Noname manuscript No.
(will be inserted by the editor)

Tableau Reasoning for Description Logics and its Extension to Probabilities

Riccardo Zese · Elena Bellodi · Fabrizio Riguzzi · Giuseppe Cota · Evelina Lamma

Received: date / Accepted: date

Abstract The increasing popularity of the Semantic Web drove to a widespread adoption of Description Logics (DLs) for modeling real world domains. To help the diffusion of DLs, a large number of reasoning algorithms have been developed. Usually these algorithms are implemented in procedural languages such as Java or C++. Most of the reasoners exploit the tableau algorithm which features non-determinism, that is not easily handled by those languages. Prolog directly manages non-determinism, thus is a good candidate for dealing with the tableau's non-deterministic expansion rules.

We present TRILL, for "Tableau Reasoner for descrIption Logics in pro-Log", that implements a tableau algorithm and is able to return explanations for queries and their corresponding probability, and TRILL^P, for "TRILL powered by Pinpointing formulas", which is able to compute a Boolean formula representing the set of explanations for a query. Reasoning on real world domains also requires the capability of managing probabilistic and uncertain information. We show how TRILL and TRILL^P can be used to compute the probability of queries to knowledge bases following DISPONTE semantics. Experiments comparing these with other systems show the feasibility of the approach.

Keywords Description Logics, Tableau, Prolog, Semantic Web

Riccardo Zese, Elena Bellodi, Giuseppe Cota, Evelina Lamma
Dipartimento di Ingegneria
University of Ferrara
Via Saragat 1, I-44122, Ferrara, Italy
E-mail: {riccardo.zese,elena.bellodi,giuseppe.cota,evelina.lamma}@unife.it

Fabrizio Riguzzi
Dipartimento di Matematica e Informatica
University of Ferrara
Via Saragat 1, I-44122, Ferrara, Italy
E-mail: fabrizio.riguzzi@unife.it

1 Introduction

The Semantic Web aims at making information regarding real world domains available in a form that is understandable by machines [31]. The World Wide Web Consortium is working for realizing this vision by supporting the development of the Web Ontology Language (OWL), a family of knowledge representation formalisms for defining ontologies. OWL is based on Description Logics (DLs), a set of languages that are restrictions of first order logic (FOL) with decidability and, in some cases, low complexity. For example, the OWL DL sublanguage is based on the expressive $\mathcal{SHOIN}(\mathbf{D})$ DL while OWL 2 corresponds to the $\mathcal{SROIQ}(\mathbf{D})$ DL [31]. Moreover, uncertain information is intrinsic to real world domains, thus the combination of probability and logic theories is of foremost importance.

In order to fully support the development of the Semantic Web, efficient DL reasoners, such as Pellet, RacerPro, FaCT++ and HermiT were implemented, that offer a variety of services (classification, satisfiability, query answering, entailment, consistency, etc.) for ontologies. One of the most common approaches for reasoning is the tableau algorithm that features some non-deterministic expansion rules. This requires the developers to implement a search strategy in an or-branching search space. Moreover, if we want to compute the probability of a query, the algorithm has to compute all the explanations for the query, thus it has to explore all the non-deterministic choices taken during the execution. Despite the large number of available reasoners, only few of them are able to manage probabilistic information as well.

In this paper, after introducing DLs (Section 2), we present the tableau algorithm and discuss its implementation via logic programming (Section 3). In particular, we present the systems TRILL for "Tableau Reasoner for deScrIption Logics in proLog" and TRILL^P for "TRILL powered by Pinpointing formulas". They are tableau reasoners for the $\mathcal{SHOIQ}$ DL and for the $\mathcal{ALC}$ DL respectively, both implemented in Prolog. Prolog's search strategy is exploited for taking into account the non-determinism of the tableau rules. TRILL and TRILL^P use the Thea2 library [64] for parsing OWL in its various dialects. Thea2 translates OWL files into a Prolog representation in which each axiom is mapped to a fact. TRILL and TRILL^P can check the consistency of a concept and the entailment of an axiom from an ontology. We then present various experiments on the feasibility of the use of Prolog for implementing reasoning algorithms.

Then the paper discusses probabilistic inference and the extensions of TRILL and TRILL^P for managing uncertain information (Section 4). We first describe the probabilistic semantics DISPONTE [7,58] which enables the definition of probabilistic ontologies.

Both TRILL and TRILL^P are extended to compute the probability of the queries following DISPONTE. We provide experimentations to check the competitiveness of TRILL and TRILL^P with respect to state-of-the-art probabilistic inference systems such as PRONTO [37] and BORN [13]. Finally, we conclude the paper and highlight directions for future work.

2 Description Logics

DLs are knowledge representation formalisms that are at the basis of the Semantic Web [1,2] and are used for modeling ontologies. They possess nice computational properties such as decidability and/or low complexity.

Usually, DLs' syntax is based on concepts and roles which correspond respectively to sets of individuals and sets of pairs of individuals of the domain. We first briefly describe $\mathcal{ALC}$ and then $\mathcal{SHOIQ}$.

Let $\mathbf{C}$, $\mathbf{R}$ and $\mathbf{I}$ be sets of *atomic concepts*, *atomic roles* and *individuals*, respectively. *Concepts* are defined by induction as follows. Each $C \in \mathbf{C}$ is a concept, $\bot$ and $\top$ are concepts. If C, C_1 and C_2 are concepts and $R \in \mathbf{R}$, then $(C_1 \sqcap C_2)$, $(C_1 \sqcup C_2)$ and $\neg C$ are concepts, as well as $\exists R.C$ and $\forall R.C$. 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 the conjunction of $C \sqsubseteq D$ and $D \sqsubseteq C$. An *ABox* $\mathcal{A}$ is a finite set of *concept membership axioms* $a : C$, *role membership axioms* $(a, b) : R$, *equality axioms* $a = b$ and *inequality axioms* $a \neq b$, where $C \in \mathbf{C}$, $R \in \mathbf{R}$ and $a, b \in \mathbf{I}$. A *knowledge base* (KB) $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ consists of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$ and is usually assigned a semantics in terms of interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a non-empty *domain* and $\cdot^{\mathcal{I}}$ is the *interpretation function* that assigns an element in $\Delta^{\mathcal{I}}$ to each $a \in \mathbf{I}$, a subset of $\Delta^{\mathcal{I}}$ to each $C \in \mathbf{C}$ and a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ to each $R \in \mathbf{R}$.

The mapping $\cdot^{\mathcal{I}}$ is extended to all concepts (where $R^{\mathcal{I}}(x) = \{y | (x, y) \in R^{\mathcal{I}}\}$) as:

$$
\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\
\bot^{\mathcal{I}} &= \emptyset \\
(C_1 \sqcap C_2)^{\mathcal{I}} &= C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} \\
(C_1 \sqcup C_2)^{\mathcal{I}} &= C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}} \\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(\forall R.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} | R^{\mathcal{I}}(x) \subseteq C^{\mathcal{I}}\} \\
(\exists R.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} | R^{\mathcal{I}}(x) \cap C^{\mathcal{I}} \neq \emptyset\}
\end{aligned}
$$

In the following we describe $\mathcal{SHOIQ}$ showing what it adds to $\mathcal{ALC}$. A *role* is either an atomic role $R \in \mathbf{R}$ or the inverse R^- of an atomic role $R \in \mathbf{R}$. We use $\mathbf{R}^-$ to denote the set of all inverses of roles in $\mathbf{R}$. An *RBox* $\mathcal{R}$ consists of a finite set of *transitivity axioms* $Trans(R)$, where $R \in \mathbf{R}$, and *role inclusion axioms* $R \sqsubseteq S$, where $R, S \in \mathbf{R} \cup \mathbf{R}^-$.

If $a \in \mathbf{I}$, then $\{a\}$ is a concept called *nominal*, and if C, C_1 and C_2 are concepts and $R \in \mathbf{R} \cup \mathbf{R}^-$, then $\geq nR.C$ and $\leq nR.C$ for an integer $n \geq 0$ are also concepts. A $\mathcal{SHOIQ}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{R}, \mathcal{A})$ consists of a TBox $\mathcal{T}$, an RBox $\mathcal{R}$ and an ABox $\mathcal{A}$.

The mapping $\cdot^{\mathcal{I}}$ is extended to all new concepts (where $\#X$ denotes the cardinality of the set X and $R^{\mathcal{I}}(x, C)$ is defined as $\{y | (x, y) \in R^{\mathcal{I}}, y \in C^{\mathcal{I}}\}$)

as:

$$(R^-)^{\mathcal{I}} = \{(y,x)|(x,y) \in R^{\mathcal{I}}\}$$
$$\{a\}^{\mathcal{I}} = \{a^{\mathcal{I}}\}$$
$$(\geq nR.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} | \#R^{\mathcal{I}}(x,C) \geq n\}$$
$$(\leq nR.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} | \#R^{\mathcal{I}}(x,C) \leq n\}$$

$\mathcal{SHOIQ}$ is decidable iff there are no number restrictions on roles which are transitive or have transitive subroles.

A query Q over a KB $\mathcal{K}$ is usually an axiom for which we want to test the entailment from the KB, written $\mathcal{K} \models Q$. The entailment test may be reduced to checking the unsatisfiability of a concept in the knowledge base, i.e., the emptiness of the concept. For example, the entailment of the axiom $C \sqsubseteq D$ may be tested by checking the unsatisfiability of the concept $C \sqcap \neg D$ while the entailment of the axiom $a : C$ may be tested by checking the unsatisfiability of $a : \neg C$.

Example 1 The following KB is inspired by the ontology `people+pets` [50]:

$$\exists hasAnimal.Pet \sqsubseteq NatureLover$$
$$\mathit{fluffy} : Cat$$
$$tom : Cat$$
$$Cat \sqsubseteq Pet$$
$$(kevin, \mathit{fluffy}) : hasAnimal$$
$$(kevin, tom) : hasAnimal$$

It states that individuals that own an animal which is a pet are nature lovers and that *kevin* owns the animals *fluffy* and *tom*, which are cats. Moreover, cats are pets. The KB entails the query $Q = kevin : NatureLover$.

3 Querying KBs: the Tableau Algorithm

In this section we introduce the problem of finding all the explanations for queries. In order to answer queries to DL KBs, one of the most common approaches is the *tableau algorithm* [1], described in details in the following. Then, we present TRILL and TRILL^P that implement the tableau algorithm. Finally, we discuss related work and present experiments for non-probabilistic querying.

3.1 Tableau Algorithm

A *tableau* is an ABox represented using a tuple $G = (V, E, \mathcal{L}, \dot{\neq})$ that contains a directed graph (V, E) where each node of V corresponds to an individual a and is labeled with the set of concepts $\mathcal{L}(a)$ to which a belongs. Each edge in $\langle a, b\rangle \in E$ in the graph is labeled with the set of roles $\mathcal{L}(\langle a, b\rangle)$. The binary predicate $\dot{\neq}$ is used to specify inequalities between nodes. G is initialized with a node for each individual a of the KB, labeled with the nominal $\{a\}$ plus all

concepts C such that $a : C \in \mathcal{K}$, and an edge $e = \langle a, b\rangle$ labeled with R for each assertion $(a, b) : R \in \mathcal{K}$.

A *tableau algorithm* proves an axiom by refutation, starting from a tableau that contains the negation of the axiom. For example, the axiom $C \sqsubseteq D$ can be proved by showing that $C \sqcap \neg D$ is empty, while, if the query is a class assertion, $C(a)$, we add $\neg C$ to the label of a. For testing the inconsistency of a concept C we have to test the emptiness of C by adding a new anonymous node a to the tableau whose label contains C. Then, the *tableau algorithm* repeatedly applies a set of consistency preserving *tableau expansion rules* until a clash (i.e., a contradiction) is detected or a clash-free graph is found to which no more rules are applicable. A clash is present when a node a contains in its label both C and $\neg C$ or when the tableau requires that two individuals a and b are the same and are different. If no clashes are found, the tableau represents a model for the negation of the query, which is thus not entailed.

In order to manage non-deterministic rules, a set T of completion graphs is built instead of a single one. T is initialized with a single completion graph G_0, and in the case of application of a non-deterministic rule, the tableau G_i on which the rule is applied is replaced by the set of tableaux returned by the rule.

For ensuring the termination of the algorithm, a special condition known as *blocking* [35] is used. In a tableau a node x can be a *nominal* node if its label $\mathcal{L}(x)$ contains a *nominal* or a *blockable* node. If there is an edge $e = \langle x, y\rangle$ then y is a *successor* of x and x is a *predecessor* of y. *Ancestor* is the transitive closure of predecessor while *descendant* is the transitive closure of successor. A node y is called an *R-neighbour* of a node x if y is a successor of x and $R \in \mathcal{L}(\langle x, y\rangle)$, where $R \in \mathbf{R}$.

