Combining Proofs for Description Logic and Concrete Domain Reasoning

Christian Alrabbaa[1], Franz Baader[1], Stefan Borgwardt[1], Patrick Koopmann[2], and Alisa Kovtunova[1]

[1] Institute of Theoretical Computer Science, TU Dresden, Germany
firstname.lastname@tu-dresden.de
[2] Department of Computer Science, Vrije Universiteit Amsterdam, Netherlands
p.k.koopmann@vu.nl

Abstract. Logic-based approaches to AI have the advantage that their behavior can in principle be explained with the help of proofs of the computed consequences. For ontologies based on Description Logic (DL), we have put this advantage into practice by showing how proofs for consequences derived by DL reasoners can be computed and displayed in a user-friendly way. However, these methods are insufficient in applications where also numerical reasoning is relevant. The present paper considers proofs for DLs extended with concrete domains (CDs) based on the rational numbers, which leave reasoning tractable if integrated into the lightweight DL $\mathcal{EL}_\bot$. Since no implemented DL reasoner supports these CDs, we first develop reasoning procedures for them, and show how they can be combined with reasoning approaches for pure DLs, both for $\mathcal{EL}_\bot$ and the more expressive DL $\mathcal{ALC}$. These procedures are designed such that it is easy to extract proofs from them. We show how the extracted CD proofs can be combined with proofs on the DL side into integrated proofs that explain both the DL and the CD reasoning.

1 Introduction

Description Logics (DLs) [10] are a well-investigated family of logic-based knowledge representation languages, which are frequently used to formalize ontologies for various application domains. As the sizes of DL-based ontologies grow, tools that support improving the quality of such ontologies become more important. DL reasoners[3] can be used to detect inconsistencies and to infer other implicit consequences, such as subsumption relationships. However, for developers or users of DL-based ontologies, it is often hard to understand why a consequence computed by the reasoner actually follows from the given, possibly very large ontology. In principle, such a consequence can be explained by producing a proof for it, which shows how the consequence can be derived from the axioms in the ontology by applying certain easy-to-understand inference rules. In recent work, we have investigated how proofs for consequences derived by DL reasoners can

[3] See http://owl.cs.manchester.ac.uk/tools/list-of-reasoners/

be computed [1,2] and displayed [28] in a user-friendly way [5]. However, like previous work [17,18], this was restricted to DLs without concrete domains.

Concrete domains [9,24] (CDs) have been introduced to enable reference to concrete objects (such as numbers) and predefined predicates on these objects (such as numerical comparisons) when defining concepts. For example, assume that we measure the systolic and the diastolic blood pressure of patients. Then we can describe patients with a pulse pressure of 25 mmHg as $\mathsf{Patient} \sqcap [\mathsf{sys} - \mathsf{dia} = 25]$, where sys and dia are *features* that are interpreted as partial functions that return the systolic and the diastolic blood pressure of a patient, respectively, as rational numbers (if available). We can then state that such patients need attention using the general concept inclusion (GCI)

$$\mathsf{Patient} \sqcap [\mathsf{sys} - \mathsf{dia} = 25] \sqsubseteq \mathsf{NeedAttention}.$$

In the presence of GCIs, integrating a CD into a DL may cause undecidability [25,12] even if solvability of the constraint systems that can be formulated in the CD (in our example, sets of constraints of the form $x - y = q$ for $q \in \mathbb{Q}$) is decidable. One way to overcome this problem is to disallow role paths [16,29,8] in concrete domain restrictions, which means that these restrictions can only constrain feature values of single individuals, as in our example. Comparing feature values of different individuals, such as the age of a woman with that of her children, is then no longer possible.

For tractable (i.e., polynomially decidable) DLs like $\mathcal{EL}_\bot$, preserving decidability is not sufficient: one wants to preserve tractability. As shown in [8], this is the case if one integrates a so-called p-admissible concrete domain into $\mathcal{EL}_\bot$. The only numerical p-admissible concrete domain exhibited in [8] is the CD $\mathcal{D}_{\mathbb{Q},diff}$, which supports constraints of the form $x = q$, $x > q$, and $x + q = y$ (for constants $q \in \mathbb{Q}$). Recently, additional p-admissible concrete domains have been introduced in [12], such as $\mathcal{D}_{\mathbb{Q},lin}$, whose constraints are given by linear equations $\sum_{i=1}^{n} a_i x_i = b$. In the present paper, we will concentrate on these two p-admissible CDs, though the developed ideas and techniques can also be used for other CDs. The constraint used in our example can be expressed in both $\mathcal{D}_{\mathbb{Q},diff}$ and $\mathcal{D}_{\mathbb{Q},lin}$. Unfortunately, no implemented DL reasoner supports these two CDs. In particular, the highly efficient $\mathcal{EL}_\bot$ reasoner Elk [19] does not support any concrete domain. Instead of modifying Elk or implementing our own reasoner for $\mathcal{EL}_\bot$ with concrete domains, we develop here an iterative algorithm that interleaves Elk reasoning with concrete domain reasoning. For the CD reasoning, we could in principle employ existing algorithms and implementations, like Gaussian elimination or the simplex method [31,15] for $\mathcal{D}_{\mathbb{Q},lin}$, and SMT systems that can deal with difference logic [22,7], such as Z3,[4] for $\mathcal{D}_{\mathbb{Q},diff}$. However, since our main purpose is to generate proofs, we develop our own reasoning procedures for $\mathcal{D}_{\mathbb{Q},diff}$ and $\mathcal{D}_{\mathbb{Q},lin}$, which may not be as efficient as existing ones, but can easily be adapted such that they produce proofs.

Proofs for reasoning results in $\mathcal{EL}_\bot$ with a p-admissible CD can in principle be represented using the calculus introduced in [8] or an appropriate extension of the

[4] https://theory.stanford.edu/~nikolaj/programmingz3.html

calculus employed by Elk. However, in these calculi, the result of CD reasoning (i.e., that a set of constraints is unsatisfiable or entails another constraint) is used as an applicability condition for certain rules, but the CD reasoning leading to the satisfaction of the conditions is not explained. Instead of augmenting such a proof with separate proofs on the CD side that show why the applicability conditions are satisfied, our goal is to produce a single proof that explains both the $\mathcal{EL}_\bot$ and the CD reasoning in a uniform proof format.

We also consider the integration of the CDs $\mathcal{D}_{\mathbb{Q},diff}$ and $\mathcal{D}_{\mathbb{Q},lin}$ into the more expressive DL $\mathcal{ALC}$. To this purpose, we develop a new calculus for subsumption w.r.t. $\mathcal{ALC}$ ontologies, which is inspired by the one in [21], but has a better worst-case complexity, and then show how it can be extended to deal with concrete domain restrictions. We have implemented our reasoning and proof extraction approaches for DLs with concrete domains and have evaluated them on several self-created benchmarks designed specifically to challenge the CD reasoning and proof generation capabilities. Proofs for all results and more details about the experiments can be found in [4,3].

2 Description Logics with Concrete Domains

We recall the DLs $\mathcal{EL}_\bot$ and $\mathcal{ALC}$ [10], and then discuss their extensions $\mathcal{EL}_\bot[\mathcal{D}]$ and $\mathcal{ALC}[\mathcal{D}]$ with a concrete domain $\mathcal{D}$ [9,8]. Following [12], we use square brackets to indicate that no role paths are allowed. We also introduce the two p-admissible concrete domains $\mathcal{D}_{\mathbb{Q},diff}$ and $\mathcal{D}_{\mathbb{Q},lin}$ [8,12].

