A Description Logics Tableau Reasoner in Prolog

Riccardo Zese, Elena Bellodi, Evelina Lamma, Fabrizio Riguzzi

University of Ferrara, Italy

riccardo.zese@unife.it, elena.bellodi@unife.it, evelina.lamma@unife.it, fabrizio.riguzzi@unife.it
Outline

1. Introduction
2. OWL DL
3. Tableau Algorithm
4. Why Prolog?
5. Trill - Tableau Reasoner for Description Logics in Prolog
6. Experiments
7. Conclusions
Introduction

Semantic Web

- Aims at making information available in a form that is understandable by machines.
- Development of the Web Ontology Language (OWL)
  - Family of knowledge representation formalisms for defining knowledge bases.
  - Based on Description Logics
  - The OWL DL sublanguage is based on $SHOIN(D)$.

Reasoners

- Most DL reasoners use a tableau algorithm for doing inference.
- Most of them are implemented in a procedural language.
  - Example: Pellet, RacerPro, FaCT++
Non-determinism

- **Problem**: some tableau expansion rules are non-deterministic
  - Reasoners implement a search strategy in an or-branching space.
- We want to find all the possible explanations for a query
  - The algorithm has to explore all the non-deterministic choices done
The $\mathcal{SHOIN}(D)$ Description Logic

- Let $\mathbf{A}$, $\mathbf{R}$ and $\mathbf{I}$ be sets of atomic concepts, roles and individuals, respectively.
  - A RBox $\mathcal{R}$ consists of a finite set of transitivity axioms ($\text{Trans}(R)$), where $R \in \mathbf{R}$, and role inclusion axioms ($R \sqsubseteq S$), where $R, S \in \mathbf{R} \cup \mathbf{R}^{-}$.
  - A TBox $\mathcal{T}$ is a finite set of concept inclusion axioms $C \sqsubseteq D$, where $C$ and $D$ are concepts. We use $C \equiv D$ to abbreviate $C \sqsubseteq D$ and $D \sqsubseteq C$.
  - A ABox $\mathcal{A}$ is a finite set of concept membership axioms $a : C$, role membership axioms $(a, b) : R$, equality axioms $a \equiv b$, and inequality axioms $a \neq b$, where $C$ is a concept, $R \in \mathbf{R}$ and $a, b \in \mathbf{I}$.
- A knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{R}, \mathcal{A})$ consists of a TBox $\mathcal{T}$, an RBox $\mathcal{R}$ and an ABox $\mathcal{A}$. 
**SHOIN(D) Semantics**

- A knowledge base $\mathcal{K}$ can be given a semantics by transforming it into a predicate logic theory and then using the model-theoretic semantics of the resulting theory.

- Translation of $\text{SHOIN}$ into First Order Logic with Counting Quantifiers: $\pi_x$ and $\pi_y$ map concept expressions to logical formulas

\[
\begin{align*}
\pi_x(A) &= A(x) \\
\pi_x(\neg C) &= \neg \pi_x(C) \\
\pi_x(\{a\}) &= x = a \\
\pi_x(C \cap D) &= \pi_x(C) \land \pi_x(D) \\
\pi_x(C \cup D) &= \pi_x(C) \lor \pi_x(D) \\
\pi_x(\exists R.C) &= \exists y. R(x, y) \land \pi_y(C) \\
\pi_x(\exists R^-.C) &= \exists y. R(y, x) \land \pi_y(C) \\
\pi_x(\forall R.C) &= \forall y. R(x, y) \rightarrow \pi_y(C) \\
\pi_x(\forall R^-.C) &= \forall y. R(y, x) \rightarrow \pi_y(C) \\
\pi_x(\geq nR) &= \exists^{\geq n} y. R(x, y) \\
\pi_x(\geq nR^-) &= \exists^{\geq n} y. R(y, x) \\
\pi_x(\leq nR) &= \exists^{\leq n} y. R(x, y) \\
\pi_x(\leq nR^-) &= \exists^{\leq n} y. R(y, x)
\end{align*}
\]

and $\pi_y$ is obtained from $\pi_x$ by replacing $x$ with $y$ and vice-versa.
### SHOIN(\(D\)) Semantics