An R-neighbour y of x is *safe* if (i) x is blockable or if (ii) x is a nominal node and y is not blocked. Finally, a node x is *blocked* if it has ancestors x_0, y and y_0 such that all the following conditions are true: (1) x is a successor of x_0 and y is a successor of y_0, (2) y, x and all nodes on the path from y to x are blockable, (3) $\mathcal{L}(x) = \mathcal{L}(y)$ and $\mathcal{L}(x_0) = \mathcal{L}(y_0)$, (4) $\mathcal{L}(\langle x_0, x\rangle) = \mathcal{L}(\langle y_0, y\rangle)$. In this case, we say that y *blocks* x. A node is blocked also if it is blockable and all its predecessors are blocked; if the predecessor of a safe node x is blocked, then we say that x is indirectly blocked.

When the algorithm stops applying rules to the tableaux contained in T, it means that every completion graph in T contains a clash or no more expansion rules can be applied to it. At this point, if all the completion graphs contain a clash, the algorithm could not find a model satisfying the query and returns unsatisfiable. Otherwise, it collects all the models from all the clash-free completion graphs in T and returns satisfiable. *Soundness* and *completeness* of the tableau algorithm are proved in [1].

3.2 Finding explanations to a query as *minimal axiom sets*

A relevant problem is that of finding the explanations for a query. Its solution has been investigated by various authors [62,35,27,36] and called *axiom pinpointing* in [62], where it is defined as a non-standard reasoning service useful for tracing derivations and debugging ontologies. In particular, *minimal axiom sets* or *MinAs* for short, also called *explanations*, are introduced in [62].

Definition 1 (MinA) Let $\mathcal{K}$ be a knowledge base and Q an axiom that follows from it, i.e., $\mathcal{K} \models Q$. We call a set $M \subseteq \mathcal{K}$ a *minimal axiom set* or *MinA* for Q in $\mathcal{K}$ if $M \models Q$ and it is minimal w.r.t. set inclusion.

The problem of enumerating all MinAs is called MIN-A-ENUM. The set of all MinAs for query Q in the knowledge base $\mathcal{K}$ is indicated with ALL-MINAS$(Q, \mathcal{K})$.

In order to build explanations, the tableau algorithm has been modified [27] so that each expansion rule updates as well a *tracing function* τ, which associates labels of nodes and edges with a subset of the axioms of the KB. τ is initialized to the empty set for all the elements of its domain except for $\tau(C, a)$ and $\tau(R, \langle a, b\rangle)$, to which the values $\{a : C\}$ and $\{(a, b) : R\}$ are assigned if $a : C$ and $(a, b) : R$ are in the ABox. The tableau expansion rules for $\mathcal{SHOIQ}$ are shown in Figure 1, where the rules for the $\mathcal{ALC}$ DL are marked by $(*)$. Here, $Add(C, a)$ stands for the addition of a concept C to $\mathcal{L}(a)$ while $Add(R, \langle a, b\rangle)$ represents the addition of a role R to $\mathcal{L}(\langle a, b\rangle)$. The values of the tracing function associated with the labels which causes the clash is then put together to form the explanation.

The rules in Figure 1 are divided into *deterministic* and *non-deterministic*. The first, when applied to a tableau, produces a single new tableau. The latter, when applied to a tableau, produce a set of tableaux. Let us now describe the rules, starting from the deterministic ones. Intuitively, the $\rightarrow$ *unfold* rule looks for a subclass axiom $C \sqsubseteq D$ in the KB and if there is an individual that belongs to C, it ensures that the label of the individual also contains D. The $\rightarrow CE$ rule looks for axioms of the form $C \sqsubseteq D$ in the KB where C is a complex concept and adds the disjunction $\neg C \sqcup D$ to the label of an individual. The third rule, $\rightarrow \sqcap$, adds to an individual two concepts C_1 and C_2 if its label contains the intersection of C_1 and C_2. The $\rightarrow \exists$ rule checks whether the concept $\exists S.C$ is verified for individual a labeled with it, i.e., if the individual a is linked through role S with at least an individual that belongs to C. If this is not verified, it adds to the tableau a new anonymous individual S-linked to a labeled with C. The rules $\rightarrow \forall$ and $\rightarrow \forall^+$ ensure that, given an individual a labeled with $\forall(S.C)$, all the individuals linked with a through S or its (transitive) subroles are labeled with the class C. Rule $\rightarrow \geq$ is similar to $\rightarrow \exists$, it looks for an individual a labeled with the concept $(\geq nS.C)$ and checks if a is linked with at least n individuals, instead of only one individual, through role S. The last deterministic rule, $\rightarrow O$, handles nominal nodes, i.e., nodes labeled with a concept defined by a set of individuals. If there are two nodes corresponding to two individuals a and b sharing a nominal concept in

Deterministic rules:

→ *unfold* (∗): **if** $A \in \mathcal{L}(a)$, A atomic and $(A \sqsubseteq D) \in K$, **then**
  if D $\notin \mathcal{L}(a)$, **then**
    $Add(D, a)$
    $\tau(D, a) := (\tau(A, a) \cup \{A \sqsubseteq D\})$

→ *CE* (∗): **if** $(C \sqsubseteq D) \in K$, with C not atomic, a not blocked, **then**
  if $(\neg C \sqcup D) \notin \mathcal{L}(a)$, **then**
    $Add((\neg C \sqcup D), a)$
    $\tau((\neg C \sqcup D), a) := \{C \sqsubseteq D\}$

→ ⊓ (∗): **if** $(C_1 \sqcap C_2) \in \mathcal{L}(a)$, a is not indirectly blocked, **then**
  if $\{C_1, C_2\} \not\subseteq \mathcal{L}(a)$, **then**
    $Add(\{C_1, C_2\}, a)$
    $\tau(C_i, a) := \tau((C_1 \sqcap C_2), a)$

→ ∃ (∗): **if** $\exists S.C \in \mathcal{L}(a)$, a is not blocked, **then**
  if a has no S-neighbour b with $C \in \mathcal{L}(b)$, **then**
    create new node b, $Add(S, \langle a, b\rangle), Add(C, b)$
    $\tau(C, b) := \tau((\exists S.C), a)$
    $\tau(S, \langle a, b\rangle) := \tau((\exists S.C), a)$

→ ∀ (∗): **if** $\forall(S.C) \in \mathcal{L}(a)$, a is not indirectly blocked and
      there is an S-neighbour b of a, **then**
  if $C \notin \mathcal{L}(b)$, **then**
    $Add(C, b)$
    $\tau(C, b) := \tau((\forall S.C), a) \cup \tau(S, \langle a, b\rangle)$

→ $\forall^+$: **if** $\forall(S.C) \in \mathcal{L}(a)$, a is not indirectly blocked and
      there is an R-neighbour b of a, $Trans(R)$ and $R \sqsubseteq S$, **then**
  if $\forall R.C \notin \mathcal{L}(b)$, **then**
    $Add(\forall R.C, b)$
    $\tau((\forall R.C), b) := \tau((\forall S.C), a) \cup \tau(R, \langle a, b\rangle) \cup \{Trans(R)\} \cup \{R \sqsubseteq S\}$

→≥: **if** $(\geq nS.C) \in \mathcal{L}(a)$, a is not blocked, **then**
  if there are no n safe S-neighbours $b_1, ..., b_n$ of a with $b_i \neq b_j$
    with C in $\mathcal{L}(b_i)$ for each b_i, **then**
    create n new nodes $b_1, ..., b_n$; $Add(S, \langle a, b_i\rangle), Add(C, b_i), \neq(b_i, b_j)$
    $\tau(S, \langle a, b_i\rangle) := \tau((\geq nS), a)$
    $\tau(C, b_i) := \tau((\geq nS.C), a) \cup \tau(S, \langle a, b\rangle)$
    $\tau(\neq(b_i, b_j)) := \tau((\geq nS), a)$

→ O: **if** $\{o\} \in \mathcal{L}(a) \cap \mathcal{L}(b)$ and not $a \neq b$, **then** $Merge(a, b)$
  $\tau(Merge(a, b)) := \tau(\{o\}, a) \cup \tau(\{o\}, b)$
  For each concept C_i in $\mathcal{L}(a)$ **then**
    $\tau(C_i, b) := \tau(C_i, a) \cup \tau(Merge(a, b))$
  (similarly for roles merged, and correspondingly for concepts in $\mathcal{L}(b)$)

Non-deterministic rules:

→ ⊔ (∗): **if** $(C_1 \sqcup C_2) \in \mathcal{L}(a)$, a is not indirectly blocked, **then**
  if $\{C_1, C_2\} \cap \mathcal{L}(a) = \emptyset$, **then**
    Generate graphs $G_i := G$ for each $i \in \{1, 2\}$
    $Add(C_i, a)$ in G_i for each $i \in \{1, 2\}$
    $\tau(C_i, a) := \tau((C_1 \sqcup C_2), a)$

→≤: **if** $(\leq nS.C) \in \mathcal{L}(a)$, a is not indirectly blocked,
  and there are m S-neighbours $b_1, ..., b_m$ of a with $m > n$
    with C in $\mathcal{L}(b_i)$ for each b_i, **then**
  For each possible pair b_i, b_j, $1 \leq i, j \leq m; i \neq j$ with $C \in \mathcal{L}(b_i) \cap \mathcal{L}(b_j)$ **then**
    Generate a graph G'
    $\tau(Merge(b_i, b_j)) := \tau((\leq nS.C), a) \cup \tau(S, \langle a, b_1\rangle) ... \cup \tau(S, \langle a, b_m\rangle)$
    if b_j is a nominal node, **then** $Merge(b_i, b_j)$ in G',
    else if b_i is a nominal node or ancestor of b_j, **then** $Merge(b_j, b_i)$
    else $Merge(b_i, b_j)$ in G'
    if b_i is merged into b_j, **then** for each concept C_i in $\mathcal{L}(b_i)$,
      $\tau(C_i, b_j) := \tau(C_i, b_i) \cup \tau(Merge(b_i, b_j))$
      (similarly for roles merged, and correspondingly for concepts in b_j
      if merged into b_i)

Fig. 1 TRILL tableau expansion rules; the subset of rules marked by (∗) is employed by TRILL^P.

their labels and there is not an axiom specifying that a and b are different individuals, then a and b are merged.

The non-deterministic $\rightarrow \sqcup$ rule searches for nodes labeled with a union of concepts and creates a set of new tableaux, one for each concept contained in the union, with the corresponding concept added to the label of the node. Finally, $\rightarrow\leq$ is applied to a node a labeled with the concept $(\leq nS.C)$. In this case, if there are more than n different nodes S-linked with a, the rule creates a new tableau for each pair of those individuals where the two individuals in the pair are merged into one, in order to reduce the number of S-neighbours of a.

The tableau algorithm as described up to now returns a single MinA using the tracing function. To solve MIN-A-ENUM, reasoners written in imperative languages, like Pellet [63], have to implement a search strategy in order to explore the entire search space of the possible explanations. In particular, Pellet, that is written entirely in Java, uses Reiter's *hitting set algorithm* [53]. The algorithm, described in detail in [35], starts from a MinA S and initializes a labelled tree called *Hitting Set Tree* (HST) with S as the label of its root v. Then it selects an arbitrary axiom E in S, it removes it from $\mathcal{K}$, generating a new knowledge base $\mathcal{K}' = \mathcal{K} - \{E\}$, and tries to obtain a new explanation for Q w.r.t. $\mathcal{K}'$. If a new MinA is found, the algorithm adds a new node w and a new edge $\langle v, w\rangle$ to the tree, then it assigns this new explanation to the label of w and the axiom E to the label of the edge. The algorithm repeats this process until the unsatisfiability test returns negative: in that case the algorithm labels the new node with OK, makes it a leaf, backtracks to a previous node, selects a different axiom to be removed from the KB and repeats these operations until the HST is fully built. The algorithm also eliminates extraneous unsatisfiability tests based on previous results: once a path leading to a node labelled OK is found, any superset of that path is guaranteed to be a path leading to a node where C is satisfiable, and thus no additional unsatisfiability test is needed for that path, as indicated by an X in the node label. When the HST is fully built, all leaves of the tree are labelled with OK or X. The distinct non leaf nodes of the tree collectively represent the set ALL-MINAS$(C, \mathcal{K})$.

The correctness of this approach [35] relies on the following key observations:

1. If a node is not a leaf of HST, then its label is an element of the set ALL-MINAS$(Q, \mathcal{K})$
2. If one takes the union of the labels of the edges in any path from the root of HST to a leaf node marked with OK, then a hitting set for ALL-MINAS$(Q, \mathcal{K})$ w.r.t. $\mathcal{K}$ is obtained. In fact, all the minimal hitting sets for ALL-MINAS$(Q, \mathcal{K})$ are obtained when all the paths from the root to a leaf in HST are considered.

Formally, the correctness and completeness of the hitting set algorithm is given by the following theorem.

