Efficient TBox Reasoning with Value Restrictions—Introducing the $\mathcal{FL}_o wer$ Reasoner

Friedrich Michel, Anni-Yasmin Turhan, and Benjamin Zarrieß

Institute for Theoretical Computer Science
Technische Universität Dresden
{friedrich.michel,anni-yasmin.turhan,benjamin.zarriess}@tu-dresden.de

Abstract. The Description Logic (DL) $\mathcal{FL}_0$ uses universal quantification, whereas its well-known counter-part $\mathcal{EL}$ uses the existential one. While for $\mathcal{EL}$ deciding subsumption in the presence of general TBoxes is tractable, this is no the case for $\mathcal{FL}_0$. We present a novel algorithm for solving the ExpTime-hard subsumption problem in $\mathcal{FL}_0$ w.r.t. general TBoxes, which is based on the computation of so-called *least functional models*. To build a such a model our algorithm treats TBox axioms as rules that are applied to objects of the interpretation domain. This algorithm is implemented in the $\mathcal{FL}_o wer$ reasoner, which uses a variant of the Rete pattern matching algorithm to find applicable rules. We present an evaluation of $\mathcal{FL}_o wer$ on a large set of TBoxes generated from real world ontologies. The experimental results indicate that our prototype implementation of the specialised technique for $\mathcal{FL}_0$ leads in most cases to a huge performance gain in comparison to the highly-optimised tableau reasoners.

1 Introduction

The Description Logic (DL) $\mathcal{FL}_0$ is a minimalistic DL, since it offers only the top concept, conjunction, and value restriction (universal quantification) as constructors for building complex concepts. It is the core part of one of the very first DLs called $\mathcal{FL}^-$ ($\mathcal{F}$rame $\mathcal{L}$anguage) introduced by Brachman and Levesque [8] for formalising frames. Unfortunately, in presence of a TBox value restrictions and conjunction have been identified as exactly those constructors that make the problem of deciding the subsumption relationship between two concepts hard [15, 1]. In particular, depending on the syntactical form of the TBox the complexity of deciding subsumption in $\mathcal{FL}_0$ takes a rollercoaster ride: it starts from Ptime with the empty TBox [8], jumps to co-NP-completeness with acyclic TBoxes [15], then to Pspace-completeness with cyclic definitions [12], culminates in ExpTime-completeness in presence of general TBoxes [1], and drops back to Ptime when restricted to Horn-TBoxes [13]. This is in sharp contrast to the robust behaviour of the popular DL $\mathcal{EL}$ that differs from $\mathcal{FL}_0$ only by using existential restrictions instead of value restrictions. In $\mathcal{EL}$, the complexity stays in Ptime even in the presence of general TBoxes [1].

In this paper we devise a novel algorithm for deciding subsumption w.r.t. general $\mathcal{FL}_0$-TBoxes, describe a first implementation of it in the new $\mathcal{FL}_o wer$ reasoner and report on an evaluation of $\mathcal{FL}_o wer$ on a large collection of ontologies. There are several reasons to study subsumption algorithms for $\mathcal{FL}_0$. First, for the Boolean-complete DL $\mathcal{ALC}$ subsumption w.r.t. TBoxes is still ExpTime-complete, thus its fragment $\mathcal{FL}_0$ hardly offers an optimal trade-off between worst-case complexity and expressiveness. Nevertheless, the volatile complexity of subsumption in $\mathcal{FL}_0$ raises the question of whether hard instances of the subsumption problem are even likely to occur in application ontologies.

Second, most state-of-the-art ontology reasoners are built and optimised for expressive DLs beyond $\mathcal{ALC}$, as for example FaCT++[1], HermiT[2] or Konclude[3]. Some DL reasoners such as Konclude and MORe [17] make use of specialised algorithms for certain language fragments as part of their overall reasoning algorithm. Thus an efficient subsumption algorithm for $\mathcal{FL}_0$ might be such a dedicated procedure that can augment general ontology reasoners.

Third, dedicated methods for standard and non-standard reasoning tasks in $\mathcal{FL}_0$ w.r.t. general TBoxes have been studied recently in [5, 3, 4, 6]. Quite some of these useful inferences rely on subsumption tests as sub-procedures and $\mathcal{FL}_o wer$ can supply a base for implementing these inferences.

$\mathcal{FL}_o wer$'s subsumption algorithm for $\mathcal{FL}_0$ uses a characterisation of subsumption based on so-called tree-shaped *least functional models* [4]. For a given input concept and TBox, this algorithm generates a sufficiently large subtree of their least functional model by using the axioms from the TBox like rules to augment the tree. This process corresponds to deriving implicit consequences. To ensure termination it employs a blocking mechanism. We have implemented this algorithm in the new $\mathcal{FL}_o wer$ reasoner.[4]

The key idea of the implementation is to use a variant of the well-known Rete algorithm for rule application [9] adapted to the case without negation: we translate the TBox to a Rete network and generate the tree representing the model of the TBox by propagating the nodes through the network. More precisely, the axioms in the TBox are applied as rules to nodes in the tree. For example, consider the following $\mathcal{FL}_0$ axiom:

$$\mathsf{Animal} \sqcap \forall \mathsf{eats}.\mathsf{Plants} \sqsubseteq \mathsf{Herbivore},$$

which essentially says that animals that eat only plants are herbivores. If a node in the current tree matches the left-hand side, which means it is labelled with the name Animal and its eats-child with Plants, then we add Herbivore to its label set. Since there are potentially many nodes and many axioms to consider, it is critical for performance reasons to avoid reiterating over all nodes and axioms after each change and so the key idea from the Rete algorithm is well-suited for our task. To support this claim we have conducted an evaluation on $\mathcal{FL}_o wer$. To

[1] `owl.man.ac.uk/faktplusplus`
[2] `hermit-reasoner.com`
[3] `konclude.com`
[4] `https://github.com/attalos/fl0wer`

create a large set of challenging $\mathcal{FL}_0$ ontologies we have transformed the OWL 2 EL ontologies of the OWL reasoner competition [16] into $\mathcal{FL}_0$ by replacing existential restrictions by value restrictions. It turns out that our reasoner $\mathcal{FL}_o wer$ is in many cases able to clearly outperform highly-optimised (hyper)tableaux reasoners like HermiT or JFact on reasoning tasks for this set of ontologies.

2 Preliminaries on $\mathcal{FL}_0$ and Least Functional Models

We define the DL $\mathcal{FL}_0$ and recall the characterisation of subsumption from [4] based on least functional models.

Syntax. Let $\mathsf{N_C}$ and $\mathsf{N_R}$ be disjoint *finite* sets of *concept names* and *role names*, respectively. An $\mathcal{FL}_0$*-concept description* (*concept* for short) C is built according to the following syntax rule

$$C ::= A \mid \top \mid C \sqcap C \mid \forall r.C, \quad \text{where } A \in \mathsf{N_C}, r \in \mathsf{N_R}.$$

$\top$ is called the *top concept* and $\forall r.C$ is a *value restriction*. For nested value restrictions we use the following notation: let $\sigma = r_1 \cdots r_m \in \mathsf{N_R}^*$ for some $m \geq 0$ be a word over the alphabet $\mathsf{N_R}$ of role names. For some $A \in \mathsf{N_C}$ we write $\forall \sigma.A$ as an abbreviation of $\forall r_1. \cdots \forall r_m.A$. The empty word ϵ stands for A.