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(C \sqsubseteq D)</td>
<td>(\forall x. \pi_x(C) \rightarrow \pi_x(D))</td>
</tr>
<tr>
<td>(R \sqsubseteq S)</td>
<td>(\forall x, y. R(x, y) \rightarrow S(x, y))</td>
</tr>
<tr>
<td>Trans((R))</td>
<td>(\forall x, y, z. R(x, z) \land R(z, y) \rightarrow R(x, y))</td>
</tr>
<tr>
<td>(a : C)</td>
<td>(\pi_a(C))</td>
</tr>
<tr>
<td>((a, b) : R)</td>
<td>(R(a, b))</td>
</tr>
<tr>
<td>(a = b)</td>
<td>(a = b)</td>
</tr>
<tr>
<td>(a \neq b)</td>
<td>(a \neq b)</td>
</tr>
</tbody>
</table>
Example - people+pets ontology

\[
\begin{align*}
\text{fluffy} & : \text{Cat} \\
\text{tom} & : \text{Cat} \\
\text{Cat} & \sqsubseteq \text{Pet} \\
\exists \text{hasAnimal}. \text{Pet} & \sqsubseteq \text{NatureLover} \\
(\text{kevin}, \text{fluffy}) & : \text{hasAnimal} \\
(\text{kevin}, \text{tom}) & : \text{hasAnimal}
\end{align*}
\]

- \text{fluffy} and \text{tom} are \text{Cat}; \text{Cats} are \text{Pets}; everyone who has a pet animal (\exists \text{hasAnimal}. \text{Pet}) is a \text{NatureLover}; \text{kevin} has two animals, \text{fluffy} and \text{tom}. 

\[\text{Zese, Bellodi, Lamma, Riguzzi (ENDIF)}\]
Example - The translation

\[
\begin{align*}
\text{fluffy} : \text{Cat} & \quad \text{Cat}(\text{fluffy}) \\
\text{tom} : \text{Cat} & \quad \text{Cat}(\text{tom}) \\
\text{Cat} \sqsubseteq \text{Pet} & \quad \forall x. \text{Cat}(x) \rightarrow \text{Pet}(x) \\
\exists \text{hasAnimal}. \text{Pet} \sqsubseteq \text{NatureLover} & \quad \forall x. \exists y. \text{hasAnimal}(x, y) \land \text{Pet}(y) \rightarrow \text{NatureLover}(x) \\
(\text{kevin}, \text{fluffy}) : \text{hasAnimal} & \quad \text{hasAnimal}(\text{kevin}, \text{fluffy}) \\
(\text{kevin}, \text{tom}) : \text{hasAnimal} & \quad \text{hasAnimal}(\text{kevin}, \text{tom})
\end{align*}
\]
The tableau algorithm

- A tableau is an ABox represented as a graph in which:
  - Each node represents an individual $a$ and is labeled with the set of concepts it belongs to;
  - Each edge between two individuals $a$ and $b$ is labeled with the set of roles to which the couple $(a, b)$ belongs.

- A tableau algorithm proves an axiom by refutation
  - Axiom $E$ is entailed if $\neg E$ has no model in the KB.
  - Example 1: to test a class assertion axiom $C(a)$, it adds $\neg C$ to the label of $a$.
  - Example 2: to test the inconsistency of a concept $C$, it adds a new anonymous node $a$ to the tableau and adds $\neg C$ to the label of $a$. 
A tableau algorithm repeatedly applies a set of consistency preserving tableau expansion rules until a clash is detected or a clash-free graph is found to which no more rules are applicable.

A clash (contradiction) is either:
- a couple \((C, a)\) where \(C\) and \(\neg C\) are present in the label of a node;
- a couple \((a = b, a \neq b)\), where \(a\) and \(b\) are individuals.

If the expansion of the tableau with the query leads to at least one clash the query is entailed w.r.t. the KB.
The tableau algorithm

For ensuring the termination, the tableau algorithm uses a blocking system.

- In a tableau a node $x$ can be a nominal node (if its label contains a nominal) or a blockable node.
- In a tableau a node $x$ is blocked if it has ancestors $x_0$, $y$, $y_0$ such that:
  1. $x$ and $y$ are successor of $x_0$ and $y_0$ respectively;
  2. $x$, $y$ and all nodes on path from $y$ to $x$ are blockable;
  3. $L(x) = L(y)$ and $L(x_0) = L(y_0)$;
  4. $L(\langle x_0, x \rangle) = L(\langle y_0, y \rangle)$. 