Theorem 1 ([35]) *Let Q be a query that corresponds with a concept C unsatisfiable w.r.t. $\mathcal{K}$ and let* EXPHST$(Q, \mathcal{K})$ *be the set of explanations returned*

by the hitting set algorithm, then $\textsc{ExpHST}(Q, \mathcal{K})$ *is equal to the set of all explanations of unsatisfiability of the concept* Q *w.r.t.* $\mathcal{K}$*, so*

$$\textsc{ExpHST}(Q, \mathcal{K}) \equiv \textsc{All-MinAs}(Q, \mathcal{K})$$

Proof We divide the proof of equivalence in two parts showing that:

1. $\textsc{ExpHST}(Q, \mathcal{K}) \subseteq \textsc{All-MinAs}(Q, \mathcal{K})$
2. $\textsc{ExpHST}(Q, \mathcal{K}) \supseteq \textsc{All-MinAs}(Q, \mathcal{K})$

($\subseteq$ part)
Let $S \in \textsc{ExpHST}(Q, \mathcal{K})$, then S belongs to the label of some non-leaf node w in the hitting set tree HST generated by the algorithm. In this case, $\mathcal{L}(w) \in \textsc{All-MinAs}(Q, \mathcal{K}')$, for some $\mathcal{K}' \subseteq \mathcal{K}$. Therefore, $S \in \textsc{All-MinAs}(Q, \mathcal{K})$.

($\supseteq$ part)
Suppose, by contradiction, there exists a set $M \in \textsc{All-MinAs}(Q, \mathcal{K})$, but $M \notin \textsc{ExpHST}(Q, \mathcal{K})$. In this case, M does not coincide with the label of any node in HST. Let v_0 be the root of HST, with $\mathcal{L}(v_0) = \{E_1, ..., E_n\}$, if $M = \mathcal{L}(v_0)$ then there is a contradiction, otherwise there must be an $E_i \notin M$ and an edge of the graph whose label is E_i from which a path with a node labeled with M in it must be present. Since in the HST such a condition is not verified we have a contradiction. □

Example 2 (Tableau algorithm) Consider the following KB, similar to that of Example 1.

$$E_1 = kevin : \forall hasAnimal.Pet$$
$$E_2 = (kevin, tom) : hasAnimal$$
$$E_3 = tom : Cat$$
$$E_4 = Cat \sqsubseteq Pet$$

Let $Q = tom : Pet$ be the query. The initial tableau contains two nodes corresponding to *kevin* and *tom*. The concept $\forall hasAnimal.Pet$ is added to the label of *kevin* with the tracing function $\tau(\forall hasAnimal.Pet, kevin) = \{E_1\}$ while the label associated with *tom* is $\mathcal{L}(tom) = \{Cat, \neg Pet\}$ with the tracing function $\tau(Cat, tom) = \{E_4\}$ and $\tau(\neg Pet, tom) = \emptyset$. Finally, the node of *kevin* is linked with that of *tom* and the edge between them is labeled with *hasAnimal* and its tracing function is initialized as $\tau(hasAnimal, \langle kevin, tom\rangle) = \{E_2\}$. The initial tableau with all the labels with the corresponding values for the tracing function τ is shown in Figure 2.

Then, the tableau algorithm applies the $\rightarrow$ *unfold* rule to *tom* using axiom E_4, adding Pet to the label of *tom*. The tracing function τ is updated as: $\tau(Pet, tom) = \{E_3, E_4\}$

Now, there is a clash for the concept Pet, thus the algorithm stops and returns the explanation $\{E_3, E_4\}$. By applying the hitting set algorithm, at a certain point the axiom $E_4 = Cat \sqsubseteq Pet$ is removed from the KB and a new call to the tableau algorithm is executed. In this case, after the initialization of the tableau as in Figure 2, the $\rightarrow \forall$ rule is applied to *kevin*, adding Pet to the label of *tom*. The tracing function τ is updated as: $\tau(Pet, tom) = \tau((\forall hasAnimal.Pet), kevin) \cup \tau(hasAnimal, \langle kevin, tom\rangle) = \{E_1, E_2\}$. At

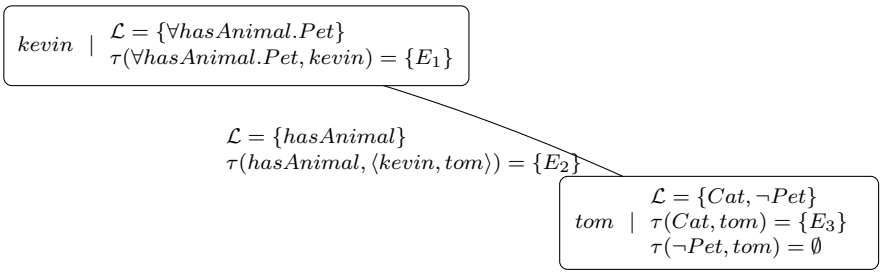


Fig. 2 Initial tableau for the Example 2. Each node of the tableau is boxed and contains to the left the name of the corresponding individual and to the right the label $\mathcal{L}$ and the tracing function τ. The edge is labeled with the set of roles that links the two individuals and the tracing function.

this point another clash is found and a new MinA $\{E_1, E_2\}$ is created. Since there are no more explanation, then ALL-MINAS$(Q, \mathcal{K}) = \{\{E_1, E_2\}, \{E_3, E_4\}\}$.

3.3 Finding explanations to a query as a *pinpointing formula*

In [3,4] the authors consider the problem of finding a *pinpointing formula* instead of ALL-MINAS$(Q, \mathcal{K})$. The pinpointing formula is a monotone Boolean formula in which each Boolean variable corresponds to an axiom of the KB. This formula is built using the variables and the conjunction and disjunction connectives. It compactly encodes the set of all MinAs. Let's assume that each axiom E of a KB $\mathcal{K}$ is associated with a propositional variable, indicated with $var(E)$. The set of all propositional variables is indicated with $var(\mathcal{K})$. A valuation ν of a monotone Boolean formula is the set of propositional variables that are true. For a valuation $\nu \subseteq var(\mathcal{K})$, let $\mathcal{K}_\nu := \{t \in \mathcal{K} | var(t) \in \nu\}$.

Definition 2 (Pinpointing formula) Given a query Q and a KB $\mathcal{K}$, a monotone Boolean formula ϕ over $var(\mathcal{K})$ is called a *pinpointing formula* for Q if for every valuation $\nu \subseteq var(\mathcal{K})$ it holds that $\mathcal{K}_\nu \models Q$ iff ν satisfies ϕ.

In Lemma 2.4 of [4] the authors proved that the set of all MinAs can be obtained by transforming the pinpointing formula into a Disjunctive Normal Form Boolean formula (DNF) and removing disjuncts implying other disjuncts.

In the following, we briefly define how a tableau algorithm can be modified to find the pinpointing formula. For more details and formal definitions see [4].

Given a KB $\mathcal{K}$, the modified algorithm associates a label $lab(a)$ that is a monotone Boolean formula over $var(\mathcal{K})$ with every assertion a. For deciding whether a rule is applicable we have to control the insertability of the new assertion. Let A be a set of labelled assertions and ψ a monotone Boolean formula, the assertion a is $\psi-$*insertable* into A if either $a \notin A$,

or $a \in A$ but $\psi \not\models lab(a)$. Given a set B of assertions and a set A of labelled assertions, the set of $\psi-insertable$ elements of B into A is defined as $ins_\psi(B, A) := \{b \in B | b \text{ is } \psi-insertable \text{ into } A\}$. For deciding the applicability of a rule we need also to give the definition of *substitution*. A substitution is a mapping $\rho : V \to D$, where V is a finite set of variables and D is a countably infinite set of constants that contains all the individuals in the KB and all the fresh individuals created by the application of the rules. Variables are seen as placeholder for individuals in the assertions. For example, an assertion can be $C(x)$ or $R(x, y)$ where C is a concept, R is a role and x and y are variables. In this case, let $C(x)$ be an assertion with the variable x and $\rho : x \to a$ a substitution, then $C(x)\rho$ denotes the assertion obtained by replacing the variable with its $\rho-image$, i.e. $C(a)$. A rule is of the form $(B_0, S) \to \{B_1, ..., B_m\}$ where B_i are finite sets of assertions and S is a finite set of axioms. A rule is applicable with a substitution ρ on the variable occurring in B_0 if $S \subseteq \mathcal{K}$, $B_0\rho \subseteq A$, where A is the set of assertions contained in the ABox and found during inference, and, for every $1 \leq i \leq m$ and every substitution ρ' on the variables occurring in $B_0 \cup B_i$, we have $ins_\psi(B_i\rho', A) \neq \emptyset$, where $\psi := \bigvee_{b \in B_0} lab(b\rho) \wedge \bigvee_{s \in S} var(s)$. Moreover, except for the $\to$ *unfold* rule, the node N to which the rule is applicable is not (indirectly) blocked. When the tableau is fully built, the algorithm conjoins the values of the tracing function of each clash for building the final pinpointing formula.

Example 3 (Tableau algorithm cont.) Consider the KB of Example 2 and the query $Q = tom : Pet$. The tableau is initialized as in the previous example and is shown in Figure 2. Then, the tableau algorithm applies the $\to$ *unfold* rule (defined as $(\{a : C\}, \{C \sqsubseteq D\}) \to \{\{a : C, a : D\}\}$) to tom adding Pet and updating the tracing function as: $\tau(Pet, tom) = E_3 \wedge E_4$

In the second iteration of rule application, the $\to \forall$ rule (defined as $(\{a : \forall S.C\}, \{(a, b) : S\}) \to \{\{a : \forall S.C, b : C\}\}$) is applied to $kevin$. Since the concept Pet already belongs to the label of tom, the resulting tracing function τ will contain the disjunction of two formulas: the previous value of $\tau(Pet, tom)$ and the new formula. Thus, the resulting tracing function is $\tau(Pet, tom) = \tau(Pet, tom) \vee (\tau((\forall hasAnimal.Pet), kevin) \cup \tau(hasAnimal, \langle kevin, tom\rangle)) = (E_3 \wedge E_4) \vee (E_1 \wedge E_2)$. At this point, the tableau is fully built and the algorithm returns the pinpointing formula $(E_3 \wedge E_4) \vee (E_1 \wedge E_2)$.

Example 4 Consider a more complicated KB shown in Example 1. We associate Boolean variables with axioms as follows:

$$
\begin{aligned}
E_1 &= \exists hasAnimal.Pet \sqsubseteq NatureLover \\
E_2 &= \mathit{fluffy} : Cat \\
E_3 &= tom : Cat \\
E_4 &= Cat \sqsubseteq Pet \\
E_5 &= (kevin, \mathit{fluffy}) : hasAnimal \\
E_6 &= (kevin, tom) : hasAnimal
\end{aligned}
$$

Let $Q = kevin : NatureLover$ be the query, then ALL-MINAS$(Q, \mathcal{K}) = \{\{E_5, E_2, E_4, E_1\}, \{E_6, E_3, E_4, E_1\}\}$, while the pinpointing formula is $((E_5 \wedge E_2) \vee (E_6 \wedge E_3)) \wedge E_4 \wedge E_1$.

3.4 TRILL and TRILL^P

Both TRILL and TRILL^P implement the tableau algorithm, the first solves MIN-A-ENUM while the second computes the pinpointing formula representing the set of all MinAs. They can answer concept and role membership queries, subsumption queries and can test the unsatisfiability of a concept of the KB or the inconsistency of the entire KB. TRILL and TRILL^P are implemented in Prolog, so the management of the non-determinism of the rules is delegated to the language. The code of TRILL and TRILL^P is available at `https://sites.google.com/a/unife.it/ml/trill`.

We use the Thea2 library [64] for converting OWL DL KBs into Prolog. Thea2 performs a direct translation of the OWL axioms into Prolog facts. For example, a simple subclass axiom between two named classes $Cat \sqsubseteq Pet$ is written using the `subClassOf/2` predicate as `subClassOf('Cat','Pet')`. For more complex axioms, Thea2 exploits the list construct of Prolog, so the axiom $NatureLover \equiv PetOwner \sqcup GardenOwner$ becomes `equivalentClasses([ 'NatureLover', unionOf(['PetOwner','GardenOwner'])])`.

In order to represent the tableau, TRILL and TRILL^P use a pair $Tableau = (A, T)$, where A is a list containing information about individuals and class assertions with the corresponding value of the tracing function. The tracing function stores a fragment of the knowledge base in TRILL and the pinpointing formula in TRILL^P. T is a triple (G, RBN, RBR) in which G is a directed graph that contains the structure of the tableau, RBN is a red-black tree (a key-value dictionary), where a key is a couple of individuals and its value is the set of the labels of the edge between the two individuals, and RBR is a red-black tree, where a key is a role and its value is the set of couples of individuals that are linked by the role. This representation allows to quickly find the information needed during the execution of the tableau algorithm. For managing the *blocking* system we use a predicate for each blocking state: `nominal/2`, `blockable/2`, `blocked/2`, `indirectly_blocked/2` and `safe/3`. Each predicate takes as arguments the individual Ind and the tableau (A, T); `safe/3`, shown in Figure 3, takes as input also the role R. For each individual Ind in the ABox, we add the atom $nominal(Ind)$ to A, then every time we have to check the blocking status of an individual we call the corresponding predicate that returns the status by checking the tableau.