2.1 Description Logics

Starting with disjoint, countably infinite sets of *concept* and *role names* $\mathsf{N_C}$ and $\mathsf{N_R}$, $\mathcal{EL}_\bot$ *concepts* are defined by the grammar $C, D ::= \top \mid \bot \mid A \mid C \sqcap D \mid \exists r.C$, where $A \in \mathsf{N_C}$ and $r \in \mathsf{N_R}$. In $\mathcal{ALC}$, we additionally have negation $\neg C$ as concept constructor. As usual, we then define $C \sqcup D := \neg(\neg C \sqcap \neg D)$ and $\forall r.C := \neg\exists r.\neg C$. An $\mathcal{ALC}$ *($\mathcal{EL}_\bot$) TBox* (a.k.a. *ontology*) $\mathcal{O}$ is a finite set of *general concept inclusions (GCIs*, a.k.a. *axioms)* $C \sqsubseteq D$ for $\mathcal{ALC}$ ($\mathcal{EL}_\bot$) concepts C and D. We denote by $\mathsf{sub}(\mathcal{O})$ the set of subconcepts of all concepts appearing in $\mathcal{O}$.

An *interpretation* is a pair $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$, where the *domain* $\Delta^\mathcal{I}$ is a non-empty set, and the *interpretation function* $\cdot^\mathcal{I}$ assigns to every concept name $A \in \mathsf{N_C}$ a set $A^\mathcal{I} \subseteq \Delta^\mathcal{I}$ and to every role name $r \in \mathsf{N_R}$ a binary relation $r^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$. This function is extended to complex concepts by defining $\top^\mathcal{I} := \Delta^\mathcal{I}$, $\bot^\mathcal{I} := \emptyset$, $(\exists r.C)^\mathcal{I} := \{d \in \Delta^\mathcal{I} \mid \exists e \in \Delta^\mathcal{I}.\, (d,e) \in r^\mathcal{I} \wedge e \in C^\mathcal{I}\}$, $(\neg C)^\mathcal{I} = \Delta^\mathcal{I} \setminus C^\mathcal{I}$, and $(C \sqcap D)^\mathcal{I} := C^\mathcal{I} \cap D^\mathcal{I}$. The interpretation $\mathcal{I}$ is a *model* of $C \sqsubseteq D$ if $C^\mathcal{I} \subseteq D^\mathcal{I}$ (written $\mathcal{I} \models C \sqsubseteq D$), and it is a model of an ontology $\mathcal{O}$ ($\mathcal{I} \models \mathcal{O}$) if it is a model of all axioms in $\mathcal{O}$. An ontology $\mathcal{O}$ is *consistent* if it has a model, and an axiom $C \sqsubseteq D$ is *entailed* by $\mathcal{O}$ (written $\mathcal{O} \models C \sqsubseteq D$) if every model of $\mathcal{O}$ is a model of $C \sqsubseteq D$; in this case, we also say that C is *subsumed* by D w.r.t. $\mathcal{O}$. The *classification* of $\mathcal{O}$ is the set $\mathsf{CL}(\mathcal{O}) := \{\langle C, D\rangle \mid C, D \in \mathsf{sub}(\mathcal{O}),\ \mathcal{O} \models C \sqsubseteq D\}$.[5] The three

[5] Often, the classification is done only for concept names in $\mathcal{O}$, but we use a variant that considers all subconcepts, as it is done by the $\mathcal{EL}_\bot$ reasoner Elk.

reasoning problems of deciding consistency, checking subsumption, and computing the classification are mutually reducible in polynomial time. Reasoning is P-complete in $\mathcal{EL}_\bot$ and ExpTime-complete in $\mathcal{ALC}$ [10].

2.2 Concrete Domains.

Concrete domains have been introduced as a means to integrate reasoning about quantitative features of objects into DLs [9,24,12]. Given a set $\mathsf{N_P}$ of *concrete predicates* and an arity $\mathsf{ar}(P) \in \mathbb{N}$ for each $P \in \mathsf{N_P}$, a *concrete domain (CD)* $\mathcal{D} = (\Delta^\mathcal{D}, \cdot^\mathcal{D})$ over $\mathsf{N_P}$ consists of a set $\Delta^\mathcal{D}$ and relations $P^\mathcal{D} \subseteq (\Delta^\mathcal{D})^{\mathsf{ar}(P)}$ for all $P \in \mathsf{N_P}$. We assume that $\mathsf{N_P}$ always contains a nullary predicate $\bot$, interpreted as $\bot^\mathcal{D} := \emptyset$, and a unary predicate $\top$ interpreted as $\top^\mathcal{D} := \Delta^\mathcal{D}$. Given a set $\mathsf{N_V}$ of *variables*, a *constraint* $P(x_1, \ldots, x_{\mathsf{ar}(P)})$, with $P \in \mathsf{N_P}$ and $x_1, \ldots, x_{\mathsf{ar}(P)} \in \mathsf{N_V}$, is a predicate whose argument positions are filled with variables.

Example 1. The concrete domain $\mathcal{D}_{\mathbb{Q},diff}$ has the set $\mathbb{Q}$ of rational numbers as domain and, in addition to $\top$ and $\bot$, the concrete predicates $x = q$, $x > q$, and $x + q = y$, for constants $q \in \mathbb{Q}$, with their natural semantics [8]. For example, $(x + q = y)^{\mathcal{D}_{\mathbb{Q},diff}} = \{(p, r) \in \mathbb{Q} \times \mathbb{Q} \mid p + q = r\}$.[6]

The concrete domain $\mathcal{D}_{\mathbb{Q},lin}$ has the same domain as $\mathcal{D}_{\mathbb{Q},diff}$, but its predicates other than $\{\top, \bot\}$ are given by linear equations $\sum_{i=1}^{n} a_i x_i = b$, for $a_i, b \in \mathbb{Q}$, with the natural semantics [12], e.g. the linear equation $x + y - z = 0$ is interpreted as the ternary addition predicate $(x + y - z = 0)^{\mathcal{D}_{\mathbb{Q},lin}} = \{(p, q, s) \in \mathbb{Q}^3 \mid p + q = s\}$.

The expressivity of these two CDs is orthogonal: The $\mathcal{D}_{\mathbb{Q},diff}$ predicate $x > q$ cannot be expressed as a conjunction of constraints in $\mathcal{D}_{\mathbb{Q},lin}$, whereas the $\mathcal{D}_{\mathbb{Q},lin}$ predicate $x + y = 0$ cannot be expressed in $\mathcal{D}_{\mathbb{Q},diff}$. □

A constraint $\alpha = P(x_1, \ldots, x_{\mathsf{ar}(P)})$ is *satisfied* by an assignment $v \colon \mathsf{N_V} \to \Delta^\mathcal{D}$ (written $v \models \alpha$) if $\big(v(x_1), \ldots, v(x_{\mathsf{ar}(P)})\big) \in P^\mathcal{D}$. An *implication* is of the form $\gamma \to \delta$, where γ is a conjunction and δ a disjunction of constraints; it is *valid* if all assignments satisfying all constraints in γ also satisfy some constraint in δ (written $\mathcal{D} \models \gamma \to \delta$). A conjunction γ of constraints is *satisfiable* if $\gamma \to \bot$ is not valid. The CD $\mathcal{D}$ is *convex* if, for every valid implication $\gamma \to \delta$, there is a disjunct α in δ s.t. $\gamma \to \alpha$ is valid. It is *p-admissible* if it is convex and validity of implications is decidable in polynomial time. This condition has been introduced with the goal of obtaining tractable extensions of $\mathcal{EL}_\bot$ with concrete domains [8].

