Reasoning algorithms for description logics

G. Falquet

Université de Genève

17 November 2021

Objectives

Given a set of axioms $\mathcal{O}$ (TBox, RBox, Abox) infer implicit knowledge

- subsumption: $\mathcal{O} \models C \sqsubseteq D$
- consistency: for each class C there is a model $\mathcal{I}$ of $\mathcal{O}$ such that $C^{\mathcal{I}}$ is not empty
- instance checking: check if if $\mathcal{O} \models C(a)$

Remarks

1. $C \sqsubseteq D$ if and only if $C \sqcap \neg D$ is not satisfiable.
2. C is subsumed by D iff for any domain $\triangle$ and any extension function I over $\triangle$

$$I(C) \subseteq I(D)$$

A structural algorithm

Works only for $\mathcal{FL}^{-}$

$\mathcal{FL}^{-}$ is limited to $A \mid C \sqcap D \mid \forall R.C \mid \exists R$

2-phases algorithm:

1. Normalization
2. Recursive comparison

Normalization

Flatten all embedded conjunctions :

$$A \sqcap (B \sqcap C) \rightarrow A \sqcap B \sqcap C$$

Factorize all conjunctions of universal quantifiers over the same role

$$\forall R.C \sqcap \forall R.D \rightarrow \forall R.(C \sqcap D)$$

The $\sqsubseteq(C, D)$ algorithm

Let $C = C_1 \sqcap \cdots \sqcap C_n$ and $D = D_1 \sqcap \cdots \sqcap D_m$
$\sqsubseteq (C, D)$ returns **true** iff for every D_j :

1. if D_j is atomic or of the form $\exists R$ then there exists C_i such that $C_i = D_j$;
2. if D_j is of the form $\forall R.D'$ then there exists C_i of the form $\forall R.C'$ such that $\sqsubseteq(C', D')$

Exercise

Use the algorithm to check

- $\text{Adult} \sqcap \text{Male} \sqsubseteq \text{Adult}$
- $\text{Adult} \sqcap \text{Male} \sqcap \text{Rich} \sqsubseteq \text{Rich} \sqcap \text{Adult}$
- $\forall \text{child}.(\text{Adult} \sqcap \text{Male}) \sqsubseteq \forall \text{child}.\text{Adult}$
- $\forall \text{child}.\text{Adult} \sqcap \exists \text{child} \sqsubseteq \forall \text{child}.\text{Adult}$
- $\forall \text{child}.\text{Adult} \not\sqsubseteq \exists \text{child}$
- $\exists \text{child} \not\sqsubseteq \forall \text{child}.\text{Adult}$

Properties of the algorithm

Time complexity $O(|C| \times |D|)$

Soundness The algorithm is sound. Whenever is answers "yes" then C is subsumed by D.

Completeness Whenever $C \sqsubseteq D$ the algorithm answers "yes"

Limits of structural algorithms

- Algorithms based on a syntactic analysis cannot handle more complex logics.
- For instance, $A \sqcup \neg A$ subsumes any concept C even if C does not mention A.

Tableau algorithms

Tableau algorithm prove the non satisfiability of a concept by trying to build a model.
They take advantage of the "tree model property": if there is a model then there is a model that has a tree shape (the object-relation graph is a tree)

TBox and ABox

- N_C : set of concept names
- N_R : role names
- N_I : individual names

ABox : set of assertions of the form

- $C(a)$, C is a concept expression, a an individual
- $r(a, b)$, r is a role name

Model of an ABox

Interpretation I of the roles and concept such that

- I assigns to each individual a an object $I(a) \in \Delta$
- if $C(a)$ is in the ABox then $I(a) \in I(C)$
- if $r(a, b)$ is in the ABox then $(I(a), I(b)) \in I(r)$

Consistency An ABox is consistent if it has a model.

Instance An individual a is an instance of C if in every model I of the ABox A, $I(a) \in I(C)$. Notation $A \models C(a)$

Reformulation $A \models C(a)$ iff $A \cup \{\neg C(a)\}$ is inconsistant

From acyclic TBoxes to ABoxes

If a TBox has no circular definition it is always possible to rewrite every concept definition

$$C \equiv Expr$$

as

$$C \equiv Expr'$$

where $Expr'$ contains only basic (not defined) concept names.
Then if the ABox contains $C(a)$ it can be rewritten as $Expr'(a)$. This is a way to empty the TBox

- This process may produce an exponentially large ABox.

Satisfiability Algorithm

To test the satisfiability of C.
The algorithm tries to build a model I in which $I(C)$ is not empty.