Deterministic and non-deterministic tableau expansion rules are implemented following a different interface; this will facilitate the insertion of new rules in the future. All non-deterministic rules are implemented following the interface $rule\_name(Tab, TabList)$, thus they take as input the current tableau Tab and return the list of tableaux $TabList$ created by the application of the rule to Tab. Figure 4 shows the code of the non-deterministic rule $\rightarrow \sqcup$. The

```
safe(Ind,R,(ABox,(T,RBN,RBR))):-
  rb_lookup(R,V,RBR),
  member((X,Ind),V),
  blockable(X,(ABox,(T,RBN,RBR))),!.

safe(Ind,R,(ABox,(T,RBN,RBR))):-
  rb_lookup(R,V,RBR),
  member((X,Ind),V),
  nominal(X,(ABox,(T,RBN,RBR))),!,
  \+ blocked(Ind,(ABox,(T,RBN,RBR))).
```

Fig. 3 Code of the predicates `safe/3`. An `R`-neighbour `Ind` of `X` is safe if (i) `X` is blockable - corresponding with the first definition, where the predicate `blockable/2` is called - or if (ii) `X` is a nominal node and `Ind` is not blocked - checked by `nominal/2` and `blocked/2` -. The predicates `rb_lookup/3` and `member/2` are used to find an `R`-predecessor `X` to check if `Ind` is safe following the definition.

```
or_rule((A,T),L):-
  find((classAssertion(unionOf(LC),Ind),Expl),A),
  \+ indirectly_blocked(Ind,(A,T)),
  findall((A1,T),scan_or_list(LC,Ind,Expl,A,T,A1),L),
  dif(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).
```

Fig. 4 Code of the $\rightarrow \sqcup$ rule. It unifies the list `L` with all the tableau resulting by the application of the rule. `scan_or_list/6` simply check if the concept `C` can be added with the explanation `Expl` to the list or not.

predicate `or_rule/2` searches in the tableau (A, T) for an individual to which the rule can be applied and unifies L with the list of new tableaux created by `scan_or_list/6`. `find/2` implements the search for a class assertion. Since the data structure that stores class assertions is currently a list, `find/2` simply calls `member/2`. `absent/3` checks if the class assertion axiom with the associated explanation is already present in A, and in this case it checks the applicability of the expansion rule.

Deterministic rules are implemented by a predicate $rule\_name(Tab, Tab1)$ that, given the current tableau Tab, returns the tableau $Tab1$ obtained by the application of the rule to Tab. Figure 5 shows part of the code of the deterministic rule $\rightarrow$ *unfold*. The predicate `unfold_rule/2` searches in (A, T) for an individual to which the rule can be applied and calls the predicate `find_sub_sup_class/3` in order to find the class to be added to the label of the individual.

```
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),
  dif(C,D).
```

Fig. 5 Code of the → *unfold* rule. It takes an atomic class `C` from the input tableau and looks for a class `D` which is a superclass or an equivalent class of `C` and it is not already in the tableau, builds the explanation for the new class assertion found and adds it to the resulting tableau.

Expansion rules are applied in order by `apply_all_rules/2`, first the deterministic ones and then the non-deterministic ones, as shown in Figure 6. The predicate `apply_det_rules/3` takes as input the list of deterministic rules and the current tableau and returns a tableau obtained by the application of one of the rules. It is called as `apply_det_rules(RuleList,Tab,Tab1)`. After the application of a deterministic rule, a cut avoids backtracking to other possible choices for the deterministic rules.

Then, non-deterministic rules are tried sequentially with the predicate `apply_nondet_rules/3`, shown in Figure 6, that is called as `apply_nondet_rules(RuleList,Tab,Tab1)`. It takes as input the list of non-deterministic rules and the current tableau and returns a tableau obtained with the application of one of the rules. If a non-deterministic rule is applicable, the list of tableaux obtained by its application is returned by the predicate corresponding to the applied rule, a cut is performed to avoid backtracking to other rule choices and a tableau from the list is non-deterministically chosen with the `member/2` predicate. If no rule is applicable, the input tableau is returned and the rule application stops, otherwise a new round of rule application is performed.

Finally, the `findall/3` predicate is used on the set of the built tableaux for finding all the clashes contained in them in order to collect all the possible explanations.

Example 5 (Tableau algorithm cont.) Consider examples 2 and 3. Both TRILL and TRILL^P start applying expansion rules using `apply_all_rules/2`, which in turn calls the goal `apply_det_rules/3`. At this point, predicate `unfold_rule/2` succeeds and updates the tableau `Tab0`, returning it to `apply_all_rules/2`. Now, a new round of application of the rules is executed. As before, `apply_det_rules/3` is executed and the new tableau is re-

```
apply_all_rules(Tab0,Tab):-
  apply_det_rules([...],Tab0,Tab1),
  (Tab0=Tab1 *->
     Tab=Tab1;
     apply_all_rules(Tab1,Tab)
  ).

apply_det_rules([],Tab0,Tab):-
  apply_nondet_rules([...],Tab0,Tab).

apply_det_rules([H|_],Tab0,Tab):-
  call(H,Tab0,Tab),!.

apply_det_rules([_|T],Tab0,Tab):-
  apply_det_rules(T,Tab0,Tab).

apply_nondet_rules([],Tab,Tab).

apply_nondet_rules([H|_],Tab0,Tab):-
  call(H,Tab0,L),!,
  member(Tab,L),
  dif(Tab0,Tab).

apply_nondet_rules([_|T],Tab0,Tab):-
  apply_nondet_rules(T,Tab0,Tab).
```

Fig. 6 Application of the expansion rules by means of the predicates `apply_all_rules/2`, `apply_det_rules/3` and `apply_nondet_rules/3`. The list `[...]` contains the available rules and is different in TRILL and TRILL^P.

turned to `apply_all_rules/2`. Finally, no more deterministic rules are applicable thus `apply_det_rules/3` calls `apply_nondet_rules/3`. However, no non-deterministic rule is applicable thus the input tableau is returned as the final one and the explanation/pinpointing formula is returned to the user.

In each rule application round, the applicability of a rule is checked by looking whether its result is not already present in the tableau. This avoids both infinite loops in the rule application and considering alternative rules when a rule is applicable. In fact, if a rule is applicable in a tableau, it will also be so in any tableau obtained by the expansion of the original one. In this case, the choice of which expansion rule to apply introduces "don't care" non-determinism. Differently, "don't know" non-determinism is introduced by non-deterministic rules, since a single tableau is expanded into a set of tableaux. We use Prolog search only to handle "don't know" non-determinism.

In Figure 1, the symbol $(*)$ denotes the rules shared by TRILL and TRILL^P. In these rules, the operator $\cup$ for τ means union between two sets in TRILL, while in TRILL^P it joins two Boolean formulas with the `OR` Boolean operator. Moreover, when a concept is already present in a node label, TRILL checks whether to update the tracing function by verifying that the corresponding set of axioms is not a subset of τ, while TRILL^P performs a $\psi-$*insertability* test. The $\psi-$*insertability* test is done by means of a satisfiability solver. In par-

```
test(L1,L2):-
  build_f(L1,L2,F),
  cnf(F,Cnf),
  sat(Cnf).

build_f([L1],[L2],(F1*(-F2))):-
  build_f1(L1,F1,[],Var1),
  build_f1(L2,F2,Var1,_Var).
```

Fig. 7 Definition of the predicates `test/2` and `build_f/3`. In particular, `build_f1/4` takes as input a Boolean formula (the first argument) and a list containing the correspondence between Boolean variables and axioms (the third argument) and returns the Boolean formula in a format suitable for the predicate `cnf/2` (the second argument) and the updated correspondence list.

ticular, TRILL^P conjoins the negation of the pinpointing formula contained in the label of the individual in the tableau with the new Boolean formula to add to the label and tests the satisfiability of such formula. This step is performed by the `test/2` predicate shown in Figure 7. The predicate `test/2` first calls `build_f/3` which takes two Boolean formulas `L1` and `L2` and creates the conjunction that will be tested by means of the satisfiability solver. Predicates `cnf/2` and `sat/1` are defined in Prolog built-in libraries providing an interface to a sat solver. These libraries were originally presented in [16], where the authors described the implementation of an interface between Prolog and the MiniSat SAT solver [21], a small (about 1200 lines of C code) and efficient sat-solver. Predicate `cnf/2` converts a propositional formula `F`, in which the Boolean operators `and`, `or` and `not` are represented by `*`, `+` and `-` respectively, into a conjunctive normal form `Cnf`. Finally, `sat/1` takes as input such a conjunctive normal form formula and succeeds if it is satisfiable. If the test returns true, TRILL^P combines the two Boolean formulas with the `OR` Boolean operator.

3.5 Related Work

Usually, DL reasoners implement a tableau algorithm using a procedural language. Since some tableau expansion rules are non-deterministic, the developers have to implement a search strategy from scratch. Moreover, in order to solve MIN-A-ENUM, all different ways of entailing an axiom must be found. For example, Pellet [63] is a tableau reasoner for OWL, written in Java, which computes ALL-MINAS$(Q, \mathcal{K})$ by finding a single MinA using the tableau algorithm and then applying the hitting set algorithm to find all the other MinAs.

Reasoners written in Prolog can exploit its backtracking facilities for performing the search. This has been observed in various works. In [6] the authors proposed a tableau reasoner in Prolog for FOL based on free-variable semantic tableaux. However, the reasoner is not tailored to DLs. Meissner [47] presented the implementation of a Prolog reasoner for the DL $\mathcal{ALCN}$. This work was the basis of [30], that considered $\mathcal{ALC}$ and improved [47] by implementing

heuristic search techniques to reduce the running time. The work [22] added to [30] the possibility of returning explanations for queries but it still handled only $\mathcal{ALC}$.

In [32] the authors presented the KAON2 algorithm that exploits basic superposition, a refutational theorem proving method for FOL with equality, and a new inference rule, called decomposition, to reduce a $\mathcal{SHIQ}$ KB to a disjunctive datalog program.

DLog [42] is an ABox reasoning algorithm for the $\mathcal{SHIQ}$ language that permits storing the content of the ABox externally in a database and answers instance check and instance retrieval queries by transforming the KB into a Prolog program. TRILL differs from these works for the considered DL and from DLog for the capability of answering general queries.

A different approach is shown in [54], that introduced a system for reasoning on a logic-based ontology representation language, called OntoDLP, which is an extension of (disjunctive) ASP and can interoperate with OWL. This system, called OntoDLV, rewrites the OWL KB into the OntoDLP language, can retrieve information directly from external OWL ontologies and answers queries by using ASP. OntoDLV cannot find the set of explanations.

In [24] and [23] we addressed reasoning for Datalog$^{\pm}$ ontologies with an Abductive Logic Programming framework named $\mathcal{S}$CIFF, with existential and universal variables, and Constraint Logic Programming constraints in rule heads. The underlying abductive proof procedure can be directly exploited as an ontological reasoner for query answering and consistency checking.

3.6 Experiments

In order to test whether a Prolog implementation of the tableau algorithm can be competitive with an implementation using a procedural language we did a comparison with Pellet [63]. In this test we stressed the non-determinism given by the choice of which rule to apply. In particular we artificially created a set of KBs of increasing size of the following form:

$$
\begin{aligned}
&C_{1,1} \sqsubseteq C_{1,2} \sqsubseteq ... \sqsubseteq C_{1,n} \sqsubseteq C_{n+1}\\
&C_{1,1} \sqsubseteq C_{2,2} \sqsubseteq ... \sqsubseteq C_{2,n} \sqsubseteq C_{n+1}\\
&C_{1,1} \sqsubseteq C_{3,2} \sqsubseteq ... \sqsubseteq C_{3,n} \sqsubseteq C_{n+1}\\
&...\\
&C_{1,1} \sqsubseteq C_{m,2} \sqsubseteq ... \sqsubseteq C_{m,n} \sqsubseteq C_{n+1}
\end{aligned}
$$

with m and n that vary between 1 and 7. Finally, we add the assertion $a : C_{1,1}$ and we ran queries of the form $Q = a : C_{n+1}$. For each KB, m explanations can be found and every explanation will contain $n + 1$ axioms, n subclass axioms and 1 assertion axiom. The idea is to create an increasing number of backtracking points in order to test how the hitting set algorithm exploited by Pellet manages them. Table 1 reports the average time on 100 query execution

Table 1 Average time (in seconds) for computing all the explanations for queries with the reasoners TRILL, TRILL^P and Pellet. The cells containing "–" represent cases in which the execution was interrupted because of the very high computation time (10 minutes).