Example 2. The CDs $\mathcal{D}_{\mathbb{Q},diff}$ and $\mathcal{D}_{\mathbb{Q},lin}$ are both p-admissible, as shown in [8] and [12], respectively. However, if we combined their predicates into a single CD, then we would lose convexity. In fact, $\mathcal{D}_{\mathbb{Q},diff}$ has the constraints $x > 0$ and $x = 0$. In addition, $y > 0$ (of $\mathcal{D}_{\mathbb{Q},diff}$) and $x + y = 0$ (of $\mathcal{D}_{\mathbb{Q},lin}$) express $x < 0$. Thus, the implication $x + y = 0 \to x > 0 \lor x = 0 \lor y > 0$ is valid, but none of the implications $x + y = 0 \to \alpha$ for $\alpha \in \{x > 0,\ x = 0,\ y > 0\}$ is valid. □

[6] The index *diff* in its name is motivated by the fact that such a predicate fixes the difference between the values of two variables.

To integrate a concrete domain $\mathcal{D}$ into description logics, the most general approach uses role paths $r_1 \dots r_k$ followed by a *concrete feature* f to instantiate the variables in constraints, where the r_i are roles and f is interpreted as a partial function $f^{\mathcal{I}} \colon \Delta^{\mathcal{I}} \to \Delta^{\mathcal{D}}$. Using the concrete domain $\mathcal{D}_{\mathbb{Q},lin}$, the concept $\mathsf{Human} \sqcap \exists \mathsf{age}, \mathsf{parent}\,\mathsf{age}.[2x - y = 0]$, for age being a concrete feature and parent a role name, describes humans with a parent that has twice their age.[7] However, in the presence of role paths, p-admissibility of the CD does not guarantee decidability of the extended DL. Even if we just take the ternary addition predicate of $\mathcal{D}_{\mathbb{Q},lin}$, the extension of $\mathcal{ALC}$ with it becomes undecidable [11], and the paper [12] exhibits a p-admissible CD whose integration into $\mathcal{EL}_\bot$ destroys decidability. Therefore, in this paper we disallow role paths, which effectively restricts concrete domain constraints to the feature values of single abstract objects. Under this restriction, the integration of a p-admissible CD leaves reasoning in P for $\mathcal{EL}_\bot$ [8] and in ExpTime for $\mathcal{ALC}$ [23].[8] Disallowing role paths also enables us to simplify the syntax by treating variables directly as concrete features.

Formally, the description logics $\mathcal{EL}_\bot[\mathcal{D}]$ and $\mathcal{ALC}[\mathcal{D}]$ are obtained from $\mathcal{EL}_\bot$ and $\mathcal{ALC}$ by allowing constraints α from the CD $\mathcal{D}$ to be used as concepts, where we employ the notation $[\alpha]$ to distinguish constraints visually from classical concepts. Interpretations $\mathcal{I}$ are extended by associating to each variable $x \in \mathsf{N_V}$ a *partial* function $x^{\mathcal{I}} \colon \Delta^{\mathcal{I}} \to \Delta^{\mathcal{D}}$, and defining $[\alpha]^{\mathcal{I}}$ as the set of all $d \in \Delta^{\mathcal{I}}$ for which (a) the assignment $v_d^{\mathcal{I}}(x) := x^{\mathcal{I}}(d)$ is defined for all variables x occurring in α, and (b) $v_d^{\mathcal{I}} \models \alpha$.

Example 3. Extending the medical example from the introduction, we can state that, for a patient in the intensive care unit, the heart rate and blood pressure are monitored, using the GCI $\mathsf{ICUpatient} \sqsubseteq [\top(\mathsf{hr})] \sqcap [\top(\mathsf{sys})] \sqcap [\top(\mathsf{dia})]$, which says that, for all elements of the concept $\mathsf{ICUpatient}$, the values of the variables hr, sys, dia are defined. The pulse pressure pp can then be defined via $\mathsf{ICUpatient} \sqsubseteq [\mathsf{sys} - \mathsf{dia} - \mathsf{pp} = 0]$. Similarly, the maximal heart rate can be defined by $\mathsf{ICUpatient} \sqsubseteq [\mathsf{maxHR} + \mathsf{age} = 220]$. All the constraints employed in these GCIs are available in $\mathcal{D}_{\mathbb{Q},lin}$. One might now be tempted to use the GCI $\mathsf{ICUpatient} \sqcap ([\mathsf{pp} > 50] \sqcup [\mathsf{hr} > \mathsf{maxHR}]) \sqsubseteq \mathsf{NeedAttention}$ to say that ICU patients whose pulse pressure is larger than 50 mmHG or whose heart rate is larger than their maximal heart rate need attention. However, while $[\mathsf{pp} > 50]$ is a $\mathcal{D}_{\mathbb{Q},diff}$ constraint, it is not available in $\mathcal{D}_{\mathbb{Q},lin}$, and $[\mathsf{hr} > \mathsf{maxHR}]$ is available in neither. But we can raise an alert when the heart rate gets near the maximal one using $[\mathsf{maxHR} - \mathsf{hr} = 5] \sqsubseteq \mathsf{NeedAttention}$ since it is a statement over $\mathcal{D}_{\mathbb{Q},lin}$. ◻

3 Combined Concrete and Abstract Reasoning

We start by showing how classification in $\mathcal{EL}_\bot[\mathcal{D}]$ can be realized by interleaving a classifier for $\mathcal{EL}_\bot$ with a constraint solver for $\mathcal{D}$. Then we describe our constraint solvers for $\mathcal{D}_{\mathbb{Q},lin}$ and $\mathcal{D}_{\mathbb{Q},diff}$.

[7] See [23] for syntax and semantics of concepts using role paths.

[8] The result in [23] applies to p-admissible CDs $\mathcal{D}$ since it is easy to show that the extension of $\mathcal{D}$ with the negation of its predicates satisfies the required conditions.

Algorithm 1: Classification algorithm for $\mathcal{EL}_\bot[\mathcal{D}]$

$\mathcal{O}' := \mathcal{O}^{-\mathcal{D}}$, $\mathcal{N} := \emptyset$
while $\mathcal{N} \neq \mathsf{CL}(\mathcal{O}')$ **do**
  $\mathcal{N} := \mathsf{CL}(\mathcal{O}')$
  foreach $C \in \mathsf{sub}(\mathcal{O}^{-\mathcal{D}})$ **do**
    $\mathbf{D}_C := \{\alpha \in \mathcal{C}(\mathcal{O}) \mid \langle C, A_\alpha\rangle \in \mathsf{CL}(\mathcal{O}')\}$
    if $\mathcal{D} \models \bigwedge \mathbf{D}_C \to \bot$ **then**
      $\mathcal{O}' := \mathcal{O}' \cup \{\prod_{\alpha\in\mathbf{D}_C} A_\alpha \sqsubseteq \bot\}$
    else
      $\mathcal{O}' := \mathcal{O}' \cup \{\prod_{\alpha\in\mathbf{D}_C} A_\alpha \sqsubseteq A_\beta \mid \beta \in \mathcal{C}(\mathcal{O}), \mathcal{D} \models \bigwedge \mathbf{D}_C \to \beta\}$
return $\mathcal{N}[A_\alpha \mapsto \alpha \mid \alpha \in \mathcal{C}(\mathcal{O})]$