A *general concept inclusion* (GCI) is of the form $C \sqsubseteq D$, where C and D are concepts. A *TBox* is a finite set of GCIs.

Semantics. An interpretation $\mathcal{I}$ is a pair $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$, consisting of a non-empty set $\Delta^\mathcal{I}$ (*domain* of $\mathcal{I}$) and an *interpretation function* $\cdot^\mathcal{I}$ that maps every concept name $A \in \mathsf{N_C}$ to a subset of the domain $A^\mathcal{I} \subseteq \Delta^\mathcal{I}$ and every role name $r \in \mathsf{N_R}$ to a binary relation $r^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$. The interpretation function is extended to (complex) concepts as follows:

$$(C \sqcap D)^\mathcal{I} := C^\mathcal{I} \sqcap D^\mathcal{I}, \qquad (\top)^\mathcal{I} := \Delta^\mathcal{I}, \text{ and}$$
$$(\forall r.C)^\mathcal{I} := \{d \in \Delta^\mathcal{I} \mid \forall e \in \Delta^\mathcal{I}.(d,e) \in r^\mathcal{I} \implies e \in C^\mathcal{I}\}$$

A GCI $C \sqsubseteq D$ is *satisfied* in $\mathcal{I}$, denoted by $\mathcal{I} \models C \sqsubseteq D$, iff $C^\mathcal{I} \subseteq D^\mathcal{I}$. $\mathcal{I}$ is a *model* of a TBox $\mathcal{T}$, denoted by $\mathcal{I} \models \mathcal{T}$, iff $\mathcal{I}$ satisfies all GCIs in $\mathcal{T}$. The concept C is *subsumed by the concept D w.r.t. a TBox $\mathcal{T}$*, denoted by $C \sqsubseteq_\mathcal{T} D$, iff $C^\mathcal{I} \subseteq D^\mathcal{I}$ is satisfied in all models of $\mathcal{T}$.

Our focus is on a certain kind of tree-shaped interpretation. A *functional interpretation* is a tree with domain $\mathsf{N_R}^*$, where each element has exactly one child node for each role name.

Definition 1. *An interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ is called a* functional interpretation *iff $\Delta^\mathcal{I} = \mathsf{N_R}^*$ and $r^\mathcal{I} = \{(\sigma, \sigma r) \mid \sigma \in \mathsf{N_R}^*\}$ for all $r \in \mathsf{N_R}$. A functional interpretation $\mathcal{I}$ is called* functional model *of a concept C w.r.t. TBox $\mathcal{T}$ iff $\mathcal{I} \models \mathcal{T}$ and $\epsilon \in C^\mathcal{I}$. For two functional interpretations $\mathcal{I}$ and $\mathcal{J}$ we write*

$$\mathcal{I} \subseteq \mathcal{J} \quad \textit{iff} \quad A^\mathcal{I} \subseteq A^\mathcal{J} \textit{ for all } A \in \mathsf{N_C}.$$

The notion of a functional interpretation fixes the domain and the interpretation of role names. Thus, a functional interpretation is uniquely determined by the interpretation of concept names.

Definition 2. *Let C be a concept and $\mathcal{T}$ be a TBox. The interpretation $\mathcal{I}_{C,\mathcal{T}} = (\mathsf{N_R}^*, \cdot^{\mathcal{I}_{C,\mathcal{T}}})$ is the functional interpretation satisfying*

$$A^{\mathcal{I}_{C,\mathcal{T}}} = \{\sigma \in \mathsf{N_R}^* \mid C \sqsubseteq_{\mathcal{T}} \forall\sigma.A\} \text{ for all } A \in \mathsf{N_C}.$$

We summarize some properties of $\mathcal{I}_{C,\mathcal{T}}$ which characterize $\mathcal{I}_{C,\mathcal{T}}$ as the *least* functional model of C w.r.t. $\mathcal{T}$.

Theorem 1 ([4, 2]). *It holds that $\mathcal{I}_{C,\mathcal{T}}$ is a functional model of C w.r.t. $\mathcal{T}$. Furthermore, if $\mathcal{J}$ is a functional model of C w.r.t. $\mathcal{T}$, then $\mathcal{I}_{C,\mathcal{T}} \subseteq \mathcal{J}$.*

Subsumption can now be characterised as inclusion between least functional models. It follows that $C \sqsubseteq_{\mathcal{T}} D$ iff $\mathcal{I}_{C,\mathcal{T}} \subseteq \mathcal{I}_{D,\mathcal{T}}$.

3 Subsumption Algorithm for $\mathcal{FL}_0$ with General TBoxes

We define a decision procedure for subsumption of two concepts w.r.t. a TBox based on a finite representation of the least functional model obtained by "applying" GCIs. The algorithm expects input in a certain normal form. A concept is in *normal form* if it is either $\top$ or a conjunction of concept names and value restrictions of the form $\forall r.A$ with $r \in \mathsf{N_R}$ and $A \in \mathsf{N_C}$. A GCI $C \sqsubseteq D$ is in normal form if both C and D are in normal form and a TBox $\mathcal{T}$ is in normal form if all GCIs in $\mathcal{T}$ are in normal form. By using the equivalence $\forall r.(C \sqcap D) \equiv \forall r.C \sqcap \forall r.D$ and via "flattening" of nested value restrictions by introducing fresh concept names every TBox can be transformed in linear time into normal form.

In the remainder of this section $\mathcal{T}$ denotes a TBox in normal form. Given two concept names A and B mentioned in $\mathcal{T}$ we want to decide whether $A \sqsubseteq_{\mathcal{T}} B$ holds. The algorithm computes a finite subtree of the tree $\mathcal{I}_{A,\mathcal{T}}$ such that one can read off the named subsumers (concept names) of A w.r.t. $\mathcal{T}$ at the root. The structure that the algorithm operates on is a *partial functional interpretation*. It is the same as a functional interpretation but the domain is only a *finite* prefix-closed subset of $\mathsf{N_R}^*$, that is, a finite tree.

Definition 3. *The interpretation $\mathcal{Y} = (\Delta^{\mathcal{Y}}, \cdot^{\mathcal{Y}})$ is a* partial functional interpretation *iff $\Delta^{\mathcal{Y}} \subset \mathsf{N_R}^*$ is a* finite *prefix-closed set and for all $r \in \mathsf{N_R}$ it holds that $r^{\mathcal{Y}} = \{(\sigma, \sigma r) \mid (\sigma, \sigma r) \in \Delta^{\mathcal{Y}} \times \Delta^{\mathcal{Y}}\}$.*

The algorithm for deciding $A \sqsubseteq_{\mathcal{T}} B$ generally proceeds as follows: it starts with a partial functional interpretation $\mathcal{Y}$ that just consists of

$$\Delta^{\mathcal{Y}} := \{\epsilon\} \qquad A^{\mathcal{Y}} := \{\epsilon\} \qquad B^{\mathcal{Y}} := \emptyset \text{ for all } B \in \mathsf{N_C} \setminus \{A\}.$$

In each iteration a single element of the current tree and a single GCI from $\mathcal{T}$ is chosen such that the chosen element matches the left-hand side, but not the

right-hand side of the GCI. Then minimal extensions of the tree are performed such that the element now matches also the right-hand side. The extension affects the domain and/or the interpretation of concept names. To ensure soundness we guarantee that for the tree $\mathcal{Y}$, the invariant $\mathcal{Y} \subseteq \mathcal{I}_{A,\mathcal{T}}$ is always satisfied. To ensure termination we distinguish blocked and non-blocked elements in the tree. The algorithm terminates if for all non-blocked elements and all GCIs, a match of the left-hand side implies a match of the right-hand side. For describing the procedure we have to define 1. what it means that a domain element of a partial interpretation matches a concept, 2. how to extend the tree to achieve a match with the right-hand side, and 3. how to distinguish blocked and non-blocked elements. For the first step we introduce the following auxiliary notions.