	Reasoner	1	2	3	4	5	6	7
1	**TRILL**	0.15	0.15	0.15	0.15	0.13	0.14	0.16
	TRILL^P	0.30	0.26	0.25	0.29	0.25	0.24	0.28
	Pellet	2.88	2.62	2.71	2.73	2.73	2.48	2.84
2	**TRILL**	0.15	0.16	0.14	0.14	0.15	0.16	0.13
	TRILL^P	0.24	0.27	0.28	0.28	0.32	0.30	0.25
	Pellet	2.79	2.70	2.72	2.66	3.06	2.85	2.98
3	**TRILL**	0.14	0.15	0.14	0.13	0.14	0.14	0.15
	TRILL^P	0.24	0.27	0.25	0.26	0.28	0.28	0.26
	Pellet	2.59	2.98	3.07	3.27	3.65	4.06	4.57
4	**TRILL**	0.14	0.14	0.14	0.15	0.14	0.15	0.13
	TRILL^P	0.27	0.30	0.28	0.31	0.30	0.30	0.27
	Pellet	2.83	3.29	3.61	4.97	6.51	7.91	11.11
5	**TRILL**	0.15	0.14	0.16	0.14	0.14	0.14	0.14
	TRILL^P	0.29	0.32	0.33	0.32	0.39	0.46	0.49
	Pellet	3.04	3.94	5.98	10.50	22.30	67.82	254.27
6	**TRILL**	0.16	0.16	0.14	0.14	0.16	0.15	0.16
	TRILL^P	0.29	0.29	0.35	0.47	0.51	0.50	0.51
	Pellet	3.32	5.19	12.43	69.05	568.04	–	–
7	**TRILL**	0.16	0.15	0.14	0.15	0.16	0.15	0.15
	TRILL^P	0.32	0.33	0.47	0.53	0.51	0.52	0.48
	Pellet	3.66	8.70	92.24	–	–	–	–

for each system and w.r.t. each KB when computing all the explanations for the query Q. Columns correspond to n while the rows corresponds to m. The cells filled with "–" mean that the execution was interrupted because of the very high computation time. We set a time limit of 10 minutes for the query execution. The results show that even small KBs may lead to high running time for Pellet, and that TRILL and TRILL^P manage such cases with extremely good performances.

4 Probabilistic Querying

DISPONTE [55,58] applies the distribution semantics [61] of probabilistic logic programming to DLs. In this section we first define DISPONTE and then we show how TRILL and TRILL^P can compute the probability of queries according to this semantics. Finally, we discuss related work and present experiments for probabilistic querying.

4.1 The DISPONTE semantics

In DISPONTE, a *probabilistic knowledge base* $\mathcal{K}$ contains a set of *probabilistic axioms* which take the form

$$p :: E \tag{1}$$

where p is a real number in $[0,1]$ and E is a DL axiom. The probability p can be interpreted as an *epistemic probability*, i.e., as the degree of our belief in the truth of axiom E. For example, a probabilistic concept membership axiom $p :: a : C$ means that we have degree of belief p in $C(a)$. A probabilistic concept inclusion axiom of the form $p :: C \sqsubseteq D$ represents the fact that we believe in the truth of $C \sqsubseteq D$ with probability p.

The idea of DISPONTE is to associate independent Boolean random variables with the axioms, which are pairwise independent. To obtain a *world* w we decide whether to include each axiom or not in w. A world therefore is a non-probabilistic KB that can be assigned a semantics in the usual way. A query is entailed by a world if it is true in every model of the world.

Formally, an *atomic choice* is a pair (E_i, k) where E_i is the ith probabilistic axiom and $k \in \{0,1\}$ indicates whether E_i is chosen to be included in a world ($k = 1$) or not ($k = 0$). A set of atomic choices κ is consistent if only one decision is taken for each axiom, i.e., $(E_i, k) \in \kappa, (E_i, m) \in \kappa$ implies $k = m$. A consistent set of atomic choices is also called *composite choice*. The probability of a composite choice κ is $P(\kappa) = \prod_{(E_i,1)\in\kappa} p_i \prod_{(E_i,0)\in\kappa}(1-p_i)$, where p_i is the probability associated with axiom E_i. A *selection* σ is a composite choice which contains an atomic choice (E_i, k) for every axiom of the theory. A selection σ identifies a theory w_σ called a *world* in this way: $w_\sigma = \{E_i | (E_i, 1) \in \sigma\}$. The probability of a world w_σ is $P(w_\sigma) = P(\sigma) = \prod_{(E_i,1)\in\sigma} p_i \prod_{(E_i,0)\in\sigma}(1-p_i)$. $P(w_\sigma)$ is a probability distribution over worlds, i.e., $\sum_{w\in\mathcal{W}_\mathcal{K}} P(w) = 1$, where $\mathcal{W}_\mathcal{K}$ is the set of all worlds. A composite choice κ *identifies* the set of worlds ω_κ that are "compatible" with κ: $\omega_\kappa = \{w_\sigma | w_\sigma \in \mathcal{W}_\mathcal{K}, \kappa \subseteq \sigma\}$. Similarly, a set of composite choices K identifies the set of worlds ω_K "compatible" with K: $\omega_K = \bigcup_{\kappa\in K} \omega_\kappa$. A composite choice κ is an *explanation* for a query Q if Q is entailed by every world of ω_κ. A *covering* set of explanations for Q is a set of composite choices K such that every world w_σ in which the query is true is included in the set of worlds identified by K.

We can now assign probabilities to queries. Given a world w, the probability of a query Q is defined as $P(Q|w) = 1$ if $w \models Q$ and 0 otherwise. The probability of a query can be defined by marginalizing the joint probability of the query and the worlds.

$$P(Q) = \sum_{w\in\mathcal{W}_\mathcal{K}} P(Q, w) \tag{2}$$

$$= \sum_{w\in\mathcal{W}_\mathcal{K}} P(Q|w)P(w) \tag{3}$$

$$= \sum_{w\in\mathcal{W}_\mathcal{K}: w\models Q} P(w) \tag{4}$$

The following example illustrates inference under the DISPONTE semantics.

Example 6 Consider the following KB, a probabilistic version of that proposed in Example 1.

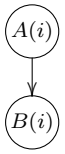
$$
\begin{aligned}
(E_1)\ 0.5\ ::\ & \exists hasAnimal.Pet \sqsubseteq NatureLover \\
& \mathit{fluffy} : Cat \\
& \mathit{tom} : Cat \\
(E_2)\ 0.6\ ::\ & Cat \sqsubseteq Pet \\
& (\mathit{kevin}, \mathit{fluffy}) : hasAnimal \\
& (\mathit{kevin}, \mathit{tom}) : hasAnimal
\end{aligned}
$$

It indicates that the individuals that own an animal which is a pet are nature lovers with a 50% probability and cats are pets with a 60% probability. These probabilities are degrees of belief in the respective axioms. For example, the fact that cats are pets is believed with degree 60%, meaning that the writer of the KB cannot be absolutely sure of the axiom, for example because she heard of wild cats, because she extracted the axiom with statistical methods from text, because she cannot define for sure the concepts of cats and pets, or for other reasons. The KB has four possible worlds:

$$\{(E_1), (E_2)\},\ \{(E_1)\},\ \{(E_2)\},\ \{\}$$

and the query axiom $Q = \mathit{kevin} : NatureLover$ is true in the first of them, while in the remaining ones it is false. The probability of the query is $P(Q) = 0.5 \cdot 0.6 = 0.3$.

The assumption of independence of the axioms may seem restrictive. However, any probabilistic relationship between assertions that can be represented with a Bayesian network can be modelled in this way. For example, suppose you want to model a general dependency between the assertions $A(i)$ and $B(i)$ regarding classes A and B and individual i. This dependency can be represented with the Bayesian network of Figure 8.



Prob. Table 1: $P(A(i))$

$A(i)$	
0	$1 - p_1$
1	p_1

Prob. Table 2: $P(B(i)|A(i))$

$B(i)$ \ $A(i)$	0	1
0	$1 - p_2$	$1 - p_3$
1	p_2	p_3

Fig. 8 Bayesian Network representing the dependency between $A(i)$ and $B(i)$.

The joint probability distribution $P(A(i), B(i))$ over the two Boolean random variables $A(i)$ and $B(i)$ is

$$\begin{aligned} P(0,0) &= (1-p_1)(1-p_2) \\ P(0,1) &= (1-p_1)(p_2) \\ P(1,0) &= p_1(1-p_3) \\ P(1,1) &= p_1 p_3 \end{aligned}$$

This dependence can be modelled with the following DISPONTE KB:

$$\begin{aligned} E_1 &= p_1 :: i : A \\ E_2 &= p_2 :: \neg A \sqsubseteq B \\ E_3 &= p_3 :: A \sqsubseteq B \end{aligned}$$

We can associate the Boolean random variables X_1 with (E_1), X_2 with (E_2) and X_3 with (E_3), where X_1, X_2 and X_3 are mutually independent. These three random variables generate 8 worlds. $\neg A(i) \wedge \neg B(i)$ is true in the worlds

$$w_1 = \{\},\ w_2 = \{(E_3)\}$$

whose probabilities are

$$\begin{aligned} P'(w_1) &= (1-p_1)(1-p_2)(1-p_3) \\ P'(w_2) &= (1-p_1)(1-p_2)p_3 \end{aligned}$$

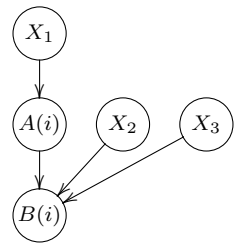
so $P'(\neg A(i), \neg B(i)) = (1-p_1)(1-p_2)(1-p_3) + (1-p_1)(1-p_2)p_3 = P(0,0)$. We can prove similarly that the distributions P and P' coincide for all joint states of $A(i)$ and $B(i)$.

Modelling the dependency between $A(i)$ and $B(i)$ with the KB above is equivalent to represent the Bayesian network of Figure 8 with the Bayesian network of Figure 9.

It can be easily checked that the distributions P and P'' of the two networks agree on the variables $A(i)$ and $B(i)$, i.e., that $P(A(i), B(i)) = P''(A(i), B(i))$ for any value of $A(i)$ and $B(i)$. From Figure 9 is also clear that X_1, X_2 and X_3 are mutually unconditionally independent, thus showing that it is possible to represent any dependence with independent random variables. So we can model general dependencies among assertions with DISPONTE.

4.2 Computing the probability

The number of different worlds is exponential in the number of probabilistic axioms, so their enumeration is unfeasible. A possible approach for computing the probability of queries to KBs following the DISPONTE semantics relies on the use of mutually exclusive explanations. Once collected, in fact, the probability of the query corresponds to the sum of the probabilities of the individual mutually exclusive explanations. The tableau algorithm shown in Section 3 and implemented in TRILL can be used to find the set of minimal explanations. In order to make them mutually exclusive we can exploit a



Prob. Table 3: $P''(X_1)$

X_1	
0	$1-p_1$
1	p_1

Prob. Table 4: $P''(X_2)$

X_2	
0	$1-p_2$
1	p_2

Prob. Table 5: $P''(A(i)|X_1)$

$A(i)$ \ X_1	0	1
0	1	0
1	0	1

Prob. Table 6: $P''(X_3)$

X_3	
0	$1-p_3$
1	p_3

Prob. Table 7: $P''(B(i)|X_1, X_2, X_3)$

$B(i)$ \ X_1, X_2, X_3	0,0,0	0,0,1	0,1,0	0,1,1	1,0,0	1,0,1	1,1,0	1,1,1
0	1	1	0	0	1	0	1	0
1	0	0	1	1	0	1	0	1

Fig. 9 Bayesian Network modelling the distribution over $A(i)$, $B(i)$, X_1, X_2, X_3.

splitting algorithm as shown in [52]. Alternatively, we can assign independent Boolean random variables to the axioms contained in the explanations and define the DNF Boolean formula f_K as $f_K(\mathbf{X}) = \bigvee_{\kappa \in K} \bigwedge_{(E_i,1)} X_i \bigwedge_{(E_i,0)} \overline{X_i}$ where $\mathbf{X} = \{X_i | (E_i, k) \in \kappa, \kappa \in K\}$ is the set of Boolean random variables.

TRILL^P, instead, computes directly a pinpointing formula which is a monotone Boolean formula that represents the set of all explanations.

Irrespective of which representation of the explanations we choose, a DNF or a general pinpointing formula, we can apply knowledge compilation and *transform it into a Binary Decision Diagram (BDD)*, from which we can compute the probability of the query with a dynamic programming algorithm that is linear in the size of the BDD.

A BDD for a function of Boolean variables is a rooted graph that has one level for each Boolean variable. A node n in a BDD has two children: one corresponding to the 1 value of the variable associated with the level of n, indicated with $child_1(n)$, and one corresponding to the 0 value of the variable, indicated with $child_0(n)$. When drawing BDDs, the 0-branch - the one going to $child_0(n)$ - is distinguished from the 1-branch by drawing it with a dashed line. The leaves store either 0 or 1.

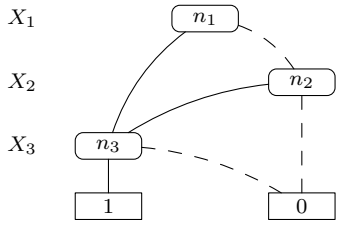


Fig. 10 BDD for Example 7.