3.1 Reasoning in $\mathcal{EL}_\bot[\mathcal{D}]$

The idea is that we can reduce reasoning in $\mathcal{EL}_\bot[\mathcal{D}]$ to reasoning in $\mathcal{EL}_\bot$ by abstracting away CD constraints by new concept names, and then adding GCIs that capture the interactions between constraints. To be more precise, let $\mathcal{D}$ be a p-admissible concrete domain, $\mathcal{O}$ an $\mathcal{EL}_\bot[\mathcal{D}]$ ontology, and $\mathcal{C}(\mathcal{O})$ the finite set of constraints occurring in $\mathcal{O}$. We consider the ontology $\mathcal{O}^{-\mathcal{D}}$ that results from replacing each $\alpha \in \mathcal{C}(\mathcal{O})$ by a fresh concept name A_α. Since $\mathcal{D}$ is p-admissible, the valid implications over the constraints in $\mathcal{C}(\mathcal{O})$ can then be fully encoded by the $\mathcal{EL}_\bot$ ontology

$$\mathcal{O}_\mathcal{D} := \{A_{\alpha_1} \sqcap \cdots \sqcap A_{\alpha_n} \sqsubseteq \bot \mid \alpha_1, \ldots, \alpha_n \in \mathcal{C}(\mathcal{O}),\ \mathcal{D} \models \alpha_1 \wedge \cdots \wedge \alpha_n \to \bot\} \cup$$
$$\{A_{\alpha_1} \sqcap \cdots \sqcap A_{\alpha_n} \sqsubseteq A_\beta \mid \alpha_1, \ldots, \alpha_n, \beta \in \mathcal{C}(\mathcal{O}),\ \mathcal{D} \models \alpha_1 \wedge \cdots \wedge \alpha_n \to \beta\}.$$

The definition of $\mathcal{O}_\mathcal{D}$ is an adaptation of the construction introduced in [23, Theorem 2.14] for the more general case of admissible concrete domains. The problem is, however, that $\mathcal{O}_\mathcal{D}$ is usually of exponential size since it considers all subsets $\{\alpha_1, \ldots, \alpha_n\}$ of $\mathcal{C}(\mathcal{O})$. Thus, the reasoning procedure for $\mathcal{EL}_\bot[\mathcal{D}]$ obtained by using $\mathcal{O}^{-\mathcal{D}} \cup \mathcal{O}_\mathcal{D}$ as an abstraction of $\mathcal{O}$ would also be exponential. To avoid this blow-up, we test implications of the form $\alpha_1 \wedge \cdots \wedge \alpha_n \to \bot$ and $\alpha_1 \wedge \cdots \wedge \alpha_n \to \beta$ for validity in $\mathcal{D}$ only if this information is needed, i.e., if there is a concept C that is subsumed by the concept names $A_{\alpha_1}, \ldots, A_{\alpha_n}$.

The resulting approach for classifying the $\mathcal{EL}_\bot[\mathcal{D}]$ ontology $\mathcal{O}$, i.e., for computing $\mathsf{CL}(\mathcal{O}) = \{\langle C, D\rangle \mid C, D \in \mathsf{sub}(\mathcal{O}),\ \mathcal{O} \models C \sqsubseteq D\}$ is described in Algorithm 1, where we assume that $\mathsf{CL}(\mathcal{O}')$ is computed by a polynomial-time $\mathcal{EL}_\bot$ classifier, such as ELK, and that the validity of implications in $\mathcal{D}$ is tested using an appropriate constraint solver for $\mathcal{D}$. Since $\mathcal{D}$ is assumed to be p-admissible, there is a constraint solver that can perform the required tests in polynomial time. Thus, we can show that this algorithm is sound and complete, and also runs in polynomial time.

Theorem 4. *Algorithm 1 computes* $\mathsf{CL}(\mathcal{O})$ *in polynomial time.*

Next, we show how constraint solvers for $\mathcal{D}_{\mathbb{Q},lin}$ and $\mathcal{D}_{\mathbb{Q},diff}$ can be obtained.

3.2 Reasoning in $\mathcal{D}_{\mathbb{Q},lin}$

To decide whether a finite conjunction of linear equations is satisfiable or whether it implies another equation, we can use Gaussian elimination [31], which iteratively eliminates variables from a set of linear constraints in order to solve them. Each elimination step consists of a choice of constraint α that is used to eliminate a variable x_i from another constraint γ by adding a suitable multiple $q \in \mathbb{Q}$ of α, such that, in the sum $\gamma + q\alpha$, the coefficient a_i of x_i becomes 0. This can be used to eliminate x_i from all constraints except α, which can then be discarded to obtain a system of constraints with one less variable. For example, using $\alpha\colon 2x + 3y = 5$ to eliminate x from $\gamma\colon 4x - 6y = 1$ using $q = -2$ yields the new equation $-12y = -9$.

To decide whether $\alpha_1 \wedge \cdots \wedge \alpha_n \to \bot$ is valid in $\mathcal{D}_{\mathbb{Q},lin}$, we must test whether the system of linear equations $\alpha_1, \ldots, \alpha_n$ is unsolvable. For this, we apply Gaussian elimination to this system. If we obtain a constraint of the form $0 = b$ for non-zero b, then the system is unsolvable; otherwise, we obtain $0 = 0$ after all variables have been eliminated, which shows solvability. In case $\alpha_1 \wedge \cdots \wedge \alpha_n \to \bot$ is not valid, Algorithm 1 requires us to test whether $\alpha_1 \wedge \cdots \wedge \alpha_n \to \beta$ is valid for constraints β different from $\bot$. This is the case iff the equation β is a linear combination of the equations $\alpha_1, \ldots, \alpha_n$. For this, we can also apply Gaussian elimination steps to eliminate all variables from β using the equations $\alpha_1, \ldots, \alpha_n$. If this results in the constraint $0 = 0$, it demonstrates that β is a linear combination; otherwise, it is not.

In principle, one could use standard libraries from linear algebra (e.g. for Gaussian elimination or the simplex method [31,15,27]) to implement a constraint solver for $\mathcal{D}_{\mathbb{Q},lin}$. We decided to create our own implementation based on Gaussian elimination, mainly for two reasons. First, most existing numerical libraries are optimized for performance and use floating-point arithmetic. Hence, the results may be erroneous due to repeated rounding [13]. Second, even if rational arithmetic with arbitrary precision is used [15], it is not trivial to extract from these tools a step-by-step account of how the verdict (valid or not) was obtained, which is a crucial requirement for extracting proofs.

3.3 Reasoning in $\mathcal{D}_{\mathbb{Q},diff}$

The constraints of $\mathcal{D}_{\mathbb{Q},diff}$ can in principle be simulated in *difference logic*, which consists of Boolean combinations of expressions of the form $x - y \leq q$, and for which reasoning can be done using the Bellman-Ford algorithm for detecting negative cycles [22,7]. However, it is again not clear how proofs for the validity of implications can be extracted from the run of such a solver. For this reason, we implemented a simple saturation procedure that uses the rules in Fig. 1 to derive implied constraints, where side conditions are shown in gray; these rules are similar to the rewrite rules for DL-Lite queries with CDs in [6]. We eagerly apply the rules $\mathsf{R}_{\neq}$, $\mathsf{R}_{<}$, and $\mathsf{R}^{+}_{\neq}$, which means that we only need to keep one constraint of the form $x + q = y$ in memory, for each pair (x, y). Since $x > q$ implies $x > p$ for all $p < q$, it similarly suffices to remember one unary

$$\frac{x = q \quad x = p}{\bot}\,\mathsf{R}_{\neq} : q \neq p \qquad \frac{x + q = y \quad y + p = z}{x + (q + p) = z}\,\mathsf{R}_{+} \qquad \frac{}{x + 0 = x}\,\mathsf{R}_{0}$$

$$\frac{x + q = y \quad x + p = y}{\bot}\,\mathsf{R}^{+}_{\neq} : q \neq p \qquad \frac{x = q \quad y = p}{x + (p - q) = y}\,\mathsf{R}_{-} \qquad \frac{x + q = y}{y + (-q) = x}\,\mathsf{R}_{\leftrightarrow}$$

$$\frac{x = q \quad x > p}{\bot}\,\mathsf{R}_{<} : q < p \qquad \frac{x = q \quad x + p = y}{y = q + p}\,\mathsf{R}_{=} \qquad \frac{x > q \quad x + p = y}{y > q + p}\,\mathsf{R}_{>}$$