Definition 4. *Let $\mathcal{Y} = (\Delta^{\mathcal{Y}}, \cdot^{\mathcal{Y}})$ be a partial functional interpretation and D a concept in normal form. The set of elements in $\Delta^{\mathcal{Y}}$ that* match *D, denoted by $\mathsf{match}(D, \mathcal{Y})$, is defined inductively as follows:*

$$\begin{aligned}
\mathsf{match}(\top, \mathcal{Y}) &:= \Delta^{\mathcal{Y}};\\
\mathsf{match}(A, \mathcal{Y}) &:= \{w \in \Delta^{\mathcal{Y}} \mid w \in A^{\mathcal{Y}}\} \text{ for all } A \in \mathsf{N_C};\\
\mathsf{match}(\forall r.A, \mathcal{Y}) &:= \{\sigma \in \Delta^{\mathcal{Y}} \mid \text{ there exists } \sigma r \in \Delta^{\mathcal{Y}} \text{ and } \sigma r \in A^{\mathcal{Y}}\}\\
&\qquad \text{for all } r \in \mathsf{N_R} \text{ and } A \in \mathsf{N_C};\\
\mathsf{match}(C_1 \sqcap C_2, \mathcal{Y}) &:= \mathsf{match}(C_1, \mathcal{Y}) \cap \mathsf{match}(C_2, \mathcal{Y}).
\end{aligned}$$

Since $\mathcal{Y}$ is partial functional (i.e. has *at most* one child per node for each role name), it is easy to see that $\sigma \in \mathsf{match}(C, \mathcal{Y})$ implies $\sigma \in C^{\mathcal{Y}}$. The converse need not be true, as σ may have no r-child in $\Delta^{\mathcal{Y}}$. We define $\sigma \in \Delta^{\mathcal{Y}}$ *violates* $C \sqsubseteq D$ (in normal form) iff $\sigma \in \mathsf{match}(C, \mathcal{Y})$ and $\sigma \notin \mathsf{match}(D, \mathcal{Y})$. Given a TBox $\mathcal{T}$ in normal form and a partial functional interpretation $\mathcal{Y}$ we define the *set of all incomplete elements* as follows

$$\mathsf{ic}(\mathcal{Y}, \mathcal{T}) := \{\sigma \in \Delta^{\mathcal{Y}} \mid \text{ there is } E \sqsubseteq F \in \mathcal{T} \text{ such that } \sigma \text{ violates } E \sqsubseteq F\}.$$

Intuitively, the elements in $\mathsf{ic}(\mathcal{Y}, \mathcal{T})$ are those eligible for an extension of $\mathcal{Y}$ towards building a representation of the least functional model, while those in $\Delta^{\mathcal{Y}} \setminus \mathsf{ic}(\mathcal{Y}, \mathcal{T})$ are not. As an additional filter for extensions we define a blocking condition. First, we introduce the auxiliary notions for the blocking mechanism consisting of the standard notion of a prefix and proper prefix and a strict total order on $(\mathsf{N_R})^*$.

Definition 5. *Let $\sigma, \rho \in {\mathsf{N_R}}^*$. We write $\rho \in \mathsf{prefix}(\sigma)$ iff $\sigma = \rho\widehat{\sigma}$ for some $\widehat{\sigma} \in {\mathsf{N_R}}^*$. We write $\rho \in \mathsf{pprefix}(\sigma)$ iff $\rho \in \mathsf{prefix}(\sigma)$ and $\rho \neq \sigma$ to denote a* proper prefix *of σ. Let $\mathsf{N_R} = \{r_1, \ldots, r_n\}$, $\sigma = r_{i_1} \cdots r_{i_k} \in {\mathsf{N_R}}^*$ and $\rho = r_{j_1} \cdots r_{j_\ell} \in {\mathsf{N_R}}^*$ be two words with $i_1, \ldots, i_k, j_1, \ldots, j_\ell \in \{1, \ldots, n\}$. Let $\prec_{\mathbb{N}}$ denote the lexicographic order over tuples of natural numbers. We define*

$$\sigma \prec \rho \;\text{ iff }\; |\sigma| < |\rho| \text{ or if } k = \ell, \text{ then } (i_1, \ldots, i_k) \prec_{\mathbb{N}} (j_1, \ldots, j_\ell).$$

The blocking condition is defined by induction on $\prec$.

Definition 6. *Let $\mathcal{Y} = (\Delta^{\mathcal{Y}}, \cdot^{\mathcal{Y}})$ be a partial functional interpretation and let $\sigma \in \Delta^{\mathcal{Y}}$. By $\mathcal{Y}(\sigma) := \{A \in \mathsf{N_C} \mid \sigma \in A^{\mathcal{Y}}\}$ we denote the* label *of σ in $\mathcal{Y}$. The* set of all blocked elements *in $\Delta^{\mathcal{Y}}$ is defined as the smallest set satisfying all of the following conditions:*

1. *ϵ is not blocked.*
2. *Let $\sigma \in \Delta^{\mathcal{Y}}$ and assume for all $\sigma' \in \Delta^{\mathcal{Y}}$ with $\sigma' \prec \sigma$ the blocking status is already defined. The following conditions apply to σ:*
   (a) *If there exists $\omega \in \Delta^{\mathcal{Y}}$ with $\omega \prec \sigma$ such that $\mathcal{Y}(\sigma) = \mathcal{Y}(\omega)$ and ω is not blocked, then σ is blocked.*
   (b) *If there exists $\rho \in \mathsf{pprefix}(\sigma)$ such that ρ is blocked, then σ is blocked.*

The set of all non-blocked *elements in $\mathcal{Y}$ is denoted by $\mathsf{nb}(\mathcal{Y})$.*

Next, we define what an extension step is. Such a step expands a single non-blocked and incomplete element in a partial functional interpretation.