Example 7 Consider the following probabilistic KB inspired by the `people+pets` ontology.

$$\begin{aligned}
& \exists hasAnimal.Pet \sqsubseteq NatureLover \\
& (kevin, fluffy) : hasAnimal \\
& (kevin, tom) : hasAnimal \\
E_1 = 0.4 \quad :: \quad & fluffy : Cat \\
E_2 = 0.3 \quad :: \quad & tom : Cat \\
E_3 = 0.6 \quad :: \quad & Cat \sqsubseteq Pet
\end{aligned}$$

Here individuals that own an animal which is a pet are surely nature lovers and *kevin* has the animals *fluffy* and *tom*. Moreover, we believe in the fact that *fluffy* and *tom* are cats and that cats are pets with a certain probability. A covering set of explanations for the query axiom $Q = kevin : NatureLover$ is $K = \{\kappa_1, \kappa_2\}$ where $\kappa_1 = \{(E_1, 1), (E_3, 1)\}$ and $\kappa_2 = \{(E_2, 1), (E_3, 1)\}$.

If we associate the random variables X_1 with E_1, X_2 with E_2 and X_3 with E_3, the BDD associated with the set K of explanations is shown in Figure 10.

A BDD performs a Shannon expansion of the Boolean formula $f(\mathbf{X})$, so that, if X is the variable associated with the root level of a BDD, the formula $f(\mathbf{X})$ can be represented as $f(\mathbf{X}) = X \wedge f^X(\mathbf{X}) \vee \overline{X} \wedge f^{\overline{X}}(\mathbf{X})$ where $f^X(\mathbf{X})$ ($f^{\overline{X}}(\mathbf{X})$) is the formula obtained by $f(\mathbf{X})$ by setting X to 1 (0). Now the two disjuncts are pairwise exclusive and the probability of $f(\mathbf{X})$ being true can be computed as $P(f(\mathbf{X})) = P(X)P(f^X(\mathbf{X})) + (1 - P(X))P(f^{\overline{X}}(\mathbf{X}))$ by knowing the probabilities $P(X)$ of the Boolean variables of being true.

4.3 Related Work

First-order logics of probability were discussed in [5] and [28] where two different types of probability, statistical and epistemic, were presented, the first type captures statistical information about the world while the second captures a degree of belief. For example, the first type represents probabilistic statements such as "the probability that a randomly chosen bird flies is 0.9", while the latter represents probabilistic statements such as "the probability

that Tweety (a particular bird) flies is 0.9". DISPONTE allows only Type 2 statements: $0.9 :: Bird \sqsubseteq Flies$ means that each particular bird flies with a probability of 0.9.

Prob-$\mathcal{ALC}$ [46], as DISPONTE, considers only epistemic probabilities. It follows a possible world semantics and allows concept expressions of the form $P_{\geq n}C$ and $\exists P_{\geq n}R.C$ in the TBox, indicating the set of individuals that belong to C with probability greater than n and the set of individuals a connected to at least another individual b of C by role R such that the probability of $R(a, b)$ is greater than n respectively. Axioms of the form $P_{\geq n}C(a)$ and $P_{\geq n}R(a, b)$ are allowed in the ABox. Prob-$\mathcal{ALC}$ is complementary to DISPONTE $\mathcal{ALC}$ as it allows new concept and assertional expressions while DISPONTE allows probabilistic axioms.

In [34] the authors presented an approach for querying probabilistic databases in the presence of an OWL2 QL ontology. Each assertion is assumed to be stored in a database and associated with probabilistic events that are probabilistically independent, resulting in a semantics very similar to the distribution semantics. The authors addressed the problem of computing probabilities to conjunctive queries w.r.t. datasets where probabilities can occur only in assertional informations. Events can be Boolean combinations of atomic events in general, but the authors considered also a restricted setting where each assertion is associated with a distinct atomic event. This setting is subsumed by DISPONTE.

Heinsohn [29] extended the DL $\mathcal{ALC}$ in order to allow the expression of statistical information on the terminological knowledge such as partial concept overlapping. Jaeger [33] extended [29] by allowing the definition of probabilistic assertions and combines the resulting probability distributions using cross-entropy minimization. Similarly, [39] presented a probabilistic description logic based on Bayesian networks that deals with statistical terminological knowledge.

In [20] the authors proposed a probabilistic extension of OWL that admits a translation into Bayesian networks. The semantics assigns a probability distribution $P(a)$ over individuals, i.e. the probability values of all individuals sum up to 1. Classes can be assigned a probability defined as the sum of the probabilities of the individuals belonging the class.

PR-OWL [17,12] is an upper ontology that provides a framework for building probabilistic ontologies. It allows to use the first-order probabilistic logic MEBN [41] for representing uncertainty in ontologies.

DISPONTE differs from [29,33,39,20,17,12] because it minimally extends DLs and provides a unified framework for representing different types of probabilistic knowledge: from assertional to terminological degree of belief knowledge.

FOProbLog [10] is an extension of ProbLog where a program contains a set of *probabilistic facts*, i.e. facts annotated with probabilities, and a set of general clauses which can have positive and negative probabilistic facts in their body. Each fact is assumed to be probabilistically independent. FOProbLog follows the distribution semantics and exploits BDDs to compute the probability of

queries. FOProbLog is a reasoner for FOL that is not tailored to DLs, so the algorithm could be suboptimal for them.

Bayesian networks are exploited also in [18] and [14]. Here, a KB is associated with a Bayesian network with variables V. Axioms take the form $E : X = x$ where E is a DL axiom and $X = x$ is an annotation with $X \subseteq V$ and x a set of values for these variables. The Bayesian network assigns a probability to every assignment of V, called a world. The authors show that the probability of a query $Q = E : X = x$ is given by the sum of the probabilities of the worlds where $X = x$ is satisfied and where E is a logical consequence of the theory composed of the axioms whose annotation is true in the world. A similar approach was presented in [26] where Markov networks are used instead of Bayesian networks. The approach of [14] was applied in [15,13] to extend the $\mathcal{EL}$ DL, defining the probabilistic DL called $\mathcal{BEL}$. The system BORN [13] answers probabilistic subsumption queries w.r.t. $\mathcal{BEL}$ KBs. It exploits ProbLog for managing the probabilistic part of the KB. DISPONTE is a special case of these semantics where every axiom $E_i : X_i = x_i$ is such that X_i is a single Boolean variable and the Bayesian network has no edges, i.e., all the variables are independent. This is an important special case that greatly simplifies reasoning, as computing the probability of the worlds takes a time linear in the number of variables. However, in case the added expressiveness of these formalisms is needed, the Bayesian network could be translated into an equivalent one with only mutually unconditionally independent random variables as shown in Figure 9.

CR$\mathcal{ALC}$ [45] extends $\mathcal{ALC}$ by allowing the definition of statistical axioms only. It adopts an interpretation-based semantics based on probability measures over the space of interpretations with a fixed domain. Inference can then be performed by a first order loopy belief propagation algorithm on ground direct acyclic graphs which represents the KBs w.r.t. queries. These graphs have a node for each concept, role and restriction $\exists R.C$ and $\forall R.C$, the edges connect related concepts and roles.

A different approach to the combination of DLs with probability is taken in [25,43,44]. The logic proposed in these papers is called P-$\mathcal{SHIQ}(\mathbf{D})$ and uses probabilistic lexicographic entailment from probabilistic default reasoning. It allows both terminological probabilistic knowledge as well as assertional probabilistic knowledge about instances of concepts and roles. In particular, TBox probabilistic axioms are expressed through *conditional constraints* of the form $(D|C)[l, u]$ that informally mean "generally, if an object belongs to C, then it belongs to D with a probability in $[l, u]$". PRONTO [37] is a system that can be used to perform inference. P-$\mathcal{SHIQ}(\mathbf{D})$ is based on Nilsson's probabilistic logic [49] that defines probabilistic interpretations instead of a single probability distribution over theories. Each probabilistic interpretation Pr defines a probability distribution over the set of usual interpretations Int. The probability of a logical formula F according to Pr, denoted $Pr(F)$, is the sum of all $Pr(I)$ such that $I \in Int$ and $I \models F$. A probabilistic KB K is a set of probabilistic formulas of the form $F \geq p$. A probabilistic interpretation Pr satisfies $F \geq p$ iff $Pr(F) \geq p$. A probabilistic interpretation Pr is a model of a

probabilistic KB if it satisfies all of its probabilistic formulas. The probability of a logical formula F according to a probabilistic KB K belongs to the set of values $\{Pr(F)|Pr \text{ is a model of } K\}$. So with Nilsson logic one reasons with intervals of probability values. $Pr(F) \geq p$ is a tight logical consequence of K iff p is the infimum of $Pr(F)$ subject to all models Pr of K.

A combination between DLs and logic programs was presented in [11] in order to integrate ontologies and rules. They use a tightly coupled approach to (probabilistic) disjunctive description logic programs. They define a description logic program as a pair (L, P), where L is a DL KB and P is a disjunctive logic program which contains rules on concepts and roles of L. P may contain probabilistic alternatives in the style of ICL [51]. Interpretations assign a probability to ground atoms, in the style of Nilsson probabilistic logic [49].

Nilsson's logic allows weaker conclusions than the distribution semantics: consider a probabilistic ontology composed of the axioms $0.4 :: a : C$ and $0.5 :: b : C$ and a probabilistic KB composed of $C(a) \geq 0.4$ and $C(b) \geq 0.5$. The distribution semantics allows us to say that $P(a : C \vee b : C) = 0.7$, while with Nilsson's logic we can only say that $Pr(C(a) \vee C(b)) \in [0.5, 1]$. This is due to the fact that in the distribution semantics the probabilistic axioms are considered as independent, which allows to make stronger conclusions. However, this does not restrict expressiveness as one can specify any joint probability distribution over the logical ground atoms, possibly introducing new atoms if needed as is shown in Section 4.1.

The use of Nilsson logic causes the approaches of [25, 43, 44, 11] to reason with intervals of probability values, while with DISPONTE we work with point probabilities.

Recently, BUNDLE [56, 59] was proposed for reasoning over DISPONTE KBs. BUNDLE exploits Pellet for solving MIN-A-ENUM and computes the probability of queries by exploiting BDDs.

4.4 Experiments

In order to evaluate TRILL and TRILL^P for probabilistic querying, we compared them with BUNDLE [56, 59] and BORN [13]. We used four different knowledge bases of various complexity:

- BRCA[1], which models the risk factors of breast cancer;
- an extract of the DBPedia[2] ontology obtained from Wikipedia;
- Biopax level 3[3], which models metabolic pathways;
- Vicodi[4], which contains information on European history.

BRCA ("Breast Cancer Risk Assessment") [38] states the risk factors of breast cancer depending on several factors such as age, drugs taken, ethnicity, etc. For

[1] `http://www2.cs.man.ac.uk/~klinovp/pronto/brc/cancer_cc.owl`
[2] `http://dbpedia.org/`
[3] `http://www.biopax.org/`
[4] `http://www.vicodi.org/`

```
...
<owl:Class rdf:about="&cancer_ra;LackOfExercise">
  <rdfs:subClassOf rdf:resource="&cancer_ra;KnownFactor"/>
</owl:Class>

<owl:Class rdf:about="&cancer_ra;LateFirstChild">
  <rdfs:subClassOf rdf:resource="&cancer_ra;KnownFactor"/>
</owl:Class>
...
```

Fig. 11 Axioms from BRCA. They state that the lack of exercise may increase the risk of having breast cancer as well as having the first child at old age.

```
...
<owl:Class rdf:about="&dbpedia;ComicsCreator">
  <rdfs:subClassOf rdf:resource="&dbpedia;Person"/>
</owl:Class>
...
<owl:Class rdf:about="&dbpedia;Hospital">
  <rdfs:subClassOf rdf:&dbpedia;Place"/>
</owl:Class>
...
```

Fig. 12 Axioms from DBPedia. They state that a comic creator is a person and that hospitals are places.

example, as shown in Figure 11, it states that known factors for the disease are the lack of exercise or having the first child at a late age. The original version presented in [38] reduced risk assessment to probabilistic entailment in P-$\mathcal{SHIQ}(\mathbf{D})$. For this test we took only the non-probabilistic part and we made probabilistic 50 axioms randomly chosen. BRCA has $\mathcal{ALCHF}(\mathbf{D})$ expressiveness and contains 491 axioms, 154 different classes and 15 different properties (12 object properties and 3 data properties).

DBPedia contains structured information about Wikipedia. It is built on the data contained in the sideboxes shown in wiki pages. Some examples of axioms are shown in Figure 12. DBPedia's expressiveness is $\mathcal{EL}$. The portion we consider contains 267 axioms and 118 classes.

Biopax [19] was defined in order to allow integration on analysis of biological pathways data. The BioPax project defines 3 different levels, in this test we used level 3 that represents molecular and genetic interactions together with pathways including molecular states. Figure 13 shows the axioms asserting that physical entities are composed of complexes which are in turn physical entities different from DNAs, RNAs or small molecules. The version of Biopax used has $\mathcal{SHIN}(\mathbf{D})$ expressiveness and has 925 axioms, 69 classes, 55 object properties and 41 data properties.