Fig. 1. Saturation rules for $\mathcal{D}_{\mathbb{Q},diff}$ constraints

Algorithm 2: Reasoning algorithm for $\mathcal{D}_{\mathbb{Q},diff}$

Input: An implication $\bigwedge \mathbf{D} \to \beta$ in $\mathcal{D}_{\mathbb{Q},diff}$
Output: true iff $\mathcal{D}_{\mathbb{Q},diff} \models \bigwedge \mathbf{D} \to \beta$

```
D' := saturate(D)
if ⊥ ∈ D' or β ∈ D' then return true
if β is x > q then
    if x = p ∈ D' with p > q then return true
    if x > p ∈ D' with p ≥ q then return true
return false
```

constraint of the form $x = q$ or $x > q$ for each variable x. Apart from the three rules deriving $\bot$, we can prioritize rules in the order R_{-}, $\mathsf{R}_{\leftrightarrow}$, R_{0}, R_{+}, $\mathsf{R}_{=}$, $\mathsf{R}_{>}$, since none of the later rules can enable the applications of earlier rules to derive new constraints. The full decision procedure is described in Algorithm 2.

Theorem 5. *Algorithm 2 terminates in time polynomial in the size of* $\bigwedge \mathbf{D} \to \beta$ *and returns* true *iff* $\mathcal{D}_{\mathbb{Q},diff} \models \bigwedge \mathbf{D} \to \beta$.

4 Proofs for $\mathcal{EL}_{\bot}[\mathcal{D}]$ Entailments

Our goal is now to use the procedures described in Section 3 to obtain separate proofs for the DL part and the CD part of an entailment, which we then want to combine into a single proof, as illustrated in Fig. 2.

Fig. 2(a) shows an example of an Elk-proof, a proof generated by the Elk reasoner [18] for the final ontology $\mathcal{O}' \supseteq \mathcal{O}^{-\mathcal{D}}$ from Algorithm 1. The labels $\mathsf{R}_{\sqsubseteq}$ and $\mathsf{R}^{+}_{\sqcap}$ indicate the rules from the internal calculus of Elk [19], and $(*)$ marks an axiom added by Algorithm 1, where α is $2x + 3y = 5$, β is $4y = 3$, and γ is $4x - 6y = 1$. We now describe how to obtain the proof (b) for the CD implication $\alpha \wedge \beta \to \gamma$, and how to integrate both proofs into the $\mathcal{EL}_{\bot}[\mathcal{D}_{\mathbb{Q},lin}]$ proof (c).

4.1 Proofs for the Concrete Domains

For $\mathcal{D}_{\mathbb{Q},diff}$, the saturation rules in Fig. 1 can be seen as proof steps. Thus, the algorithms in [1,2] can easily be adapted to extract $\mathcal{D}_{\mathbb{Q},diff}$ proofs. Inferences due

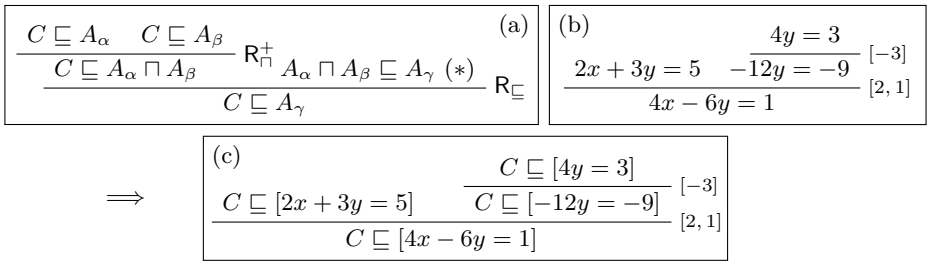

Fig. 2. (a) $\mathcal{EL}_\bot$ proof over $\mathcal{O}'$, (b) $\mathcal{D}_{\mathbb{Q},lin}$ proof and (c) integrated $\mathcal{EL}_\bot[\mathcal{D}_{\mathbb{Q},lin}]$ proof.

to Lines 2, 4 and 5 in Algorithm 2 are captured by the following additional rules:

$$\frac{\bot}{\beta}\,\mathsf{R}_\bot \qquad \frac{x = p}{x > q}\,\mathsf{R}^+_> : p > q \qquad \frac{x > p}{x > q}\,\mathsf{R}^-_> : p \geq q$$

For $\mathcal{D}_{\mathbb{Q},lin}$, inferences are Gaussian elimination steps that derive $\sigma + c\rho$ from linear constraints σ and ρ, and we label them with $[1, c]$ to indicate that σ is multiplied by 1 and ρ by c. This directly gives us a proof if the conclusion is $\bot$ (or, equivalently, $0 = b$ for non-zero b). However, proofs for implications $\bigwedge \mathbf{D} \to \gamma$ need to be treated differently. The Gaussian method would use $\mathbf{D}$ to eliminate the variables from γ to show that γ is a linear combination of $\mathbf{D}$, and would yield a rather uninformative proof with final conclusion $0 = 0$. To obtain a proof with γ as conclusion, we reverse the proof direction by recursively applying the following transformation starting from an inference step that has γ as a premise:

$$\frac{\sigma \qquad \rho}{\tau}\,[1, c] \qquad \rightsquigarrow \qquad \frac{\rho \qquad \tau}{\sigma}\,[-c, 1] \tag{†}$$

Then we transform the next inference to obtain an inference that has τ as the conclusion, and continue this process until $0 = 0$ becomes a leaf, which we then remove from the proof.

In our example, we would start with the following "proof" for $\mathcal{D} \models \alpha \wedge \beta \to \gamma$:

$$\frac{\dfrac{4x-6y=1 \qquad 2x+3y=5}{-12y=-9}\,[1,-2] \qquad 4y=3}{0=0}\,[1,3]$$

After applying two transformation steps (†), we obtain the proof in Fig. 2(b).

4.2 Combining the Proofs

It remains to integrate the concrete domain proofs into the DL proof over $\mathcal{O}^{-\mathcal{D}}$. As a consequence of Algorithm 1, in Fig. 2(a), the introduced concept names A_α, A_β, A_γ occur in axioms with the same left-hand side C. The idea is to add this *DL context* C to every step of the CD proof (b) to obtain the $\mathcal{EL}_\bot[\mathcal{D}]$-proof (c). This proof replaces the applications of $\mathsf{R}^+_\sqcap$ and $\mathsf{R}_\sqsubseteq$ in the original DL proof (a), and both (a) and (c) have essentially the same leafs and conclusion,

except that the auxiliary concept names $A_\alpha, A_\beta, A_\gamma$ were replaced by the original constraints and the auxiliary axiom $(*)$ was eliminated. In general, such proofs can be obtained by simple post-processing of proofs obtained separately from the DL and CD reasoning components, and we conjecture that the integrated proof (c) is easier to understand in practice than the separate proofs (a) and (b), since the connection between the DL and CD contexts is shown in all steps.