Definition 7. *Let $\mathcal{Y}$ and $\mathcal{Z}$ be two partial functional interpretations, $\mathcal{T}$ a TBox in normal form, $m, n \geq 0$*

$$\alpha = C \sqsubseteq (A_1 \sqcap \cdots \sqcap A_m \sqcap \forall r_1.B_1 \sqcap \cdots \forall r_n.B_n) \text{ and } \alpha \in \mathcal{T}$$

a GCI and $\sigma \in \mathsf{nb}(\mathcal{Y}) \cap \mathsf{ic}(\mathcal{Y}, \mathcal{T})$ a non-blocked, incomplete element in $\mathcal{Y}$ violating α. Then $\mathcal{Y}$ expands *α at σ to $\mathcal{Z}$, denoted $\mathcal{Y} \vdash^{\sigma}_{\alpha} \mathcal{Z}$ iff $\mathcal{Z}$ satisfies the conditions*

- $\Delta^{\mathcal{Z}} = \Delta^{\mathcal{Y}} \cup \{\sigma r_1, \ldots, \sigma r_n\}$;
- $A_i^{\mathcal{Z}} = A_i^{\mathcal{Y}} \cup \{\sigma\}$ *for all* $i = 1, \ldots, m$;
- $B_j^{\mathcal{Z}} = B_j^{\mathcal{Y}} \cup \{\sigma r_j\}$ *for all* $j = 1, \ldots, n$; *and*
- $Q^{\mathcal{Z}} = Q^{\mathcal{Y}}$ *for all* $Q \in \mathsf{N_C} \setminus (\{A_1, \ldots A_m, B_1, \ldots, B_n\})$.

A partial functional interpretation $\mathcal{Z}'$ is a $\mathcal{T}$-completion *of $\mathcal{Y}$, written as $\mathcal{Y} \vdash_{\mathcal{T}} \mathcal{Z}'$, iff there exists $\alpha' \in \mathcal{T}$ and $\sigma' \in \mathsf{nb}(\mathcal{Y}) \cap \mathsf{ic}(\mathcal{Y}, \mathcal{T})$ such that $\mathcal{Y} \vdash^{\sigma'}_{\alpha'} \mathcal{Z}'$ holds.*

For a given $\sigma \in \mathsf{nb}(\mathcal{Y}) \cap \mathsf{ic}(\mathcal{Y}, \mathcal{T})$ violating a GCI $\alpha \in \mathcal{T}$ there exists a unique $\mathcal{Z}$ with $\mathcal{Y} \vdash^{\sigma}_{\alpha} \mathcal{Z}$. The extension of $\mathcal{Y}$ leading to $\mathcal{Z}$ is minimal such that now σ matches the right-hand side of α in $\mathcal{Z}$. Depending on the choice of σ and the GCI there can be several $\mathcal{T}$-completions of $\mathcal{Y}$. Furthermore, it is guaranteed that either $\mathsf{nb}(\mathcal{Y}) \cap \mathsf{ic}(\mathcal{Y}, \mathcal{T}) = \emptyset$ or there exists a $\mathcal{T}$-completion of $\mathcal{Y}$.

Given the input $A, B \in \mathsf{N_C}$ and $\mathcal{T}$, the *algorithm* $\textsc{Subs}(A, B, \mathcal{T})$ for checking whether $A \sqsubseteq_{\mathcal{T}} B$ holds, computes a sequence of $\mathcal{T}$-completions until it reaches a partial functional interpretation where no non-blocked element violates any GCI. The algorithm starts with the following partial functional interpretation:

$$\Delta^{\mathcal{Y}_0} := \{\epsilon\}; \quad A^{\mathcal{Y}_0} := \{\epsilon\} \quad \text{and} \quad B^{\mathcal{Y}_0} := \emptyset \text{ for all } B \in \mathsf{N_C} \setminus \{A\}, \tag{1}$$

and computes a sequence

$$\mathcal{Y}_0 \vdash_{\mathcal{T}} \mathcal{Y}_1 \vdash_{\mathcal{T}} \cdots \mathcal{Y}_{(n-1)} \vdash_{\mathcal{T}} \mathcal{Y}_n$$

such that $\mathcal{Y}_n$ is complete in the sense that only blocked elements can remain incomplete, i.e. $\mathsf{nb}(\mathcal{Y}_n) \cap \mathsf{ic}(\mathcal{Y}_n, \mathcal{T}) = \emptyset$. It answers "yes" if $B \in \mathcal{Y}_n(\epsilon)$ (or equivalently $\epsilon \in B^{\mathcal{Y}_n}$) and "no" otherwise.

The choice of the next $\mathcal{T}$-completion is a *don't care nondeterministic choice*. For proving *soundness* it is sufficient to show that $\mathcal{Y}_i \subseteq \mathcal{I}_{A,\mathcal{T}}$ holds for each partial functional interpretation reachable from $\mathcal{Y}_0$ via $\vdash_{\mathcal{T}}$. By $\vdash_{\mathcal{T}}{}^*$ we denote the reflexive transitive closure of $\vdash_{\mathcal{T}}$.

Lemma 1. *Let $\mathcal{Y}_0$ be as in (1), $\mathcal{I}_{A,\mathcal{T}}$ the least functional model of A w.r.t. $\mathcal{T}$ and $\mathcal{Z}$ a partial functional interpretation satisfying $\mathcal{Y}_0 \vdash_{\mathcal{T}}{}^* \mathcal{Z}$. Then $\mathcal{Z} \subseteq \mathcal{I}_{A,\mathcal{T}}$.*

Proof (sketch). The proof is straightforward by induction on the length of the completion sequence.

As a consequence of this we get soundness of the overall procedure.

Lemma 2. *$Subs(A, B, \mathcal{T})$ is sound.*

Proof. If the answer is "yes", then $Subs(A, B, \mathcal{T})$ has obtained $\mathcal{Y}_n$ with $\mathcal{Y}_0 \vdash_{\mathcal{T}}{}^* \mathcal{Y}_n$ and $\epsilon \in B^{\mathcal{Y}_n}$. Since $\mathcal{Y}_n \subseteq \mathcal{I}_{A,\mathcal{T}}$, it follows that $\epsilon \in B^{\mathcal{I}_{A,\mathcal{T}}}$. From the definition of $\mathcal{I}_{A,\mathcal{T}}$ it follows that $A \sqsubseteq_{\mathcal{T}} B$ holds.

We prove that the algorithm terminates, which means that it always reaches a partial functional interpretation $\mathcal{Z}$, where the set $\mathsf{nb}(\mathcal{Z}, \mathcal{T}) \cap \mathsf{ic}(\mathcal{Z}, \mathcal{T})$ is empty. Using the blocking condition, the following lemma about the length of the elements in the tree follows immediately. The *length of an element* $\sigma \in \mathsf{N_R}^*$ is denoted by $|\sigma|$. We have $|\epsilon| = 0$ and $|\sigma' r| = |\sigma'| + 1$ with $r \in \mathsf{N_R}$.