Finally, Vicodi [48] is an extract of the Vicodi knowledge base that contains information on European history. It models historical events and important personalities such as popes or princes, as shown in Figure 14. Vicodi's expres-

```
...
<owl:ObjectProperty rdf:about="&biopax;component">
  <rdf:type rdf:resource="&owl;InverseFunctionalProperty"/>
  <rdfs:domain rdf:resource="&biopax;Complex"/>
  <rdfs:range rdf:resource="&biopax;PhysicalEntity"/>
</owl:ObjectProperty>
...
<owl:Class rdf:about="&biopax;Complex">
  <rdfs:subClassOf rdf:resource="&biopax;PhysicalEntity"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&biopax;memberPhysicalEntity"/>
      <owl:allValuesFrom rdf:resource="&biopax;Complex"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="&biopax;Dna"/>
  <owl:disjointWith rdf:resource="&biopax;DnaRegion"/>
  <owl:disjointWith rdf:resource="&biopax;Protein"/>
  <owl:disjointWith rdf:resource="&biopax;Rna"/>
  <owl:disjointWith rdf:resource="&biopax;RnaRegion"/>
  <owl:disjointWith rdf:resource="&biopax;SmallMolecule"/>
  ...
</owl:Class>
...
```

Fig. 13 Axioms from Biopax. A complex is a physical entity different from DNAs, RNAs or their regions, proteins and small molecules. It can be part of another physical entity.

```
...
<owl:Class rdf:about="&vicodi;Pope">
  <rdfs:subClassOf rdf:resource="&vicodi;Religious-Leader"/>
</owl:Class>
...
<owl:Class rdf:about="&vicodi;Prince">
  <rdfs:subClassOf rdf:resource="&vicodi;Head-of-State"/>
</owl:Class>
...
<owl:Class rdf:about="&vicodi;War">
  <rdfs:subClassOf rdf:resource="&vicodi;Event"/>
</owl:Class>
...
```

Fig. 14 Axioms from Vicodi. A Pope is a religious leader, while a prince is a head of a state. Finally a war is an historical event.

siveness is $\mathcal{ALH}(\mathbf{D})$ and contains 209 axioms, 168 classes, 6 object properties and 2 data properties

For the tests, we used a version of the DBPedia and Biopax KBs without the ABox and a version of BRCA and of Vicodi with an ABox containing 1 individual and 19 individuals respectively. For each KB we made probabilistic 50 of its axioms. The probability values were learned using EDGE [57], a system for parameter learning from a set of positive and negative examples in

Table 2 Average number of MinAs and average time (in seconds) for computing the probability of queries with the reasoners TRILL, TRILL^P and BUNDLE.

Dataset	avg. n. of minAs	TRILL time (s)	TRILL^P time (s)	BUNDLE time (s)	BORN time (s)
BRCA	6.49	27.87	4.74	6.96	n.a.
DBPedia	16.32	51.56	4.67	3.79	5.16
Biopax level 3	3.92	0.12	0.12	1.85	n.a.
Vicodi	1.02	0.19	0.19	1.12	n.a.

the form of class assertion axioms that we regard as true (false), and for which we would like to get a high (low) probability.

For each dataset we randomly created 100 different queries. In particular, for the DBPedia and Biopax datasets, we created 100 subclass-of queries, while for the other KBs we created 80 subclass-of and 20 instance-of queries. For generating the subclass-of queries, we randomly selected two classes that are connected in the hierarchy of classes, so that each query had at least one explanation. For the instance-of queries, we randomly selected an individual a and a class to which a belongs by following the hierarchy of the classes, starting from the classes to which a explicitly belongs in the KB.

Table 2 shows, for each ontology, the average number of different MinAs computed and the average time in seconds that TRILL, TRILL^P, BUNDLE and BORN took for computing the probability of the queries. In particular, BRCA and the used version of DBPedia contain a large number of subclass axioms between complex concepts resulting in a large number of explanations. TRILL^P performs better than TRILL because it is able to compactly encode explanations. In the last column, "n.a." means "not applicable" as BORN can handle only $\mathcal{EL}$ DL.

A second test was performed following the approach presented in [38] to investigate the scalability of our systems on versions of BRCA of increasing size, comparing them with PRONTO, the system for which BRCA was originally created, and BUNDLE. The authors of [38] included in the KB an increasing number of conditional constraints randomly created. The number of these constraints was varied from 9 to 15, and, for each number, 100 different consistent ontologies were created. To convert them into DISPONTE KBs, we translated every conditional constraint $(C|D)[l, u]$ into a probabilistic axiom of the form $u :: C \sqsubseteq D$. For instance, the statement that an average woman has up to 12.3% chance of developing breast cancer in her lifetime is expressed by $(WomanUnderAbsoluteBRCRisk|Woman)[0, 0.123]$ is translated into $0.123 :: WomanUnderAbsoluteBRCRisk \sqsubseteq Woman$. Finally, an individual was added to every KB. The individual is randomly assigned to each simple class that appears in the conditional constraints with probability 0.6. Complex classes contained in the conditional constraints were split into their components, e.g., the complex class *PostmenopausalWomanTakingTestosterone* was divided into *PostmenopausalWoman* and *WomanTakingTestosterone*. Finally, we ran 100 probabilistic queries of the form $a : C$ where a is the individual of

the KB and C is a class randomly selected among those that represent women under increased and lifetime risk such as *WomanUnderLifetimeBRCRisk* and *WomanUnderStronglyIncreasedBRCRisk*.

Figure 15 shows the execution time and the memory consumption averaged over the 100 KBs as a function of the number of probabilistic axioms. As one can see, TRILL is initially slower than all the other systems but then inference times becomes similar to that of BUNDLE. TRILL^P performances are close to that of BUNDLE and outperform PRONTO. The memory usage for both TRILL and TRILL^P is much lower than the other systems.

These tests show that both TRILL and TRILL^P can sometimes be better than other state-of-the-art probabilistic reasoners such as PRONTO and BUNDLE, even if they lack all the optimizations implemented in them. In particular, when a KB is relatively small, BUNDLE and PRONTO's higher cost is due to their expensive initialization phase that is not present in TRILL and TRILL^P. These results represent evidence that a Prolog implementation of probabilistic tableau reasoners is feasible and may lead to practical systems. Moreover, TRILL^P is faster than TRILL when more MinAs are present.

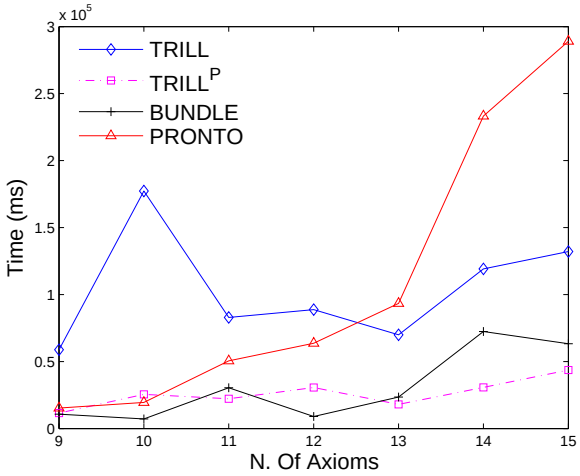
5 Conclusions

In this paper we have presented the algorithm TRILL for reasoning on $\mathcal{SHOIQ}$ KBs and the algorithm TRILL^P for reasoning on $\mathcal{ALC}$ KBs. Both reasoners implement the tableau algorithm in Prolog.

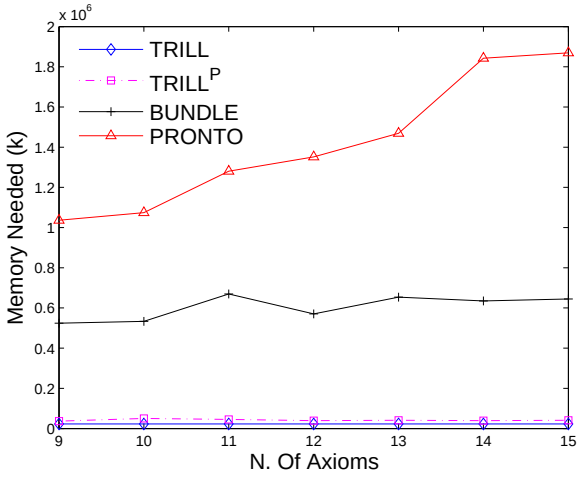
The experiments performed show that Prolog is a viable language for implementing DL reasoning algorithms and that their performances are comparable with those of a state-of-the-art reasoner such as Pellet. Moreover, we show how they can be extended to compute the probability of queries from DISPONTE KBs. A comparison with the probabilistic reasoners BUNDLE, PRONTO and BORN show that again Prolog may play a role in this field.

In order to spread the use of TRILL, we developed a Web application called "TRILL-on-SWISH" and available at `http://trill.lamping.unife.it`. The application is based on SWISH [40], a recently proposed Web framework for logic programming using various features and packages of SWI-Prolog. SWISH allows the user to write Prolog programs and ask queries in the browser without installing SWI-Prolog. TRILL-on-SWISH allows users to write a KB in the RDF/XML format directly in the web page or load it from a URL, and specify queries that are answered by TRILL running on the server. Once the computation ends, the results are sent to the client browser and visualized in the Web page.

In the future we plan to apply various optimizations to our systems in order to better manage the expansion of the tableau. In particular, we plan to carefully choose the rule and node application order. We are also studying an extension of our systems for managing KBs integrating rules and DL axioms. Moreover, we plan to exploit TRILL for implementing algorithms for learning the parameters of probabilistic DISPONTE KBs, along the lines of [8,9,60].



(a) Average execution time (ms) for inference.



(b) Average memory consumption (Kb) for inference.

Fig. 15 Comparison between TRILL, TRILL^P, BUNDLE and PRONTO on BRCA KB.

References

1. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press (2003)
2. Baader, F., Horrocks, I., Sattler, U.: Description logics. In: Handbook of knowledge representation, chap. 3, pp. 135–179. Elsevier (2008)
3. Baader, F., Peñaloza, R.: Automata-based axiom pinpointing. J. Autom. Reasoning **45**(2), 91–129 (2010)

4. Baader, F., Peñaloza, R.: Axiom pinpointing in general tableaux. J. Log. Comput. **20**(1), 5–34 (2010)
5. Bacchus, F.: Representing and reasoning with probabilistic knowledge - a logical approach to probabilities. MIT Press, Cambridge, MA, USA (1990)
6. Beckert, B., Posegga, J.: leantap: Lean tableau-based deduction. J. Autom. Reasoning **15**(3), 339–358 (1995)
7. Bellodi, E., Lamma, E., Riguzzi, F., Albani, S.: A distribution semantics for probabilistic ontologies. In: F. Bobillo, et al. (eds.) URSW 2011, *CEUR Workshop Proceedings*, vol. 778. Sun SITE Central Europe (2011)
8. Bellodi, E., Riguzzi, F.: Learning the structure of probabilistic logic programs. In: S.H. Muggleton, A. Tamaddoni-Nezhad, F.A. Lisi (eds.) ILP 2011, *LNCS*, vol. 7207, pp. 61–75. Springer (2012). DOI 10.1007/978-3-642-31951-8_10
9. Bellodi, E., Riguzzi, F.: Expectation Maximization over binary decision diagrams for probabilistic logic programs. Intel. Data Anal. **17**(2), 343–363 (2013)
10. Bruynooghe, M., Mantadelis, T., Kimmig, A., Gutmann, B., Vennekens, J., Janssens, G., Raedt, L.D.: Problog technology for inference in a probabilistic first order logic. In: ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings, *Frontiers in Artificial Intelligence and Applications*, vol. 215, pp. 719–724. IOS Press (2010)
11. Calì, A., Lukasiewicz, T., Predoiu, L., Stuckenschmidt, H.: Tightly coupled probabilistic description logic programs for the semantic web. In: Journal on Data Semantics XII, pp. 95–130. Springer (2009)
12. Carvalho, R.N., Laskey, K.B., Costa, P.C.G.: PR-OWL 2.0 - bridging the gap to OWL semantics. In: F. Bobillo, et al. (eds.) URSW 2010, *CEUR Workshop Proceedings*, vol. 654. Sun SITE Central Europe (2010)
13. Ceylan, İ.İ., Mendez, J., Peñaloza, R.: The bayesian ontology reasoner is born! In: M. Dumontier, B. Glimm, R.S. Gonçalves, M. Horridge, E. Jiménez-Ruiz, N. Matentzoglu, B. Parsia, G.B. Stamou, G. Stoilos (eds.) Informal Proceedings of the 4th International Workshop on OWL Reasoner Evaluation (ORE-2015) co-located with the 28th International Workshop on Description Logics (DL 2015), *CEUR Workshop Proceedings*, vol. 1387, pp. 8–14. CEUR-WS.org (2015)
14. Ceylan, İ.İ., Peñaloza, R.: Bayesian description logics. In: M. Bienvenu, M. Ortiz, R. Rosati, M. Simkus (eds.) Informal Proceedings of the 27th International Workshop on Description Logics, Vienna, Austria, July 17-20, 2014., *CEUR Workshop Proceedings*, vol. 1193, pp. 447–458. CEUR-WS.org (2014)
15. Ceylan, İ.İ., Peñaloza, R.: Probabilistic query answering in the bayesian description logic *BE*l. In: C. Beierle, A. Dekhtyar (eds.) Proceedings of Scalable Uncertainty Management - 9th International Conference, SUM 2015, *Lecture Notes in Computer Science*, vol. 9310, pp. 21–35. Springer (2015)
16. Codish, M., Lagoon, V., Stuckey, P.J.: Logic programming with satisfiability. TPLP **8**(1), 121–128 (2008)
17. da Costa, P.C.G., Laskey, K.B., Laskey, K.J.: PR-OWL: A bayesian ontology language for the semantic web. In: P.C.G. da Costa, C. d'Amato, N. Fanizzi, K.B. Laskey, K.J. Laskey, T. Lukasiewicz, M. Nickles, M. Pool (eds.) Uncertainty Reasoning for the Semantic Web I, ISWC International Workshops, URSW 2005-2007, Revised Selected and Invited Papers, *Lecture Notes in Computer Science*, vol. 5327, pp. 88–107. Springer (2008)
18. d'Amato, C., Fanizzi, N., Lukasiewicz, T.: Tractable reasoning with bayesian description logics. In: S. Greco, T. Lukasiewicz (eds.) Scalable Uncertainty Management, Second International Conference, SUM 2008, Naples, Italy, October 1-3, 2008. Proceedings, *Lecture Notes in Computer Science*, vol. 5291, pp. 146–159. Springer (2008)
19. Demir, E., Cary, M.P., Paley, S., Fukuda, K., Lemer, C., Vastrik, I., Wu, G., D'Eustachio, P., Schaefer, C., Luciano, J., et al.: The biopax community standard for pathway data sharing. Nature biotechnology **28**(9), 935–942 (2010)
20. Ding, Z., Peng, Y.: A probabilistic extension to ontology language OWL. In: 37th Hawaii International Conference on System Sciences (HICSS-37 2004), CD-ROM / Abstracts Proceedings, 5-8 January 2004, Big Island, HI, USA. IEEE Computer Society (2004)