Lemma 6. *Let $\mathcal{O}'$ be the final ontology computed in Algorithm 1. Given an* ELK*-proof $\mathcal{P}'$ for $\mathcal{O}' \models C^{-\mathcal{D}} \sqsubseteq D^{-\mathcal{D}}$ and proofs for all $\mathcal{D}$-implications $\alpha_1 \wedge \cdots \wedge \alpha_n \to \beta$ used in $\mathcal{P}'$, we can construct in polynomial time an $\mathcal{EL}_\bot[\mathcal{D}]$-proof for $\mathcal{O} \models C \sqsubseteq D$.*

5 Generating Proofs for $\mathcal{ALC}[\mathcal{D}]$

For $\mathcal{ALC}[\mathcal{D}]$, a black-box algorithm as for $\mathcal{EL}_\bot[\mathcal{D}]$ is not feasible, even though we consider only p-admissible concrete domains and no role paths. The intuitive reason is that $\mathcal{ALC}$ itself is not convex, and we cannot simply use the classification result to determine which implications $\alpha_1 \wedge \ldots \wedge \alpha_n \to \beta$ in $\mathcal{D}$ are relevant. On the other hand, adding all valid implications is not practical, as there can be exponentially many. We thus need a glass-box approach, i.e. a modified $\mathcal{ALC}$ reasoning procedure that determines the relevant CD implications on-demand.

Moreover, to obtain proofs for $\mathcal{ALC}[\mathcal{D}]$, we need a reasoning procedure that derives new axioms from old ones, and thus classical tableau methods [14,26] are not suited. However, existing consequence-based classification methods for $\mathcal{ALC}$ [30] use complicated calculi that are not needed for our purposes. Instead, we use a modified version of a calculus from [21], which uses only three inference rules, but performs double exponentially many inferences in the worst case. Our modification ensures that we perform at most exponentially many inferences, and are thus worst-case optimal for the ExpTime-complete $\mathcal{ALC}[\mathcal{D}]$.

5.1 A Simple Resolution Calculus for $\mathcal{ALC}$

The calculus represents GCIs $\top \sqsubseteq L_1 \sqcup \cdots \sqcup L_n$ as *clauses* of the form

$$L_1 \sqcup \ldots \sqcup L_n \qquad L_i \ ::= \ A \ \mid \ \neg A \ \mid \ \exists r.D \ \mid \ \forall r.D$$

where $n \geq 0$, $A, D \in \mathsf{N_C}$ and $r \in \mathsf{N_R}$. To decide $\mathcal{O} \models A \sqsubseteq B$, we normalize $\mathcal{O}$ into a set of clauses, introducing fresh concept names for concepts under role restrictions, and add two special clauses $A_{\text{LHS}} \sqcup A$, $A_{\text{RHS}} \sqcup \neg B$, with fresh concept names A_{LHS} and A_{RHS}. The latter are used to track relevant inferences for constructing the final proof, for which we transform all clauses back into GCIs.

Our inference rules are shown in Fig. 3. **A1** is the standard resolution rule from first-order logic, which is responsible for direct inferences on concept names. The rules **r1** and **r2** perform inferences on role restrictions. They consider an existential role restriction $\exists r.D$ and a (possibly empty) set of value restrictions over r, whose conjunction is unsatisfiable due to a clause over the nested concepts. The concept D may not be relevant for this, which is why there are two

$$\textbf{A1}\colon \frac{C_1 \sqcup A, \quad C_2 \sqcup \neg A}{C_1 \sqcup C_2} \qquad \textbf{r1}\colon \frac{C \sqcup \exists r.D, \quad C_1 \sqcup \forall r.D_1, \ \ldots, \ C_n \sqcup \forall r.D_n, \quad \neg D_1 \sqcup \ldots \sqcup \neg D_n}{C \sqcup C_1 \sqcup \ldots \sqcup C_n}$$

$$\textbf{r2}\colon \frac{C \sqcup \exists r.D, \quad C_1 \sqcup \forall r.D_1, \ \ldots, \ C_n \sqcup \forall r.D_n, \quad \neg D \sqcup \neg D_1 \sqcup \ldots \sqcup \neg D_n}{C \sqcup C_1 \sqcup \ldots \sqcup C_n}$$

Fig. 3. Inference rules for $\mathcal{ALC}$ clauses.

rules. Those rules are the main difference to the original calculus in [21], where a more expensive, incremental mechanism was used instead. To transform this calculus into a practical method, we use optimizations common for resolution-based reasoning in first-order logic: ordered resolution, a set-of-support strategy, as well as backward and forward subsumption deletion. In particular, our set-of-support strategy starts with a set of *support clauses* containing only the clauses with A_{LHS} and A_{RHS}. Inferences are always performed with at least one clause from this set, and the conclusion becomes a new support clause. If a support clause contains a literal $\exists r.D/\forall r.D$, we also add all clauses containing $\neg D$ as support clauses [20].

5.2 Incorporating the Concrete Domain and Creating the Proof

To incorporate concrete domains, we again work on the translation $\mathcal{O}^{-\mathcal{D}}$ replacing each constraint α with A_α. In $\mathcal{ALC}[\mathcal{D}]$, constraints can also occur in negated form, which means that we can have literals $\neg A_\alpha$ expressing the negation of a constraint. We keep track of the set $\mathbf{D}$ of concrete domain constraints α for which A_α occurs positively in a support clause. We then use the proof procedure for $\mathcal{D}$ (see Section 4.1) to generate all implications of the form $\alpha_1 \wedge \ldots \wedge \alpha_n \to \beta$, where $\{\alpha_1, \ldots, \alpha_n\} \subseteq \mathbf{D}$ is subset-minimal, for which we add the corresponding clauses $\neg A_{\alpha_1} \sqcup \ldots \sqcup \neg A_{\alpha_n} \sqcup A_\beta$. If $\beta = \bot$, we instead add $\neg A_{\alpha_1} \sqcup \ldots \sqcup \neg A_{\alpha_n}$.

Theorem 7. *Let $\mathcal{O}$ be an $\mathcal{ALC}[\mathcal{D}]$ ontology and $\mathcal{N}$ the normalization of $\mathcal{O}^{-\mathcal{D}}$. Then our method takes at most exponential time, and it derives $A_{LHS} \sqcup A_{RHS}$ or a subclause from $\mathcal{N}$ iff $\mathcal{O} \models C \sqsubseteq D$.*

Proofs generated using the calculus operate on the level of clauses. We transform them into proofs of $\mathcal{O}^{-\mathcal{D}} \models A \sqsubseteq B$ by 1) adding inference steps that reflect the normalization, 2) if necessary, adding an inference to produce $A_{\text{LHS}} \sqcup A_{\text{RHS}}$ from a subclause 3) replacing A_{LHS} by $\neg A$ and A_{RHS} by B, 4) replacing all other introduced concept names by the complex concepts they were introduced for, and 5) transforming clauses into more human-readable GCIs using some simple rewriting rules (see [4] for details). In the resulting proof, the initial clauses $A \sqcup A_{LHS}$ and $\neg B \sqcup A_{RHS}$ then correspond to the tautologies $A \sqsubseteq A$ and $B \sqsubseteq B$. To get a proof for $\mathcal{O} \models A \sqsubseteq B$, we use a procedure similar to the one from Section 4.2 to integrate concrete domain proofs. Because the integration requires only simple structural transformations, the complexity of computing the combined proofs is determined by the corresponding complexities for the DL and

