjunctive Queries in  $\mathcal{ALC}$

3 Modeling Error Explanation

- Black-box methods
- Algorithms based on CS-trees
- Algorithm based on Reiter's Algorithm
- **Algorithm based on Reiter's Algorithm**



## Finding a single $MUPS(C, Y)$ – example

### Example

The run of  $singleMUPS(Person, \mathcal{K}_5)$  introduced next.



# Finding a single $MUPS(C, Y)$ – example

## Example

The run of  $singleMUPS(Person, \mathcal{K}_5)$  introduced next.

1.PHASE :

$$\begin{aligned}\mathcal{K}_5 &= \{\alpha_1, \alpha_2, \alpha_3\} & R(Person, \{\alpha_1\}) &= true \\ \mathcal{S} &= \{\alpha_1\}\end{aligned}$$





# Finding a single $MUPS(C, Y)$ – example

## Example

The run of  $singleMUPS(Person, \mathcal{K}_5)$  introduced next.

1.PHASE :

$$\begin{aligned}\mathcal{K}_5 &= \{\alpha_1, \alpha_2, \alpha_3\} & R(Person, \{\alpha_1, \alpha_2\}) &= \text{false} \\ \mathcal{S} &= \{\alpha_1, \alpha_2\}\end{aligned}$$



# Finding a single $MUPS(C, Y)$ – example

## Example

The run of  $singleMUPS(Person, \mathcal{K}_5)$  introduced next.

1.PHASE :

$$\mathcal{K}_5 = \{\alpha_1, \alpha_2, \alpha_3\} \quad R(Person, \{\alpha_1, \alpha_2\}) = \text{false}$$
$$S = \{\alpha_1, \alpha_2\}$$

2.PHASE :

$$S = \{\alpha_1, \alpha_2\} \quad R(Person, \{\alpha_1, \alpha_2\} - \{\alpha_1\}) = \text{true}$$
$$K = \{\alpha_1\}$$



# Finding a single $MUPS(C, Y)$ – example

## Example

The run of  $singleMUPS(Person, \mathcal{K}_5)$  introduced next.

1.PHASE :

$$\mathcal{K}_5 = \{\alpha_1, \alpha_2, \alpha_3\} \quad R(Person, \{\alpha_1, \alpha_2\}) = \text{false}$$
$$S = \{\alpha_1, \alpha_2\}$$

2.PHASE :

$$S = \{\alpha_1, \alpha_2\} \quad R(Person, \{\alpha_1, \alpha_2\} - \{\alpha_2\}) = \text{true}$$
$$K = \{\alpha_1, \alpha_2\}$$



## *singleMUPS*( $C, Y$ ) – finding a single MUPS

The following algorithm is polynomial in the number of tableau algorithm applications – the computational complexity stems from the complexity of tableau algorithm itself.

- 1 (Initialization) Denote  $S = \emptyset, K = \emptyset$



## *singleMUPS*( $C, Y$ ) – finding a single MUPS

The following algorithm is polynomial in the number of tableau algorithm applications – the computational complexity stems from the complexity of tableau algorithm itself.

- 1 (Initialization) Denote  $S = \emptyset, K = \emptyset$
- 2 (Finding superset of MUPS) While  $R(C, S) = \text{false}$ , then  $S = S \cup \{\alpha\}$  for some  $\alpha \in Y \setminus S$ .



## *singleMUPS*( $C, Y$ ) – finding a single MUPS

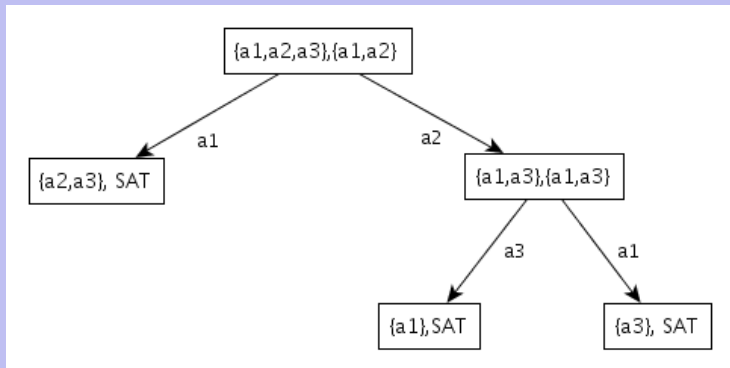
The following algorithm is polynomial in the number of tableau algorithm applications – the computational complexity stems from the complexity of tableau algorithm itself.

- 1 (Initialization) Denote  $S = \emptyset, K = \emptyset$
- 2 (Finding superset of MUPS) While  $R(C, S) = \textit{false}$ , then  $S = S \cup \{\alpha\}$  for some  $\alpha \in Y \setminus S$ .
- 3 (Pruning found set) For each  $\alpha \in S \setminus K$  evaluate  $R(C, S \setminus \{\alpha\})$ . If the result is *false*, then  $K = K \cup \{\alpha\}$ . The resulting  $K$  is itself a MUPS.