Lemma 3. *Let $\mathcal{Z}$ be a partial functional interpretation such that $\mathcal{Y}_0 \vdash_{\mathcal{T}}{}^* \mathcal{Z}$. Then the set $\mathsf{nb}(\mathcal{Z})$ is prefix-closed and $\sigma \in \mathsf{nb}(\mathcal{Z})$ implies that $|\sigma| \leq 2^{|(\mathsf{N_C})|}$.*

Lemma 3 yields an upper bound on the depth of the trees that are the result of a sequence of $\mathcal{T}$-completions starting in the initial $\mathcal{Y}_0$ consisting only of ϵ labelled with A. The *depth* of a partial functional interpretation $\mathcal{Y} = (\Delta^{\mathcal{Y}}, \cdot^{\mathcal{Y}})$, denoted by $\mathsf{depth}(\mathcal{Y})$, is the maximum length defined by

$$\mathsf{depth}(\mathcal{Y}) := \mathsf{max}(\{|\sigma| \mid \sigma \in \Delta^{\mathcal{Y}}\}).$$

Lemma 4. *Let $\mathcal{Z}$ be a partial functional interpretation such that $\mathcal{Y}_0 \vdash_{\mathcal{T}}{}^* \mathcal{Z}$. It holds that*

$$\mathsf{depth}(\mathcal{Z}) \leq 2^{|(\mathsf{N_C})|} + 1.$$

The upper bound on the depth of the tree in a $\mathcal{T}$-completion sequence also yields an upper bound on its overall size. Furthermore, we observe that $\mathcal{Y} \vdash_{\mathcal{T}} \mathcal{Y}'$ implies that $\mathcal{Y} \subsetneq \mathcal{Y}'$, i.e. a $\mathcal{T}$-completion always adds something. At the same time, each label set can at most contain $\left|\mathsf{N_C}\right|$ many names. Thus, due to the depth bound and the upper bound on the label size there cannot be an infinite sequence of $\mathcal{T}$-completions. Hence, $Subs(A, B, \mathcal{T})$ always terminates.

Lemma 5. *$Subs(A, B, \mathcal{T})$ always terminates.*

After $Subs(A, B, \mathcal{T})$ terminates, it obtains a partial functional interpretation $\mathcal{Z}$ with $\mathsf{nb}(\mathcal{Z}) \cap \mathsf{ic}(\mathcal{Z}, \mathcal{T}) = \emptyset$. For the completeness proof we construct a model of $\mathcal{T}$ and A from $\mathcal{Z}$. In particular, the construction is based on the non-blocked elements in $\mathsf{nb}(\mathcal{Z})$. It might be the case that a non-blocked element has children (role successors) that are blocked. To handle this situation the following lemma is helpful. Recall that for an element $\sigma \in \Delta^{\mathcal{Z}}$ the *label set* $\mathcal{Z}(\sigma)$ is the set of all concept names that σ satisfies in $\mathcal{Z}$ (see Def. 6).

Lemma 6. *Let $\mathcal{Z}$ be a partial functional interpretation such that $\mathcal{Y}_0 \vdash_{\mathcal{T}}^* \mathcal{Z}$ and $\mathsf{nb}(\mathcal{Z}) \cap \mathsf{ic}(\mathcal{Z}, \mathcal{T}) = \emptyset$. For all $\sigma \in \mathsf{nb}(\mathcal{Z})$ and all $r \in \mathsf{N_R}$ it holds that*

$$\text{if } \sigma r \in \Delta^{\mathcal{Z}} \text{ then there exists } \rho \in \mathsf{nb}(\mathcal{Z}) \text{ such that } \mathcal{Z}(\sigma r) = \mathcal{Z}(\rho).$$

The construction of a model of $\mathcal{T}$ and A from the final partial functional interpretation $\mathcal{Z}$ obtained from the run of the algorithm is based on the label sets of the non-blocked elements. The lemma above guarantees that the label sets of blocked children of non-blocked elements can be found in $\mathsf{nb}(\mathcal{Z})$.

Definition 8. *Let $\mathcal{Z}$ be a partial functional interpretation such that $\mathcal{Y}_0 \vdash_{\mathcal{T}}^* \mathcal{Z}$ and $\mathsf{nb}(\mathcal{Z}) \cap \mathsf{ic}(\mathcal{Z}, \mathcal{T}) = \emptyset$. We define an interpretation $\mathsf{m}(\mathcal{Z})$ as follows:*

- $\Delta^{\mathsf{m}(\mathcal{Z})} := \{\mathcal{Z}(\sigma) \mid \sigma \in \mathsf{nb}(\mathcal{Z})\}$;
- $Q^{\mathsf{m}(\mathcal{Y})} := \{X \in \Delta^{\mathsf{m}(\mathcal{Y})} \mid Q \in X\}$ *for all* $Q \in \mathsf{N_C}$;
- $r^{\mathsf{m}(\mathcal{Y})} := \{(\mathcal{Z}(\sigma), \mathcal{Z}(\sigma r)) \mid \sigma \in \mathsf{nb}(\mathcal{Z}), \sigma r \in \Delta^{\mathcal{Z}}\}$ *for all* $r \in \mathsf{N_R}$.

Lemma 6 ensures that the interpretation of role names is well-defined in $\mathsf{m}(\mathcal{Z})$. It easy to show that $\mathsf{m}(\mathcal{Z})$ is a model of $\mathcal{T}$ and A. As a consequence we get completeness of $Subs(A, B, \mathcal{T})$.

Theorem 2. *$Subs(A, B, \mathcal{T})$ is sound, complete and terminating.*

The algorithm $Subs(A, B, \mathcal{T})$ shares properties with the completion method for $\mathcal{EL}$ [1] as well as with tableau algorithms for expressive DLs [7]. Every single $\mathcal{T}$-completion step extends the label set of at least one node in the tree. Intuitively, adding the concept name C to the label set of domain element σ corresponds to deriving $A \sqsubseteq \forall\sigma.C$ as a consequence of $\mathcal{T}$. One single run of $Subs(A, B, \mathcal{T})$ not only decides whether $A \sqsubseteq B$ is entailed by $\mathcal{T}$ but computes *all* subsumers of A. This is similar to the $\mathcal{EL}$ completion method and other consequence-based calculi [18]. From tableau algorithms $Subs(A, B, \mathcal{T})$ inherits the blocking mechanism that ensures termination.

4 A Rete-based Matching Algorithm for $\mathcal{FL}_0$-Concepts

Our implementation of $Subs(A, B, \mathcal{T})$ in $\mathcal{FL}_o wer$ employs a variant of the Rete algorithm [9] to obtain candidates for each $\mathcal{T}$-completion step. While the full Rete algorithm admits also negation, we only need the positive part. In this section, we informally describe how this is realised—for full details see [14].

In order to compute a sequence of $\mathcal{T}$-completions $\mathcal{Y}_0 \vdash_\mathcal{T} \mathcal{Y}_1 \vdash_\mathcal{T} \mathcal{Y}_2 \vdash_\mathcal{T} \cdots$ starting from the tree $\mathcal{Y}_0$, one has to compute in each completion step i the pairs $(\sigma, C \sqsubseteq D) \in \left(\Delta^{\mathcal{Y}_i} \cap \mathsf{nb}(\mathcal{Y}_i)\right) \times \mathcal{T}$ such that σ matches C but not D in $\mathcal{Y}_i$. Thus, we can view a GCI $C \sqsubseteq D \in \mathcal{T}$ as a rule of the form

$$?\sigma \in \mathsf{match}(C, \mathcal{Y}_i) \rightarrow \ ?\sigma \in \mathsf{match}(D, \mathcal{Y}_i),$$

where $?\sigma$ ranges over the non-blocked domain elements of $\mathcal{Y}_i$. Since there is potentially a large number of elements in $\Delta^{\mathcal{Y}_i}$ that has to be matched with a large number of left-hand sides of GCIs (patterns) in the TBox in each step, we have chosen to implement this task using the Rete matching algorithm [9]. In each completion step the extension of the tree only affects a small number of elements: the matching element σ itself and/or its children. This makes the Rete-based algorithm particularly efficient in our setting because it stores matching information across completion steps to avoid reiterating over the whole set of pairs $\left(\Delta^{\mathcal{Y}_i} \cap \mathsf{nb}(\mathcal{Y}_i)\right) \times \mathcal{T}$ in each step. Only the elements with changes have to be rematched again in each step.