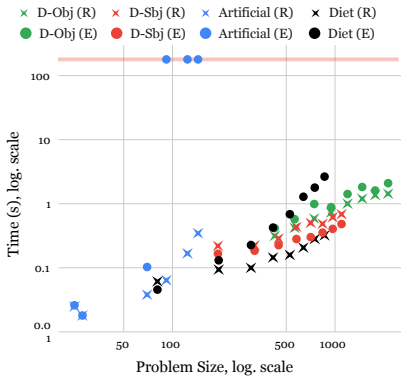

Fig. 4. $\mathcal{EL}_\bot[\mathcal{D}]$: time for reasoning (R) and explanation (E) vs. problem size

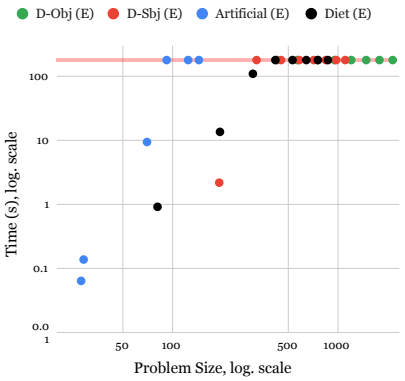

Fig. 5. $\mathcal{ALC}[\mathcal{D}]$: total reasoning and explanation time vs. problem size

the concrete domain. We can thus extend the approaches from [1,2] to obtain complexity bounds for finding proofs of small size and depth.

Theorem 8. *For $\mathcal{D} \in \{\mathcal{D}_{\mathbb{Q},lin}, \mathcal{D}_{\mathbb{Q},diff}\}$, deciding the existence of a proof of at most a given* size *can be done in* NP *for $\mathcal{EL}_\bot[\mathcal{D}]$, and in* NExpTime *for $\mathcal{ALC}[\mathcal{D}]$. For proof* depth*, the corresponding problem is in* P *for $\mathcal{EL}_\bot[\mathcal{D}_{\mathbb{Q},diff}]$, in* NP *for $\mathcal{EL}_\bot[\mathcal{D}_{\mathbb{Q},lin}]$, and in* ExpTime *for $\mathcal{ALC}[\mathcal{D}]$ (for both concrete domains).*

6 Implementation and Experiments

We implemented the algorithms described above and evaluated their performance and the produced proofs on the self-created benchmarks *Diet*, *Artificial*, *D-Sbj* and *D-Obj*, each of which consists of multiple instances scaling from small to medium-sized ontologies. The latter two benchmarks are formulated in $\mathcal{EL}_\bot[\mathcal{D}_{\mathbb{Q},diff}]$, the rest in $\mathcal{EL}_\bot[\mathcal{D}_{\mathbb{Q},lin}]$. Our tool is written using Java 8 and Scala. We used Elk 0.5, Lethe 0.85 and OWL API 4. The experiments were performed on Debian Linux 10 (24 Intel Xeon E5-2640 CPUs, 2.50GHz) with 25 GB maximum heap size and a timeout of 3 minutes for each task. Fig. 4 shows the runtimes of the approaches for $\mathcal{EL}_\bot[\mathcal{D}]$ from Sections 3 and 4 for reasoning and explanation depending on the *problem size*, which counts all occurrences of concept names, role names, and features in the ontology. A more detailed description of the benchmarks and results can be found in [4].

We observe that pure reasoning time (crosses in Fig. 4) scales well w.r.t. problem size. Producing proofs was generally more costly than reasoning, but the times were mostly reasonable. However, there are several *Artificial* instances for which the proof construction times out (blue dots). This is due to the nondeterministic choices of which linear constraints to use to eliminate the next variable, which we resolve using the Dijkstra-like algorithm described in [2], which results

in an exponential runtime in the worst case. Another downside is that some proofs were very large (> 2000 inference steps in *D-Obj*). However, we designed our benchmarks specifically to challenge the CD reasoning and proof generation capabilities (in particular, nearly all constraints in each ontology are necessary to entail the target axiom), and these results may improve for realistic ontologies.

Further analysis revealed that the reasoning times were often largely due to the calls to ELK (ranging from 23% in *Diet* to 75% in *Artificial*), which shows that the CD reasoning does not add a huge overhead, unless the number of variables per constraint grows very large (e.g. up to 88 in *Diet*). In comparison to the incremental use of ELK as a black-box reasoner, the hypothetical "ideal" case of calling ELK only once on the final saturated ontology $\mathcal{O}'$ would not save a lot of time (average gain ranging from 42% in *Diet* to 14% in *D-Obj*), which shows that the incremental nature of our approach is also not a bottleneck.

Fig. 5 shows the runtime of the $\mathcal{ALC}[\mathcal{D}]$ calculus from Section 5. As expected, it performs worse than the dedicated $\mathcal{EL}_\bot[\mathcal{D}]$ algorithms. In particular, currently there is a bottleneck for the $\mathcal{D}_{\mathbb{Q},diff}$ benchmarks (*D-Sbj* and *D-Obj*) that is due an inefficiency in the computation of the relevant CD implications $\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \beta$. In order to evaluate the increased expressivity supported by the $\mathcal{ALC}[\mathcal{D}]$ reasoner, we have also incorporated axioms with negation and universal restrictions into the *Artificial* benchmark. Currently, however, the reasoner can solve only the smallest such instance before reaching the timeout.

We also compared our CD reasoning algorithms with Z3 [27], which supports linear arithmetic (for $\mathcal{D}_{\mathbb{Q},lin}$) and difference logic (for $\mathcal{D}_{\mathbb{Q},diff}$). Ignoring the overhead stemming from the interface between Java and C++, the runtime of both approaches was generally in the same range, but our algorithms were faster on many CD reasoning problems. This may be due to the fact that, although our algorithms for $\mathcal{D}_{\mathbb{Q},lin}$ and $\mathcal{D}_{\mathbb{Q},diff}$ are not optimized very much, they are nevertheless tailored towards very specific convex fragments: linear arithmetic with only $=$, and difference logic with only $x + q = y$ and $x > q$, respectively.

7 Conclusion

We have shown that it is feasible to support p-admissible concrete domains in DL reasoning algorithms, and even to produce integrated proofs for explaining consequences in the DLs $\mathcal{EL}_\bot[\mathcal{D}]$ and $\mathcal{ALC}[\mathcal{D}]$, for the p-admissible concrete domains $\mathcal{D}_{\mathbb{Q},lin}$ and $\mathcal{D}_{\mathbb{Q},diff}$. In this work, we have restricted our attention to ontologies containing only GCIs (i.e., TBoxes) and to classification as the main reasoning problem. However, the extension of our methods to data and reasoning about individuals, e.g. $\mathsf{fred} : \mathsf{ICUpatient} \sqcap [\mathsf{hr} = 90]$, encoded in so-called *ABoxes* [10], is straightforward. Likewise, the approach for computing $\mathcal{EL}_\bot[\mathcal{D}]$ proofs can be generalized to use other reasoning calculi for $\mathcal{EL}_\bot$ instead of the one employed by ELK, which makes very small proof steps and thus generates rather large proofs.

One major problem with using proofs to explain consequences is that they may become quite large. This problem already occurs for pure DLs without CDs, and has also shown up in some of our benchmarks in this paper. One possibility

to alleviate this problem is to use an interactive proof visualization tool like Evonne [28], which allows zooming into parts of the proof and hiding uninteresting or already inspected parts. Since the integrated proofs that we generate have the same shape as pure DL proofs, they can be displayed using Evonne. It would, however, be interesting to add features tailored to CD reasoning, such as visualizing the solution space of a system of linear equations.