1. put C in negative normal form (all negations beside atomic concept)
2. crate an initial set of ABoxes: $\{\{C(a)\}\}$
3. exhaustively apply the production rules
4. if there is an ABox without *clash* (inconsistency) then C is satisfiable, otherwise it is inconsistent.

Rules for $\sqcap$ and $\sqcup$

For an ABox $\mathcal{A}$ generate one or two new ABoxes $\mathcal{A}'$ and $\mathcal{A}''$

$\rightarrow_\sqcap$ **rule** if $\mathcal{A}$ contains $(C \sqcap D)(x)$ but not $C(x)$ and $D(x)$
then $A' = \mathcal{A}' = \mathcal{A} \cup \{C(x), D(x)\}$.

$\rightarrow_\sqcup$ **rule** if $\mathcal{A}$ contains $(C \sqcup D)(x)$ but neither $C(x)$ nor $D(x)$
then $\mathcal{A}' = \mathcal{A} \cup \{C(x)\}$ and $\mathcal{A}'' = \mathcal{A}A \cup \{D(x)\}$.

Rules for ∃ and ∀

For an ABox $\mathcal{A}$ generate one or two new ABoxes $\mathcal{A}'$ and $\mathcal{A}''$

$\rightarrow_{\exists}$ rule if $\mathcal{A}$ contains $(\exists r.C)(x)$ but no individual name z such that $C(z)$ and $r(x,z)$ are in A
then $\mathcal{A}' = \mathcal{A} \cup \{C(y), r(x,y)\}$.

$\rightarrow_{\forall}$ rule if $\mathcal{A}$ contains $(\forall r.C)(x)$ and $r(x,y)$ but not $C(y)$
then $\mathcal{A}' = \mathcal{A} \cup \{C(y)\}$.

Rules for number restrictions

$\rightarrow_{\geq}$ rule if $\mathcal{A}$ contains $(\geq n\, R)(x)$ but not $R(x, z_i)$ $(1 \leq i \leq n)$ and $\mathrm{diff}(z_i, z_j)$ $(1 \leq i < j \leq n)$ where $z_1, \ldots, z_n$ are individual names
then $\mathcal{A}' = A \cup \{R(x, y_1), \ldots, R(x, y_n)\} \cup \{\mathrm{diff}(y_1, y_2), \mathrm{diff}(y_1, y_3) \ldots, \mathrm{diff}(y_{n-1}, y_n)\}$ where $y_1, \ldots, y_n$ are new individual names.

$\rightarrow_{\leq}$ rule if $\mathcal{A}$ contains $(\leq n\, R)(x)$ and $R(x, y_1), \ldots, R(x, y_{n+1})$, and $\mathrm{diff}(y_i, y_j)$ is not in $\mathcal{A}$ for some $i \neq j$
then for each pair $i > j$ such that $\mathrm{diff}(y_i, y_j)$ is not in $\mathcal{A}$ do $\mathcal{A}' = \mathcal{A} \cup$ the ABox $\mathcal{A}$ where y_i is replaced by y_j.

Example

TBox T

- $C \equiv \exists R.E$,
- $D \equiv A \sqcup \exists R.F$,
- $F \equiv E \sqcup G$

We want to prove that this TBox entails $C \sqsubseteq D$
This amounts to prove that $T \cup \{C \sqcap \neg D\}$ is inconsistent.

- $C \sqcap \neg D$ is inconsistent if we cannot find a model for $(C \sqcap \neg D)(a)$
- Expanding $(C \sqcap \neg D)(a)$ with the axioms yields
  - $((\exists R.E) \sqcap \neg(A \sqcup \exists R.F))(a)$
  - $\equiv ((\exists R.E) \sqcap \neg(A \sqcup \exists R.(E \sqcup G)))(a)$
- In negative normal form:
  - $\equiv (\exists R.E) \sqcap (\neg A \sqcap \neg\exists R.(E \sqcup G)))(a)$
  - $\equiv (\exists R.E) \sqcap (\neg A \sqcap \forall R.(\neg E \sqcap \neg G)))(a)$

Rule applications

ABox expansion

$A_0 = \{(\exists R.E) \sqcap (\neg A \sqcap \forall R.(\neg E \sqcap \neg G)))(a)\}$
$A_1 = A_0 \cup \{(\exists R.E)(a), \neg A(a), (\forall R.(\neg E \sqcap \neg G))(a)\}$ ($\sqcap$ rule)
$A_2 = A_1 \cup \{R(a, b), E(b)\}$ ($\exists$ rule)
$A_3 = A_2 \cup \{(\neg E \sqcap \neg G)(b), \neg E(b), \neg G(b)\}$ ($\forall$ rule and $\sqcap$ rule)

There is a clash in A_3, it contains $E(b)$ and $\neg E(b)$
There is no other ABox, hence $C \sqcap \neg D$ is inconsistent

Properties of the algorithm

1. rule application always terminates (no infinite loop).
2. C is consistent iff the algorithm produced at least one clash-free ABox $\mathcal{A}$.