The tableau algorithm

- Each expansion rule updates also a tracing function $\tau$.
  - It associates sets of axioms with labels of nodes and edges.
  - Is initialized as the empty set for all the elements of its domain except for the information contained in the ABox.

\[
\rightarrow \text{unfold: } \quad \text{if } A \in \mathcal{L}(a), \text{ } A \text{ atomic and } (A \sqsubseteq D) \in K, \text{ then}
\]
\[
\quad \text{if } D \notin \mathcal{L}(a), \text{ then}
\]
\[
\quad \quad \mathcal{L}(a) := \mathcal{L}(a) \cup \{D\}
\]
\[
\quad \quad \tau(D, a) := \tau(A, a) \cup \{(A \sqsubseteq D, a)\}
\]

\[
\rightarrow \sqcup: \quad \text{if } (C_1 \sqcup C_2) \in \mathcal{L}(a) \text{ and } a \text{ is not indirectly blocked, then}
\]
\[
\quad \text{if } \{C_1, C_2\} \cap \mathcal{L}(a) = \emptyset, \text{ then}
\]
\[
\quad \quad \text{Generate graphs } G_i := G \text{ for each } i \in \{1, 2\},
\]
\[
\quad \quad \mathcal{L}(a) := \mathcal{L}(a) \cup \{C_i\} \text{ for each } i \in \{1, 2\}
\]
\[
\quad \quad \tau(C_i, a) := \tau((C_1 \sqcup C_2), a)
\]

- The output of the algorithm is a set of axioms that is a fragment of the KB from which the query is entailed.
The reasoners implemented using procedural languages have to implement also a backtracking algorithm to find all the possible explanations.

Example: Pellet uses an hitting set algorithm that repeatedly removes an axiom from the KB and then computes again a new explanation.

Reasoners written in Prolog can exploit Prolog’s backtracking facilities for performing the search.
Related work

The possibility of exploiting Prolog’s backtracking facilities has been observed in many works:

- [Beckert and Posegga, 1995] proposed a tableau reasoner in Prolog for FOL that is not tailored to DLs.
- [Wielemaker et al., 2012] proposed an RDF and Semantic Web library for SWI Prolog. This library is more focused on storing and querying RDF triples, while it has limited support for OWL reasoning.
- [Ricca et al., 2009] presented OntoDLV, a system that implements a logic-based ontology representation language, called OntoDLP, by an extension of the (disjunctive) ASP named DVL.
Related work

- [Meissner, 2004] presented the implementation of a Prolog reasoner for the DL $\mathcal{ALCN}$.
- [Herchenröder, 2006] considers $\mathcal{ALC}$ and improves [Meissner, 2004] by implementing heuristic search techniques to reduce the running time.
- [Faizi, 2011] added to [Herchenröder, 2006] the possibility of returning explanations for queries w.r.t. $\mathcal{ALC}$ KBs.
- [Hustadt et al., 2008] proposed KAON2 that reduces a $\mathcal{SHIQ}$ KB into a disjunctive datalog program and performs reasoning on it.
- [Lukácsy and Szeredi, 2009] proposed DLog that is an ABox reasoning algorithm for the $\mathcal{SHIQ}$ language that allows to store the content of the ABox externally in a database.
TRILL solves the instantiated axiom pinpointing problem in which we are interested in instantiated explanations that entail a query.

An instantiated explanation is a finite set \( F = \{(F_1, \theta_1), \ldots, (F_n, \theta_n)\} \) where \( F_1, \ldots, F_n \) are axioms and \( \theta_1, \ldots, \theta_n \) are instantiations.

Instantiated axiom pinpointing is useful for

- a more fine-grained debugging of the ontology, by highlighting the individuals to which axioms are applied, it may point to parts of the ABox to be modified for repairing the KB
- probabilistic reasoning
1. Thea2 library for converting OWL DL ontologies to Prolog:
   - each OWL axiom is translated into a Prolog fact.
2. It applies all the possible expansion rules, first the non-deterministic ones then the deterministic ones.
3. It returns the set of the instantiated explanations.
The tableau is represented by a couple \((A, T)\) where

- \(A\) is a list containing all the class and role assertions with the corresponding value of the tracing function and information about nominals;
- \(T\) contains the structure of the tableau: it is a triple \((G, RBN, RBR)\) in which:
  
  1. \(G\) is a directed graph that contains the structure of the tableau,
  2. \(RBN\) is a red-black tree that traces the correspondence between a couple of individuals and the set of the labels of the edge between the two individuals,
  3. \(RBR\) is a red-black tree that traces the correspondence between a role and the set of couples of individuals that are linked by the role

This representation allows us to efficiently manage the blocking system.
Deterministic Rules - rule_name(Tab, Tab1)