In Example 3, we have seen that it would be useful to have the constraints of $\mathcal{D}_{\mathbb{Q},lin}$ and $\mathcal{D}_{\mathbb{Q},diff}$ available in a single CD. Such a CD $\mathcal{D}$ would still preserve decidability if integrated into $\mathcal{ALC}$. However, since $\mathcal{D}$ is no longer convex, our reasoning approach for $\mathcal{ALC}[\mathcal{D}]$ does not apply. Thus, it would also be interesting to see whether this approach can be extended to *admissible* CDs $\mathcal{D}$ [9,23], i.e. CDs that are closed under negation and for which satisfiability of sets of constraints is decidable.

Acknowledgments This work was supported by the DFG grant 389792660 as part of TRR 248 (`https://perspicuous-computing.science`).

References

1. Alrabbaa, C., Baader, F., Borgwardt, S., Koopmann, P., Kovtunova, A.: Finding small proofs for description logic entailments: Theory and practice. In: LPAR (2020). `https://doi.org/10.29007/nhpp`
2. Alrabbaa, C., Baader, F., Borgwardt, S., Koopmann, P., Kovtunova, A.: Finding good proofs for description logic entailments using recursive quality measures. In: CADE (2021). `https://doi.org/10.1007/978-3-030-79876-5_17`
3. Alrabbaa, C., Baader, F., Borgwardt, S., Koopmann, P., Kovtunova, A.: Combining proofs for description logic and concrete domain reasoning - RuleML+RR23 - Resources (2023). `https://doi.org/10.5281/zenodo.8208780`
4. Alrabbaa, C., Baader, F., Borgwardt, S., Koopmann, P., Kovtunova, A.: Combining proofs for description logic and concrete domain reasoning (technical report) (2023). `https://doi.org/10.48550/arXiv.2308.03705`
5. Alrabbaa, C., Borgwardt, S., Hirsch, A., Knieriemen, N., Kovtunova, A., Rothermel, A.M., Wiehr, F.: In the head of the beholder: Comparing different proof representations. In: RuleML+RR (2022). `https://doi.org/10.1007/978-3-031-21541-4_14`
6. Alrabbaa, C., Koopmann, P., Turhan, A.: Practical query rewriting for DL-Lite with numerical predicates. In: GCAI (2019). `https://doi.org/10.29007/gqll`
7. Armando, A., Castellini, C., Giunchiglia, E., Maratea, M.: A SAT-based decision procedure for the Boolean combination of difference constraints. In: SAT (2004). `https://doi.org/10.1007/11527695_2`
8. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: IJCAI (2005), `http://ijcai.org/Proceedings/05/Papers/0372.pdf`
9. Baader, F., Hanschke, P.: A scheme for integrating concrete domains into concept languages. In: IJCAI (1991), `http://ijcai.org/Proceedings/91-1/Papers/070.pdf`
10. Baader, F., Horrocks, I., Lutz, C., Sattler, U.: An Introduction to Description Logic. Cambridge Univ. Press (2017). `https://doi.org/10.1017/9781139025355`

11. Baader, F., Rydval, J.: Description logics with concrete domains and general concept inclusions revisited. In: IJCAR (2020). https://doi.org/10.1007/978-3-030-51074-9_24
12. Baader, F., Rydval, J.: Using model theory to find decidable and tractable description logics with concrete domains. JAR **66**(3), 357–407 (2022). https://doi.org/10.1007/s10817-022-09626-2
13. Barlow, J.L., Bareiss, E.H.: Probabilistic error analysis of Gaussian elimination in floating point and logarithmic arithmetic. Computing **34**(4), 349–364 (1985). https://doi.org/10.1007/BF02251834
14. Donini, F.M., Massacci, F.: ExpTime tableaux for $\mathcal{ALC}$. AIJ **124**(1), 87–138 (2000). https://doi.org/10.1016/S0004-3702(00)00070-9
15. Dutertre, B., de Moura, L.M.: A fast linear-arithmetic solver for DPLL(T). In: CAV (2006). https://doi.org/10.1007/11817963_11
16. Haarslev, V., Möller, R., Wessel, M.: The description logic $\mathcal{ALCNH}_{\mathcal{R}+}$ extended with concrete domains: A practically motivated approach. In: IJCAR (2001). https://doi.org/10.1007/3-540-45744-5_4
17. Horridge, M., Parsia, B., Sattler, U.: Justification oriented proofs in OWL. In: ISWC (2010). https://doi.org/10.1007/978-3-642-17746-0_23
18. Kazakov, Y., Klinov, P., Stupnikov, A.: Towards reusable explanation services in Protege. In: DL (2017), https://ceur-ws.org/Vol-1879/paper31.pdf
19. Kazakov, Y., Krötzsch, M., Simancik, F.: The incredible ELK - From polynomial procedures to efficient reasoning with $\mathcal{EL}$ ontologies. JAR **53**(1), 1–61 (2014). https://doi.org/10.1007/s10817-013-9296-3
20. Koopmann, P., Del-Pinto, W., Tourret, S., Schmidt, R.A.: Signature-based abduction for expressive description logics. In: KR (2020). https://doi.org/10.24963/kr.2020/59
21. Koopmann, P., Schmidt, R.A.: Uniform interpolation of $\mathcal{ALC}$-ontologies using fixpoints. In: FroCoS (2013). https://doi.org/10.1007/978-3-642-40885-4_7
22. Kroening, D., Strichman, O.: Decision Procedures - An Algorithmic Point of View, Second Edition. EATCS (2016). https://doi.org/10.1007/978-3-662-50497-0
23. Lutz, C.: The complexity of description logics with concrete domains. Ph.D. thesis (2002), https://nbn-resolving.org/urn:nbn:de:hbz:82-opus-3032
24. Lutz, C.: Description logics with concrete domains - A survey. In: Adv. in Modal Logic 4 (2002), http://www.aiml.net/volumes/volume4/Lutz.ps
25. Lutz, C.: NExpTime-complete description logics with concrete domains. ACM TOCL **5**(4), 669–705 (2004). https://doi.org/10.1145/1024922.1024925
26. Motik, B., Shearer, R., Horrocks, I.: Hypertableau reasoning for description logics. JAIR **36**, 165–228 (2009). https://doi.org/10.1613/jair.2811
27. de Moura, L.M., Bjørner, N.S.: Z3: An efficient SMT solver. In: TACAS (2008). https://doi.org/10.1007/978-3-540-78800-3_24
28. Méndez, J., Alrabbaa, C., Koopmann, P., Langner, R., Baader, F., Dachselt, R.: Evonne: A visual tool for explaining reasoning with OWL ontologies and supporting interactive debugging. CGF (2023), https://doi.org/10.1111/cgf.14730
29. Pan, J.Z., Horrocks, I.: Reasoning in the $\mathcal{SHOQ}(\mathrm{D_n})$ description logic. In: DL (2002), https://ceur-ws.org/Vol-53/Pan-Horrocks-shoqdn-2002.ps
30. Simancik, F., Kazakov, Y., Horrocks, I.: Consequence-based reasoning beyond Horn ontologies. In: IJCAI (2011). https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-187
31. Turner, P.R.: Gauss elimination: Workhorse of linear algebra (1995), https://apps.dtic.mil/sti/pdfs/ADA313547.pdf, NAWCADPAX-96-194-TR