An ABox $\mathcal{A}$ has a clash if one of these conditions is true

- $\{\perp(x)\} \subseteq \mathcal{A}$ for some individual name x
- $\{B(x), \neg B(x)\} \subseteq \mathcal{A}$ for some individual name x and some concept name B
- $\{(\leq n\, R)(x)\} \cup \{R(x, y_1), \ldots, R(x, y_{n+1})\} \cup \{\text{diff}(y_i, y_j) | 1 \leq i < j \leq n+1\} \subseteq A$ for individual names $x, y_1, \ldots, y_{n+1}$, $n > 0$, and R a role name.

Complexity (AND Branching)

The size of the ABox set generated during the process may be exponential in the size of C.
e.g. for the following family of ABoxes

$$C_1 := \exists r.A \sqcap \exists r.B,$$

$$C_2 := \exists r.A \sqcap \exists r.B \sqcap \forall r(\exists r.A \sqcap \exists r.B),$$

$$\dots$$

$$C_{n+1} := \exists r.A \sqcap \exists r.B \sqcap \forall r.C_n$$

ABox for C_1

$$(\exists r.A \sqcap \exists r.B)(a_1)$$

complete ABox:

$$\{\ldots, r(a_1, a'), r(a_1, b'), A(a'), B(b')\}$$

ABox for C_2

$$(\exists r.A \sqcap \exists r.B \sqcap \forall r(\exists r.A \sqcap \exists r.B))(a_2)$$

complete ABox:

$$\begin{aligned}\{\ldots, \quad & r(a_2, a_1), r(a_2, b_1), A(a_1), B(b_1), \\ & r(a_1, a'), r(a_1, b'), A(a'), B(b'), \\ & r(b_1, a''), r(b_1, b''), A(a''), B(b'') \quad \}\end{aligned}$$

Exponential growth (doubles at each level)

Complexity (OR Branching)

Checking the satisfiability of

$$(\exists R.A) \sqcap (\exists R.(\neg A \sqcap \neg B)) \sqcap (\exists R.B) \sqcap \leq 2R$$

To satisfy the $\exists$ we must generate

- $R(a, x_1), A(x_1)$
- $R(a, x_2), (\neg A \sqcap \neg B)(x_2)$
- $R(a, x_3), B(x_3)$

To satisfy $\leq 2R$ we must generate (and explore) 3 cases

- $x_1 = x_2$
- or $x_2 = x_3$
- or $x_1 = x_3$

For general TBoxes

Remark. A TBox

$$\{C_1 \sqsubseteq D_1, \ldots, C_n \sqsubseteq D_n\}$$

is equivalent to the TBox

$$\{\top \sqsubseteq ((\neg C_1 \sqcup D_1) \sqcap \cdots \sqcap (\neg C_n \sqcup D_n))\}$$

Thus we can consider a TBox with a single axiom of the form

$$\top \sqsubseteq C$$

.i.e. every object of the domain must belong to the interpretation of C

Additional rule

To represent the TBox axiom $\top \sqsubseteq C$ we add a new rule

$\rightarrow_{\top \sqsubseteq C}$-rule if the individual name x appears in the ABox and $C(x)$ is not present, add $C(x)$ to the ABox

Blocking

If the TBox is cyclic, the $\rightarrow_\exists$-rule may create infinite sequences of individuals connected through roles, although a finite model may exist.

Blocked rule

The application of the $\rightarrow_\exists$-rule to an individual x is blocked by an individual y if

- x is younger than y, i.e. x has been introduced by an $\rightarrow_\exists$-rule after the introduction of y
- x has no more constraints than y, i.e. $\{C : C(x) \in \text{ABox}\} \subseteq \{C : C(y) \in \text{ABox}\}$

The idea is that we can use y instead of x to create a model.