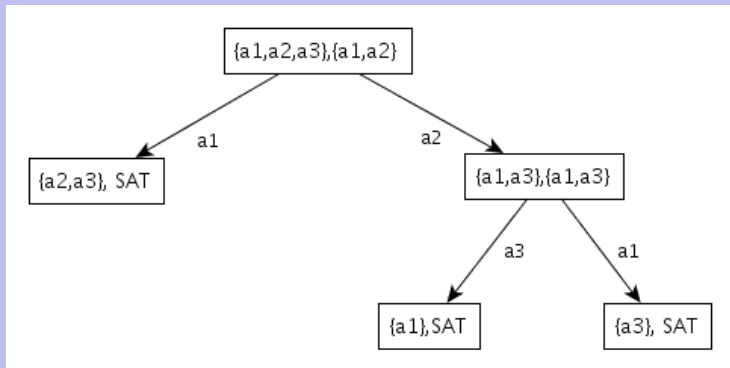
# Finding all MUPSeS – Reiter Algorithm, example

## Example (continued)



# Finding all MUPSeS – Reiter Algorithm, example

## Example (continued)



The algorithm ends up with two MUPSeS  $\{\alpha_1, \alpha_2\}$  a  $\{\alpha_1, \alpha_3\}$ . “For free” we got diagnoses  $\{\alpha_1\}$  a  $\{\alpha_2, \alpha_3\}$ .



## Finding all MUPSes – Reiter Algorithm

- Reiter algorithm runs  $singleMUPS(C, Y)$  multiple times to construct so called “Hitting Set Tree”, nodes of which are pairs  $(\mathcal{K}_i, M_i)$ , where  $\mathcal{K}_i$  lacks some axioms comparing to  $\mathcal{K}$  and  $M_i = singleMUPS(C, \mathcal{K}_i)$ , or  $M_i = \text{“SAT”}$ , if  $C$  is satisfiable w.r.t.  $\mathcal{K}_i$ .



## Finding all MUPSeS – Reiter Algorithm

- Reiter algorithm runs  $singleMUPS(C, Y)$  multiple times to construct so called “Hitting Set Tree”, nodes of which are pairs  $(\mathcal{K}_i, M_i)$ , where  $\mathcal{K}_i$  lacks some axioms comparing to  $\mathcal{K}$  and  $M_i = singleMUPS(C, \mathcal{K}_i)$ , or  $M_i = \text{“SAT”}$ , if  $C$  is satisfiable w.r.t.  $\mathcal{K}_i$ .
- Paths from the root to leaves build up *diagnoses* (i.e. minimal sets of axioms, each of which removed from  $\mathcal{K}$  causes satisfiability of  $C$ ).



## Finding all MUPSeS – Reiter Algorithm

- Reiter algorithm runs  $singleMUPS(C, Y)$  multiple times to construct so called “Hitting Set Tree”, nodes of which are pairs  $(\mathcal{K}_i, M_i)$ , where  $\mathcal{K}_i$  lacks some axioms comparing to  $\mathcal{K}$  and  $M_i = singleMUPS(C, \mathcal{K}_i)$ , or  $M_i = \text{“SAT”}$ , if  $C$  is satisfiable w.r.t.  $\mathcal{K}_i$ .
- Paths from the root to leaves build up *diagnoses* (i.e. minimal sets of axioms, each of which removed from  $\mathcal{K}$  causes satisfiability of  $C$ ).
- Number of  $singleMUPS(C, Y)$  calls is at most exponential w.r.t. the initial axioms count. Why ?



## Finding all MUPSES – Reiter Algorithm (2)

- 1 (Initialization) Find a single MUPS for  $C$  in  $\mathcal{K}$ , and construct the root  $s_0 = (\mathcal{K}, \text{singleMUPS}(C, \mathcal{K}))$  of the hitting set tree. Next, set  $Z = (s_0)$ .



## Finding all MUPSES – Reiter Algorithm (2)

- 1 (Initialization) Find a single MUPS for  $C$  in  $\mathcal{K}$ , and construct the root  $s_0 = (\mathcal{K}, \text{singleMUPS}(C, \mathcal{K}))$  of the hitting set tree. Next, set  $Z = (s_0)$ .
- 2 (Depth First Search) If  $Z$  is empty, STOP.



## Finding all MUPSES – Reiter Algorithm (2)

- 1 (Initialization) Find a single MUPS for  $C$  in  $\mathcal{K}$ , and construct the root  $s_0 = (\mathcal{K}, \text{singleMUPS}(C, \mathcal{K}))$  of the hitting set tree. Next, set  $Z = (s_0)$ .
- 2 (Depth First Search) If  $Z$  is empty, STOP.
- 3 (Test) Otherwise pop an element from  $Z$  and denote it as  $s_i = (\mathcal{K}_i, M_i)$ . If  $M_i = \text{"SAT"}$ , then go to step 2.



## Finding all MUPSES – Reiter Algorithm (2)

- 1 (Initialization) Find a single MUPS for  $C$  in  $\mathcal{K}$ , and construct the root  $s_0 = (\mathcal{K}, \text{singleMUPS}(C, \mathcal{K}))$  of the hitting set tree. Next, set  $Z = (s_0)$ .
- 2 (Depth First Search) If  $Z$  is empty, STOP.
- 3 (Test) Otherwise pop an element from  $Z$  and denote it as  $s_i = (\mathcal{K}_i, M_i)$ . If  $M_i = \text{"SAT"}$ , then go to step 2.
- 4 (Decomposition) For each  $\alpha \in M_i$  insert into  $Z$  a new node  $(\mathcal{K}_i \setminus \{\alpha\}, \text{singleMUPS}(\mathcal{K}_i \setminus \{\alpha\}, C))$ . Go to step 2.



# Modeling Error Explanation – Summary

- finding MUPSeS is the most common way for explaining modeling errors.
- black-box vs. glass box methods. Other methods involve e.g. incremental methods [0].
- the goal is to find MUPSeS (and diagnoses) – what to do in order to solve a modeling problem (unsatisfiability, inconsistency).
- above mentioned methods are quite universal – they can be used for many other problems that are not related with description logics.