As a preprocessing phase $\mathcal{FL}_o wer$ compiles the TBox $\mathcal{T}$ in normal form into a *Rete network*. In general, the network consists of three kinds of nodes: a single root node, intermediate nodes and terminal nodes. A *terminal node* holds the right-hand side of a GCI that is ready to be applied to an element. The matching is done by passing so called tokens from the root node through the intermediate nodes to the terminal nodes. A *token* is a pair of the form $(\sigma, s) \in (\mathsf{N_R}^*, \mathsf{N_R} \cup \{\epsilon\})$. Intuitively, the token (σ, ϵ) is used to check whether σ matches the concept names at the top-level of a concept and a token of the form (σ, r) with $r \in \mathsf{N_R}$ is used to check whether σ matches value restrictions with role name r.

There are three types of *intermediate nodes* that process tokens arriving from predecessor nodes in the network:

- A *concept node* is labelled with a concept name $B \in \mathsf{N_C}$. It sends a token (σ, s) to all successor nodes iff $\sigma s \in B^{\mathcal{Y}_i}$.
- A *role node* is labelled with an $s \in \mathsf{N_R} \cup \{\epsilon\}$. An arriving token of the form (σ, s') is handled as follows. If $s \in \mathsf{N_R}$, then it sends (σ, s') to all successor nodes iff $s' = s$ and if $s = \epsilon$, then it sends the token $(\sigma s', \epsilon)$ to all successor nodes.
- An *inter-element node* is labelled with a tuple $(s_1, \ldots, s_m) \in (\mathsf{N_R} \cup \{\epsilon\})^m$. It stores all arriving tokens and sends a token (σ, ϵ) to its successor nodes once *all* tokens of the form $(\sigma, s_1), \ldots, (\sigma, s_m)$ have arrived at this node.

The overall network is structured in layers. The root node with no incoming edges is on top. To represent GCIs of the form $\top \sqsubseteq C$ the root node is connected to a terminal node with C and sends all tokens directly to this terminal node. All other successors of the root node are concept nodes. The root node takes an element of the form $\sigma = \rho r \in \mathsf{N_R}^*$ and sends the token (ρ, r) to all successor nodes. A successor of a concept node can only be another concept node or a role node. A role node leads directly to an inter-element node and inter-element nodes lead to terminal nodes.

Example 1. As an example consider a TBox that contains the following GCIs:

$$A_2 \sqcap A_4 \sqcap A_5 \sqcap \forall r_1.A_3 \sqcap \forall r_1.A_4 \sqcap \forall r_2.A_1 \sqsubseteq B_7,$$
$$\forall r_2.A_3 \sqcap \forall r_2.A_4 \sqsubseteq B_8,$$
$$\forall r_1.A_6 \sqsubseteq \forall r_1.B_9.$$

The corresponding Rete network is displayed in Figure 1.

5 Implementation and Evaluation of $\mathcal{FL}_o wer$

The $\mathcal{FL}_o wer$ reasoner is implemented in Java and it takes as input a general $\mathcal{FL}_0$-TBox $\mathcal{T}$ in OWL format [10]. It implements three different reasoning tasks.

Subsumption: Given two OWL classes A and B decide whether $A \sqsubseteq_{\mathcal{T}} B$ holds.
Subsumer set: Given an OWL class A compute all classes B in $\mathcal{T}$ for which $A \sqsubseteq_{\mathcal{T}} B$ holds.
Classification: Decide for all pairs of named OWL classes A and B occurring in $\mathcal{T}$ whether $A \sqsubseteq_{\mathcal{T}} B$ holds.

To decide subsumption $\mathcal{FL}_o wer$ runs $Subs(A, B, \mathcal{T})$ but possibly stops already in case B occurs at the root of the tree. For computing the subsumer set of A a single complete run of $Subs(A, B, \mathcal{T})$ is sufficient, where the choice of B

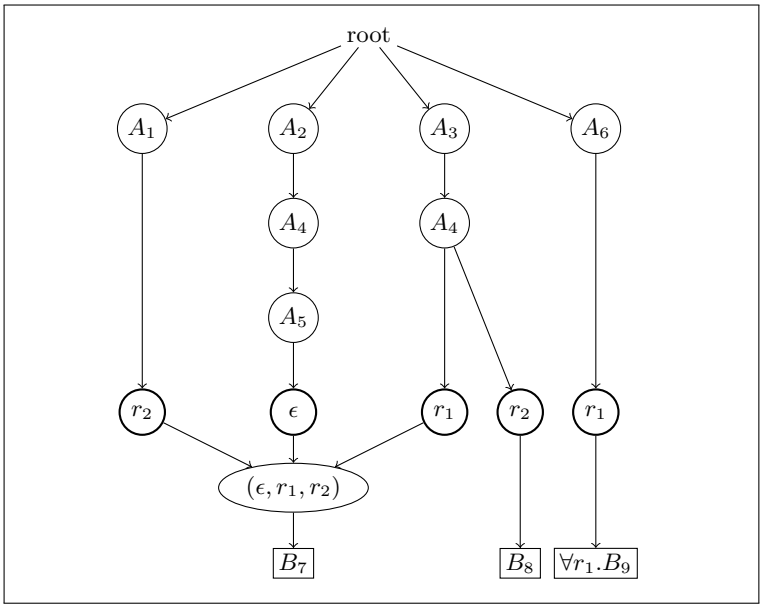

Fig. 1. Rete network for the TBox from Example 1.

is irrelevant. All subsumers of A can be found at the root of the final tree. Classification is done by running $Subs(A, B, \mathcal{T})$ for each named class A in $\mathcal{T}$ separately (again B is irrelevant). The Rete network for $\mathcal{T}$ is created only once and is reused for the remaining runs of $Subs(A, *, \mathcal{T})$ multiple times in parallel. To this end the information that is stored on inter-element nodes and final nodes is stored in an external working memory.

5.1 Test Data

Since there are not enough $\mathcal{FL}_0$-ontologies around to yield a decently large test suite, we had to generate such ontologies. Our goal was to create ontologies, that are close to ontologies from applications rather than to construct purposefully complex ones. To create ontologies with a similar structure to application ontologies, we have chosen to use $\mathcal{FL}_0$-variants of the OWL 2 EL [10] ontologies of the OWL Reasoner Evaluation (ORE) competition benchmark [16]. For each ontology from that benchmark, we essentially flipped the quantifier, i.e., replaced $\exists$ by $\forall$. We have also dropped some axioms involving role inclusions and nominals that cannot be expressed in $\mathcal{FL}_0$. Additionally, ontologies containing fewer than 500 classes were discarded. This resulted in a test suite of 159 ontologies with an average number of 54.000 classes, 5 roles and 170.000 axioms. The largest ontology in the test suite has 981.152 classes, 50 roles and 2.513.918 axioms.