OWL 2 RL and rule-based reasoning

- For RDFS there is a set of IF ... THEN ... rules that can generate all the consequences of a set of axioms
  - IF (x p y) and (p rdfs:range c) THEN (y rdf:type c)
- It is not the case with OWL 2
- But it is possible on some sublanguages of OWL

OWL 2 RL[1]

An OW 2 profile with syntactic restrictions

- Aimed at efficient reasoning with rule-based systems

With a set of inference rules for reasoning

- complete reasoning for the OWL 2 RL profile (see Theorem PR1 in [1])
- incomplete reasoning for OWL 2

1 [1] https://www.w3.org/TR/owl2-profiles/#OWL_2_RL

OWL 2 RL definition by syntactic restrictions

In an axiom *Left* $\sqsubseteq$ *Right*

Left may be
a class name (except owl:Thing),
E **and** F,
E **or** F,
R **some** C,
R **hasValue** v
oneOf $(\ldots)$,

Right may be
a class name (except owl:Thing),
E **and** F,
not C,
R **only** C,
R **hasValue** v,
max $0/1$ C

Inference Rules for Individuals

Basic rule: if the ontology contains

$$X \sqsubseteq Y$$

$$X(a)$$

it entails

$$Y(a)$$

The "shape" or $X$and Y determines inference rules

Left rules

and

E **and** $F \sqsubseteq Y$
$\Rightarrow (E \textbf{ and } F)(x) \rightarrow Y(x)$
$\Rightarrow E(x) \wedge F(x) \rightarrow Y(x)$
.

or

E **or** $F \sqsubseteq Y$
$\Rightarrow (E \textbf{ or } F)(x) \rightarrow Y(x)$
$\Rightarrow E(x) \vee F(x) \rightarrow Y(x)$
$\Rightarrow E(x) \rightarrow Y(x), F(x) \rightarrow Y(x)$

Rules for OWL 2 RL in RDF

Intersection

```
?c owl:intersectionOf (?c1, ..., ?cn)
?y, rdf:type, ?c1
?y, rdf:type, ?c2
...
?y, rdf:type, ?cn
--->
?y rdf:type ?c
```

Union

```
?C owl:unionOf ?x .
?x rdf:rest*/rdf:first ?Ci .
?y rdf:type ?Ci .
--->
?y rdf:type ?C .
```

some

R **some** $C \sqsubseteq Y$

$\Rightarrow (R \text{ some } C)(x) \rightarrow Y(x)$

$\Rightarrow \exists y : C(y) \wedge R(x,y) \rightarrow Y(x)$

$\Leftrightarrow C(y) \wedge R(x,y) \rightarrow Y(x)$

.

R hasValue v

R **has value** $v \sqsubseteq Y$

$\Rightarrow R(x,v) \rightarrow Y(x)$

… for OWL 2 RL in RDF

some

```
?X owl:someValuesFrom ?Y .
?X owl:onProperty, ?p  .
?u ?p ?v .
?v rdf:type ?Y .
--->
?u rdf:type ?X .
```

hasValue

```
?x owl:hasValue ?v.
?x owl:onProperty ?p.
?u ?p ?v.
--->
?u rdf:type ?x.
```

Right rules

not

$X \sqsubseteq$ **not** Y
$\Rightarrow X(x) \rightarrow (\textbf{not}\, Y)(x)$
$\Rightarrow \neg X(x) \vee \neg Y(x)$
$\Rightarrow \neg(X(x) \wedge Y(x))$
$\Rightarrow (X(x) \wedge Y(x)) \rightarrow$ **False**

only

$X \sqsubseteq R\,$**only** C
$\Rightarrow X(x) \rightarrow (R\,\textbf{only}\; C)(x)$
$\Rightarrow X(x) \rightarrow (R(x,y) \rightarrow C(y))$
$\Leftrightarrow (X(x) \wedge (R(x,y)) \rightarrow C(y))$

RDF Rule for All

```
?X owl:allValuesFrom ?Y .
?X owl:onProperty, ?p   .
?u, ?p, ?v .
?u, rdf:type, ?X .
--->
?v, rdf:type, ?Y .
```

Additional rules

- Equality rules (on owl:sameAs)
- Rules for property axioms
- Rules for owl:equivalentClass, disjoint, alldisjoint
- Schema (TBox) inference axioms

Example: Functional property rule

IF $\text{functional}(p)$ AND $p(x, y)$ AND $p(x, y)$ THEN $y = z$

```
?p rdf:type owl:FunctionalProperty .
?x ?p ?y .
?x ?p ?z .
--->
?y  <owl:sameAs>  ?z
```

In Practice

Complete OWL 2 reasoners (Hermit, Pellet, ...)

- usable on TBoxes, e.g. to infer the class hierarchy
- impractical on ABoxes (data)

OWL 2 RL reasoners

- efficient enough to reason on ABoxes (not complete for TBoxes)
- Implemented in triple stores (GraphDB, ...) with rules engines (with RETE algorithms)
  - in GraphDB on can load their own ruleset