21. Eén, N., Sörensson, N.: An extensible sat-solver. In: E. Giunchiglia, A. Tacchella (eds.) Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers, *Lecture Notes in Computer Science*, vol. 2919, pp. 502–518. Springer (2003)
22. Faizi, I.: A Description Logic Prover in Prolog, Bachelor's thesis, Informatics Mathematical Modelling, Technical University of Denmark (2011)
23. Gavanelli, M., Lamma, E., Riguzzi, F., Bellodi, E., Zese, R., Cota, G.: An abductive framework for datalog± ontologies. In: M.D. Vos, T. Eiter, Y. Lierler, F. Toni (eds.) Proceedings of the Technical Communications of the 31st International Conference on Logic Programming (ICLP 2015), Cork, Ireland, August 31 - September 4, 2015., *CEUR Workshop Proceedings*, vol. 1433. CEUR-WS.org (2015)
24. Gavanelli, M., Lamma, E., Riguzzi, F., Bellodi, E., Zese, R., Cota, G.: Abductive logic programming for datalog +/- ontologies. In: D. Ancona, M. Maratea, V. Mascardi (eds.) Proceedings of the 30th Italian Conference on Computational Logic, Genova, Italy, July 1-3, 2015., *CEUR Workshop Proceedings*, vol. 1459, pp. 128–143. CEUR-WS.org (2015)
25. Giugno, R., Lukasiewicz, T.: P-SHOQ(D): A probabilistic extension of SHOQ(D) for probabilistic ontologies in the semantic web. In: S. Flesca, S. Greco, N. Leone, G. Ianni (eds.) Logics in Artificial Intelligence, European Conference, JELIA 2002, Cosenza, Italy, September, 23-26, Proceedings, *Lecture Notes in Computer Science*, vol. 2424, pp. 86–97. Springer (2002)
26. Gottlob, G., Lukasiewicz, T., Simari, G.I.: Conjunctive query answering in probabilistic datalog+/- ontologies. In: S. Rudolph, C. Gutierrez (eds.) Web Reasoning and Rule Systems - 5th International Conference, RR 2011, Galway, Ireland, August 29-30, 2011. Proceedings, *Lecture Notes in Computer Science*, vol. 6902, pp. 77–92. Springer (2011)
27. Halaschek-Wiener, C., Kalyanpur, A., Parsia, B.: Extending tableau tracing for ABox updates. Tech. rep., University of Maryland (2006)
28. Halpern, J.Y.: An analysis of first-order logics of probability. Artif. Intell. **46**(3), 311–350 (1990)
29. Heinsohn, J.: Probabilistic description logics. In: R.L. de Mántaras, D. Poole (eds.) Conference on Uncertainty in Artificial Intelligence, pp. 311–318. Morgan Kaufmann (1994)
30. Herchenröder, T.: Lightweight semantic web oriented reasoning in Prolog: Tableaux inference for description logics. Master's thesis, School of Informatics, University of Edinburgh (2006)
31. Hitzler, P., Krötzsch, M., Rudolph, S.: Foundations of Semantic Web Technologies. CRCPress (2009)
32. Hustadt, U., Motik, B., Sattler, U.: Deciding expressive description logics in the framework of resolution. Inf. Comput. **206**(5), 579–601 (2008)
33. Jaeger, M.: Probabilistic reasoning in terminological logics. In: J. Doyle, E. Sandewall, P. Torasso (eds.) Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning (KR'94). Bonn, Germany, May 24-27, 1994., pp. 305–316. Morgan Kaufmann (1994)
34. Jung, J.C., Lutz, C.: Ontology-based access to probabilistic data with OWL QL. In: P. Cudré-Mauroux, J. Heflin, E. Sirin, T. Tudorache, J. Euzenat, M. Hauswirth, J.X. Parreira, J. Hendler, G. Schreiber, A. Bernstein, E. Blomqvist (eds.) The Semantic Web - ISWC 2012 - 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part I, *Lecture Notes in Computer Science (LNCS)*, vol. 7649, pp. 182–197. Springer, Berlin (2012)
35. Kalyanpur, A.: Debugging and repair of OWL ontologies. Ph.D. thesis, The Graduate School of the University of Maryland (2006)
36. Kalyanpur, A., Parsia, B., Horridge, M., Sirin, E.: Finding all justifications of OWL DL entailments. In: K. Aberer, et al. (eds.) ISWC/ASWC 2007, *LNCS*, vol. 4825, pp. 267–280. Springer (2007)
37. Klinov, P.: Pronto: A non-monotonic probabilistic description logic reasoner. In: S. Bechhofer, M. Hauswirth, J. Hoffmann, M. Koubarakis (eds.) The Semantic Web: Research and Applications, 5th European Semantic Web Conference, ESWC 2008, Tenerife, Canary Islands, Spain, June 1-5, 2008, Proceedings, *Lecture Notes in Computer Science*, vol. 5021, pp. 822–826. Springer (2008)

38. Klinov, P., Parsia, B.: Optimization and evaluation of reasoning in probabilistic description logic: Towards a systematic approach. In: A.P. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T.W. Finin, K. Thirunarayan (eds.) The Semantic Web - ISWC 2008, 7th International Semantic Web Conference, ISWC 2008, Karlsruhe, Germany, October 26-30, 2008. Proceedings, *Lecture Notes in Computer Science*, vol. 5318, pp. 213–228. Springer (2008)
39. Koller, D., Levy, A.Y., Pfeffer, A.: P-CLASSIC: A tractable probablistic description logic. In: B. Kuipers, B.L. Webber (eds.) Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97, IAAI 97, July 27-31, 1997, Providence, Rhode Island., pp. 390–397. AAAI Press / The MIT Press (1997)
40. Lager, T., Wielemaker, J.: Pengines: Web logic programming made easy. TPLP **14**(4-5), 539–552 (2014)
41. Laskey, K.B., da Costa, P.C.G.: Of starships and klingons: Bayesian logic for the 23rd century. In: UAI '05, Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence, Edinburgh, Scotland, July 26-29, 2005, pp. 346–353. AUAI Press (2005)
42. Lukácsy, G., Szeredi, P.: Efficient description logic reasoning in prolog: The dlog system. TPLP **9**(3), 343–414 (2009)
43. Lukasiewicz, T.: Probabilistic default reasoning with conditional constraints. Ann. Math. Artif. Int. **34**(1-3), 35–88 (2002)
44. Lukasiewicz, T.: Expressive probabilistic description logics. Artif. Int. **172**(6-7), 852–883 (2008)
45. Luna, J.E.O., Revoredo, K., Cozman, F.G.: Learning probabilistic Description Logics: A framework and algorithms. In: I.Z. Batyrshin, G. Sidorov (eds.) Advances in Artificial Intelligence - 10th Mexican International Conference on Artificial Intelligence, MICAI 2011, Puebla, Mexico, November 26 - December 4, 2011, Proceedings, Part I, *Lecture Notes in Computer Science*, vol. 7094, pp. 28–39. Springer, Berlin (2011)
46. Lutz, C., Schröder, L.: Probabilistic Description Logics for subjective uncertainty. In: F. Lin, U. Sattler, M. Truszczynski (eds.) Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference, KR 2010, Toronto, Ontario, Canada, May 9-13, 2010, pp. 393–403. AAAI Press, Menlo Park, CA, USA (2010)
47. Meissner, A.: An automated deduction system for description logic with alcn language. Studia z Automatyki i Informatyki **28-29**, 91–110 (2004)
48. Nagypál, G., Deswarte, R., Oosthoek, J.: Applying the semantic web: The VICODI experience in creating visual contextualization for history. Literary and Linguistic Computing **20**(3), 327–349 (2005)
49. Nilsson, N.J.: Probabilistic logic. Artif. Intell. **28**(1), 71–87 (1986)
50. Patel-Schneider P, F., Horrocks, I., Bechhofer, S.: Tutorial on OWL (2003)
51. Poole, D.: The Independent Choice Logic for modelling multiple agents under uncertainty. Artif. Intell. **94**(1-2), 7–56 (1997)
52. Poole, D.: Abducing through negation as failure: stable models within the independent choice logic. J. Log. Program. **44**(1-3), 5–35 (2000)
53. Reiter, R.: A theory of diagnosis from first principles. Artif. Intell. **32**(1), 57–95 (1987)
54. Ricca, F., Gallucci, L., Schindlauer, R., Dell'Armi, T., Grasso, G., Leone, N.: OntoDLV: An ASP-based system for enterprise ontologies. J. Log. Comput. **19**(4), 643–670 (2009)
55. Riguzzi, F., Bellodi, E., Lamma, E., Zese, R.: Epistemic and statistical probabilistic ontologies. In: F. Bobillo, et al. (eds.) URSW 2012, *CEUR Workshop Proceedings*, vol. 900, pp. 3–14. Sun SITE Central Europe (2012)
56. Riguzzi, F., Bellodi, E., Lamma, E., Zese, R.: BUNDLE: A reasoner for probabilistic ontologies. In: W. Faber, D. Lembo (eds.) Web Reasoning and Rule Systems - 7th International Conference, RR 2013, Mannheim, Germany, July 27-29, 2013. Proceedings, *Lecture Notes in Computer Science*, vol. 7994, pp. 183–197. Springer (2013)
57. Riguzzi, F., Bellodi, E., Lamma, E., Zese, R.: Parameter learning for probabilistic ontologies. In: W. Faber, D. Lembo (eds.) Web Reasoning and Rule Systems - 7th International Conference, RR 2013, Mannheim, Germany, July 27-29, 2013. Proceedings, *Lecture Notes in Computer Science*, vol. 7994, pp. 265–270. Springer (2013)

58. Riguzzi, F., Bellodi, E., Lamma, E., Zese, R.: Probabilistic description logics under the distribution semantics. Semantic Web - Interoperability, Usability, Applicability **6**(5), 447–501 (2015). DOI 10.3233/SW-140154
59. Riguzzi, F., Bellodi, E., Lamma, E., Zese, R.: Reasoning with probabilistic ontologies. In: Q. Yang, M. Wooldridge (eds.) IJCAI 2015, pp. 4310–4316. AAAI Press (2015)
60. Riguzzi, F., Bellodi, E., Lamma, E., Zese, R., Cota, G.: Learning probabilistic description logics. In: URSW III, *LNCS*, vol. 8816, pp. 63–78. Springer (2014)
61. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: ICLP 1995, pp. 715–729. MIT Press (1995)
62. Schlobach, S., Cornet, R.: Non-standard reasoning services for the debugging of description logic terminologies. In: G. Gottlob, T. Walsh (eds.) IJCAI 2003, pp. 355–362. Morgan Kaufmann (2003)
63. Sirin, E., Parsia, B., Cuenca-Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. J. Web Sem. **5**(2), 51–53 (2007)
64. Vassiliadis, V., Wielemaker, J., Mungall, C.: Processing OWL2 ontologies using thea: An application of logic programming. In: OWLED 2009, *CEUR Workshop Proceedings*, vol. 529. CEUR-WS.org (2009)