5.2 Evaluation Setup

We have tested $\mathcal{FL}_o wer$ on all three reasoning tasks (subsumption, subsumer set and classification) with the 159 ontologies from our test suite. For a comparison of the reasoning performance of $\mathcal{FL}_o wer$ with tableau-based methods we have chosen the following three state-of-the-art tableau reasoners

- HermiT[5], version 1.3.8.510,
- Openllet[6], version 2.6.3, and
- JFact[7], version 5.0.1.

All three reasoners implement the OWL API [11] and are written in Java. This allows us to measure and compare the time needed for the reasoning tasks alone—excluding the time for initialising the reasoner and for loading the ontologies using the OWL API. These two tasks can take fairly long and thus distort the impression of the performance of solving the reasoning task. In case of $\mathcal{FL}_o wer$ the initialisation phase includes the generation of the Rete network.

As a test system we have used an Intel Core i5 7600K with 4x 3.80GHz and 16 GB of RAM. We have used a setup, where only the runtime of the reasoning task is measured and not the time of loading the ontology. Java was called with *-Xmx8g* to set the maximum allocation pool (heap) size to 8 GB.

[5] `hermit-reasoner.com`
[6] `github.com/Galigator/openllet`
[7] `jfact.sourceforge.net`

While this was sufficient for JFact, HermiT and $\mathcal{FL}_o wer$, Openllet ran into some *OutOfMemory* exceptions. Those runs were counted as unsuccessful for Openllet. For each reasoning task, the time out was set to 6 minutes.

5.3 Evaluation Results

We have obtained the following results for the three reasoning tasks.

Subsumption. Of the 159 ontologies with one subsumption call each for a randomly selected pair of classes, JFact was not able to decide the subsumption in 21 cases, Openllet in 8 cases, HermiT in 3 cases, while $\mathcal{FL}_o wer$ succeeded for all 159 ontologies. In Figure 2 the runtime for the subsumption task is displayed in relation to the number of classes in the ontology. Note the use of a logarithmic scale in this and the following figures. It shows that $\mathcal{FL}_o wer$ performs best overall and is close to and often better than HermiT in terms of reasoning times.

Subsumer set. The task was to compute the subsumer sets of a randomly selected class from each test ontology. This task was not solved within the given timeout of six minutes by JFact in 37 cases, Openllet in 15 cases, HermiT in 13 cases whereas $\mathcal{FL}_o wer$ computed all again. The comparison of the runtimes is displayed in Figure 3. It shows that $\mathcal{FL}_o wer$ exhibit an almost linear behaviour and outperforms the other DL reasoners by at least one order of magnitude. Now, $\mathcal{FL}_o wer$ is especially efficient for this task as it computes the subsumer set

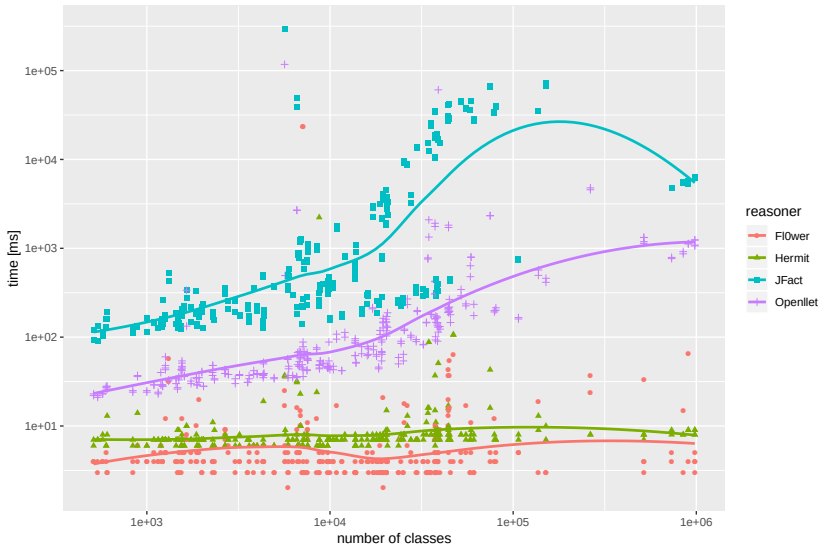

Fig. 2. Computation times for subsumption tests of the different reasoners in relation to ontology size.

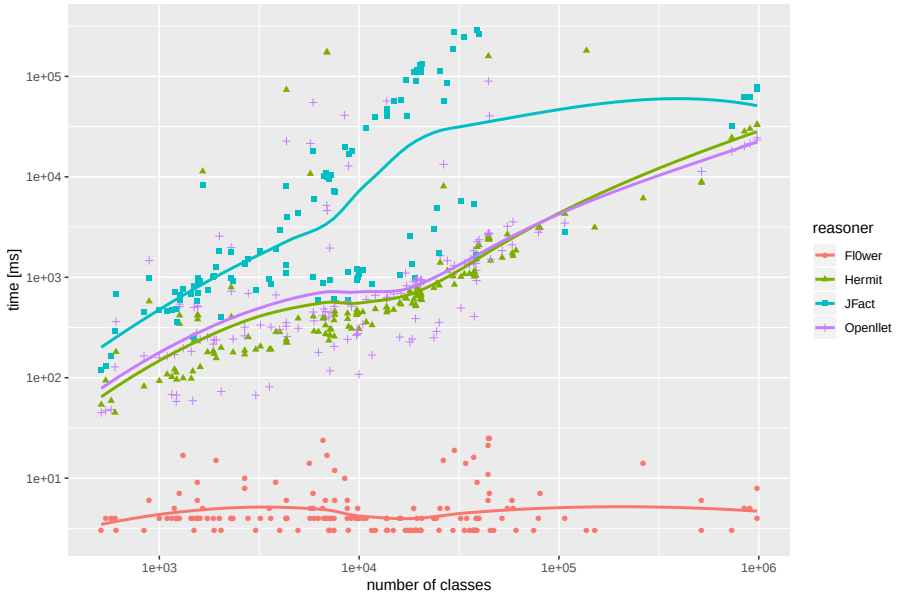

Fig. 3. Computation times for subsumer sets of the different reasoners in relation to ontology size.

of a given class within a single run, whereas the other reasoners need to perform a classification of the whole ontology (compare Figures 3 and 4).

Classification. The unsuccessful attempts to classify test ontologies within the time limit of 6 minutes sum up to 36 for JFact, 15 for Openllet, 13 for HermiT and 3 for $\mathcal{FL}_o wer$. The comparison of the runtime is displayed in Figure 4. $\mathcal{FL}_o wer$ still has an advantage in many cases, but not as huge, as for the subsumer set computation. This seems bit surprising, since the classification method in $\mathcal{FL}_o wer$ is based on its fast subsumer set computation. We assume that this is mostly due to the fact that $\mathcal{FL}_o wer$ implements a naive classification algorithm that simply computes the subsumer set for each named class.

6 Conclusions

We have presented a novel algorithm for deciding subsumption in the DL $\mathcal{FL}_0$ w.r.t. general TBoxes. The approach is to compute a (tree prefix of the) least functional model for the TBox and the potential subsumee. This approach is the basis for developing implementation friendly algorithms for other inferences in $\mathcal{FL}_0$. Furthermore, reasoners for expressive DLs often incorporate reasoners for special fragments and thus a dedicated reasoner for $\mathcal{FL}_0$ may be beneficial for them. Therefore our investigation presented here can contribute to develop a variety of DL reasoning systems.