→ unfold: if $A \in \mathcal{L}(a)$, $A$ atomic and $(A \subseteq D) \in K$, then
  if $D \notin \mathcal{L}(a)$, then
  \[ \mathcal{L}(a) := \mathcal{L}(a) \cup \{D\} \]
  \[ \tau(D, a) := \tau(A, a) \cup \{(A \subseteq D, a)\} \]

unfold_rule((A,T), ([(classAssertion(D,Ind), [(Ax,Ind)|Expl])|A],T)):-
  find((classAssertion(C,Ind),Expl),A),
  atomic(C),
  find_sub_sup_class(C,D,Ax),
  absent(classAssertion(D,Ind),[(Ax,Ind)|Expl],(A,T)).

find_sub_sup_class(C,D,subClassOf(C,D)):-
  subClassOf(C,D).

find_sub_sup_class(C,D,equivalentClasses(L)):-
  equivalentClasses(L),
  member(C,L),
  member(D,L),
  C\==D.
Non-Deterministic Rules - *rule_name*(Tab, TabList)

\[
\rightarrow \sqcup: \quad \text{if } (C_1 \sqcup C_2) \in \mathcal{L}(a) \text{ and } a \text{ is not indirectly blocked, then}
\]
\[
\quad \text{if } \{C_1, C_2\} \cap \mathcal{L}(a) = \emptyset, \text{ then}
\]
\[
\quad \text{Generate graphs } G_i := G \text{ for each } i \in \{1, 2\},
\]
\[
\quad \mathcal{L}(a) := \mathcal{L}(a) \cup \{C_i\} \text{ for each } i \in \{1, 2\}
\]
\[
\quad \tau(C_i, a) := \tau((C_1 \sqcup C_2), a)
\]

or_rule((A,T),L):-
  find((classAssertion(unionOf(LC),Ind),Expl),A),
  \(+\ indirectly_blocked(Ind,T0),
  findall((A1,T),scan_or_list(LC,Ind,Expl,A,T,A1),L),
  L\=[],!.

scan_or_list([],_Ind,_Expl,A,T,A).

scan_or_list([C|_T],Ind,Expl,A,T,[classAssertion(C,Ind),Expl]|A):-
  absent(classAssertion(C,Ind),Expl,(A,T)).

scan_or_list([_C|T],Ind,Expl,A0,T,A):-
  scan_or_list(T,Ind,Expl,A0,T,A).
Experiments

- Comparison with BUNDLE\(^1\) that also solves the instantiated axiom pinpointing problem.
- We consider four datasets:
  1. BRCA that models the risk factor of breast cancer;
  2. An extract of DBPedia;
  3. Biopax level 3 that models metabolic pathways;
  4. Vicodi that contains information on European history.
- The DBPedia and the Biopax KBs without ABox
- The BRCA with an ABox containing 1 individual
- The Vicodi with an ABox containing 19 individuals

---

Experiments

- Two different subclass-of queries w.r.t. the DBPedia and the Biopax KBs.
- Two different instance-of queries w.r.t. the BRCA and the Vicodi KBs.
- We ran each query 50 times for a total of 100 executions of the reasoner for each KB.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>n. axioms</th>
<th>av. n. expl</th>
<th>TRILL time (ms)</th>
<th>BUNDLE time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRCA</td>
<td>322</td>
<td>6.5</td>
<td>40,691</td>
<td>10,210</td>
</tr>
<tr>
<td>DBPedia</td>
<td>535</td>
<td>16.0</td>
<td>76,804</td>
<td>28,040</td>
</tr>
<tr>
<td>Biopax level 3</td>
<td>826</td>
<td>2.0</td>
<td>4</td>
<td>1,451</td>
</tr>
<tr>
<td>Vicodi</td>
<td>220</td>
<td>1.0</td>
<td>24</td>
<td>1,004</td>
</tr>
</tbody>
</table>
Conclusions and future works

The results we obtained show that:

1. Prolog is a viable language for implementing DL reasoning algorithms.
2. TRILL’s performances are comparable with those of a reasoner that solves the same problem.

Future works

- Apply optimizations to TRILL to better manage the expansion of the tableau
- Exploit TRILL to perform reasoning on probabilistic ontologies.
Thanks.

Questions?
References I

leantap: Lean tableau-based deduction.

A Description Logic Prover in Prolog, Bachelor’s thesis, Informatics Mathematical Modelling, Technical University of Denmark.

Lightweight semantic web oriented reasoning in Prolog: Tableaux inference for description logics.
Master’s thesis, School of Informatics, University of Edinburgh.

Deciding expressive description logics in the framework of resolution.

Efficient description logic reasoning in prolog: The dlog system.

An automated deduction system for description logic with alcn language.
*Studia z Automatyki i Informatyki*, 28-29:91–110.

Ontodlv: An asp-based system for enterprise ontologies.
References II