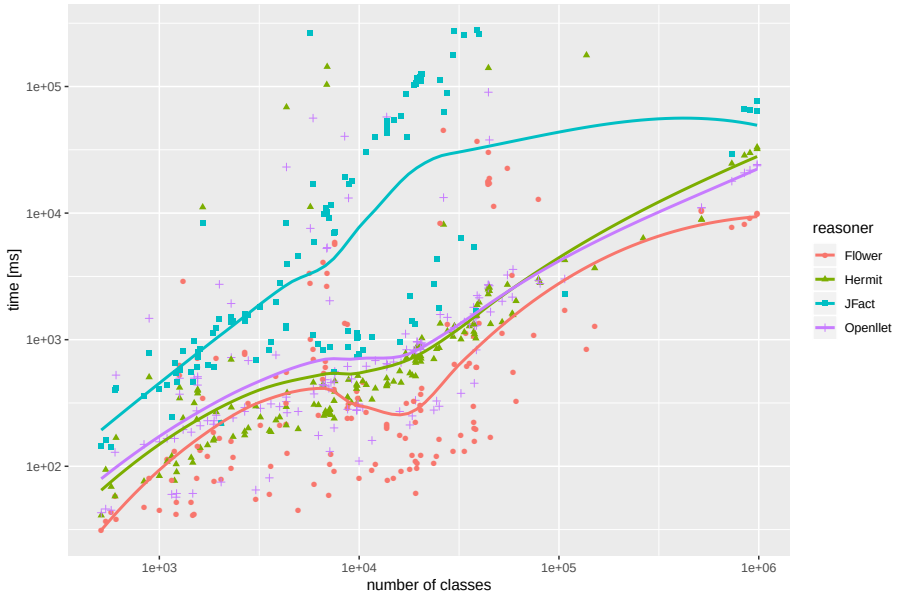

Fig. 4. Computation times for classification of the different reasoners in relation to ontology size.

Our new reasoner $\mathcal{FL}_o wer$ implements the algorithm for computing all subsumers using a variant of the Rete pattern matching algorithm. Our experimental results with our prototype showed that the specialised techniques lead in many cases to a huge performance gain in comparison to highly-optimised tableau reasoners that are designed for more expressive DLs. In particular $\mathcal{FL}_o wer$ is better in most cases than the other systems for testing a single subsumption and for computing classification. For the computation of all subsumers for a given concept $\mathcal{FL}_o wer$ truly outperforms the other DL reasoners by at least one order of magnitude. This is a remarkable results as it raises hopes that a naive method for classification can easily be sped up by simply using massively parallel hardware.

References

1. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, 2005. pp. 364–369. `http://ijcai.org/Proceedings/05/Papers/0372.pdf`
2. Baader, F., Fernández Gil, O., Pensel, M.: Standard and Non-Standard Inferences in the Description Logic $\mathcal{FL}_0$ using Tree Automata. LTCS-Report 18-04, Chair for Automata Theory, Institute for Theoretical Computer Science, TU Dresden, Dresden, Germany (2018), see http://lat.inf.tu-dresden.de/research/reports.html
3. Baader, F., Fernández Gil, O., Marantidis, P.: Matching in the Description Logic $\mathcal{FL}_0$ with respect to General TBoxes. In: LPAR-22. 22nd International Conference

on Logic for Programming, Artificial Intelligence and Reasoning, 2018. EPiC Series in Computing, vol. 57, pp. 76–94. EasyChair (2018), http://www.easychair.org/publications/paper/XrXz
4. Baader, F., Fernández Gil, O., Pensel, M.: Standard and Non-Standard Inferences in the Description Logic $\mathcal{FL}_0$ using Tree Automata. In: GCAI-2018, 4th Global Conference on Artificial Intelligence, 2018. EPiC Series in Computing, vol. 55, pp. 1–14. EasyChair (2018), http://www.easychair.org/publications/paper/H6d9
5. Baader, F., Marantidis, P., Okhotin, A.: Approximate Unification in the Description Logic $\mathcal{FL}_0$. In: Logics in Artificial Intelligence - 15th European Conference, JELIA 2016. LNCS, vol. 10021, pp. 49–63 (2016). https://doi.org/10.1007/978-3-319-48758-8_4
6. Baader, F., Marantidis, P., Pensel, M.: The Data Complexity of Answering Instance Queries in $\mathcal{FL}_0$. In: Companion of the The Web Conference 2018, WWW 2018. pp. 1603–1607. ACM (2018). https://doi.org/10.1145/3184558.3191618
7. Baader, F., Sattler, U.: An Overview of Tableau Algorithms for Description Logics. Studia Logica **69**(1), 5–40 (2001). https://doi.org/10.1023/A:1013882326814
8. Brachman, R.J., Levesque, H.J.: The tractability of subsumption in frame-based description languages. In: Proceedings of the National Conference on Artificial Intelligence. 1984. pp. 34–37. AAAI Press (1984), http://www.aaai.org/Library/AAAI/1984/aaai84-036.php
9. Forgy, C.: Rete: A fast algorithm for the many patterns/many objects match problem. Artif. Intell. **19**(1), 17–37 (1982). https://doi.org/10.1016/0004-3702(82)90020-0
10. Cuenca Grau, B., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P.F., Sattler, U.: OWL 2: The next step for OWL. J. Web Semant. **6**(4), 309–322 (2008)
11. Horridge, M., Bechhofer, S.: The OWL API: A java API for OWL ontologies. Semantic Web **2**(1), 11–21 (2011)
12. Kazakov, Y., de Nivelle, H.: Subsumption of concepts in $\mathcal{FL}_0$ for (cyclic) terminologies with respect to descriptive semantics is PSPACE-complete. In: Proceedings of the 2003 Int. Workshop on Description Logics (DL'03), 2003. CEUR Workshop Proceedings, http://ceur-ws.org/Vol-81/kazakov.pdf
13. Krötzsch, M., Rudolph, S., Hitzler, P.: Complexity boundaries for Horn description logics. In: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence. pp. 452–457. AAAI Press (2007), http://www.aaai.org/Library/AAAI/2007/aaai07-071.php
14. Michel, F.: Entwurf und Implementierung eines Systems zur Entscheidung von Subsumption in der Beschreibungslogik $\mathcal{FL}_0$. Bachelor's thesis, TU Dresden (2017), in German
15. Nebel, B.: Terminological reasoning is inherently intractable. Artif. Intell. **43**(2), 235–249 (1990). https://doi.org/10.1016/0004-3702(90)90087-G, https://doi.org/10.1016/0004-3702(90)90087-G
16. Parsia, B., Matentzoglu, N., Gonçalves, R.S., Glimm, B., Steigmiller, A.: The OWL reasoner evaluation (ORE) 2015 competition report. Journal of Automated Reasoning **59**(4), 455–482 (2017)
17. Romero, A.A., Cuenca Grau, B., Horrocks, I.: More: Modular combination of OWL reasoners for ontology classification. In: International Semantic Web Conference (1). LNCS, vol. 7649, pp. 1–16. Springer (2012)
18. Simancik, F., Kazakov, Y., Horrocks, I.: Consequence-based reasoning beyond Horn ontologies. In: IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, 2011. pp. 1093–1098. IJCAI/AAAI (2011). https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-187