Combining Proofs for Description Logic and Concrete Domain Reasoning (Extended Version)

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) [8] 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 [26] in a user-friendly way [3]. However, like previous work [15,16], this was restricted to DLs without concrete domains.

Concrete domains [7,22] (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 [23,10] 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 [14,27,6] 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 [6], 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 [6] 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 [10], 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 [17] 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 [29,13] for $\mathcal{D}_{\mathbb{Q},lin}$, and SMT systems that can deal with difference logic [20,5], 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 [6] 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 [19], 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 at `https://lat.inf.tu-dresden.de/~alrabbaa/rulemlrr23/cdProofs.html`.

2 Description Logics with Concrete Domains

We recall the DLs $\mathcal{EL}_\bot$ and $\mathcal{ALC}$ [8], and then discuss their extensions $\mathcal{EL}_\bot[\mathcal{D}]$ and $\mathcal{ALC}[\mathcal{D}]$ with a concrete domain $\mathcal{D}$ [7,6]. Following [10], 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 $\mathcal{D}_{\mathbb{Q},lin}$ [6,10].

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 $\mathcal{ALC}$ [8].

2.2 Concrete Domains.

Concrete domains have been introduced as a means to integrate reasoning about quantitative features of objects into DLs [7,22,10]. 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 $P \in \mathsf{N_P}$. For technical reasons, 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 [6]. For example, $(x + q = y)^{\mathcal{D}_{\mathbb{Q},diff}} = \{(p, r) \in \mathbb{Q} \times \mathbb{Q} \mid p + q = r\}$. The index *diff* in its name is motivated by the fact that such a predicate fixes the difference between the values of two variables.

The concrete domain $\mathcal{D}_{\mathbb{Q},lin}$ has the same domain as $\mathcal{D}_{\mathbb{Q},diff}$, but its predicates other than $\top$ and $\bot$ are given by linear equations $\sum_{i=1}^{n} a_i x_i = b$, for rational numbers a_i, b, with the natural semantics [10]. For example, 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. For example, 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 [6].

Example 2. The CDs $\mathcal{D}_{\mathbb{Q},diff}$ and $\mathcal{D}_{\mathbb{Q},lin}$ are both p-admissible, as shown in [6] and [10], 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.

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 $\mathsf{Human} \sqcap \exists \mathsf{age}, \mathsf{parent}\,\mathsf{age}.[2x - y = 0]$ describes humans that have a parent that has twice their age. 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 [9], and the paper [10] 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 $\mathcal{EL}_\bot$ [6] and in ExpTime for $\mathcal{ALC}$ [21].[6] 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 the first constraint is a $\mathcal{D}_{\mathbb{Q},diff}$ constraint, it is not available in $\mathcal{D}_{\mathbb{Q},lin}$, and the second one is available in neither. But we can raise an alert when the heart rate gets near the maximal one using $[\mathsf{maxHR} - \mathsf{hr} = 5] \sqsubseteq \mathsf{NeedAttention}$.

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 $\mathcal{D}_{\mathbb{Q},diff}$.

[6] The result in [21] 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 $\mathcal{EL}_\bot[\mathcal{D}]$

$\mathcal{O}' := \mathcal{O}^{-\mathcal{D}}$, $\mathbf{N} := \emptyset$
while $\mathbf{N} \neq \mathsf{CL}(\mathcal{O}')$ **do**
  $\mathbf{N} := \mathsf{CL}(\mathcal{O}')$
  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 $\mathbf{N}[A_\alpha \mapsto \alpha \mid \alpha \in \mathcal{C}(\mathcal{O})]$

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, $\mathcal{O}$ be an $\mathcal{EL}_\bot[\mathcal{D}]$ ontology, and $\mathcal{C}(\mathcal{O})$ be 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 $\mathcal{EL}_\bot[\mathcal{D}]$ ontology

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

The definition of $\mathcal{O}_\mathcal{D}$ is an adaptation of the construction introduced in [21, 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 be 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 [29], 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 [29,13,25]) 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 [11]. Second, even if rational arithmetic with arbitrary precision is used [13], 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 [20,5]. 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 [4]. 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

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

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

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

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

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

Input: An implication $\bigwedge \mathbf{D} \to \beta$ in $\mathcal{D}_{\mathbb{Q},\mathit{diff}}$
Output: true iff $\mathcal{D}_{\mathbb{Q},\mathit{diff}} \models \bigwedge \mathbf{D} \to \beta$
1 $\mathbf{D}' := \mathsf{saturate}(\mathbf{D})$
2 **if** $\bot \in \mathbf{D}'$ **or** $\beta \in \mathbf{D}'$ **then return** true
3 **if** β is $x > q$ **then**
4 | **if** $x = p \in \mathbf{D}'$ with $p > q$ **then return** true
5 | **if** $x > p \in \mathbf{D}'$ with $p \geq q$ **then return** true
6 **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* $\mathcal{D}_{\mathbb{Q},\mathit{diff}} \models \bigwedge \mathbf{D} \to \beta$.

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

Our goal is now to use the procedures described in the previous section 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 a proof generated by the ELK reasoner [16] 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 [17], 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 $\mathcal{EL}_{\bot}[\mathcal{D}_{\mathbb{Q},\mathit{lin}}]$ proof (c).

4.1 Proofs for the Concrete Domains

For $\mathcal{D}_{\mathbb{Q},\mathit{diff}}$, we can use the algorithms from [1,2] to extract proofs from the rule instances used in the saturation in Fig. 2 (see Fig. 1). Inferences due to Lines 2, 4

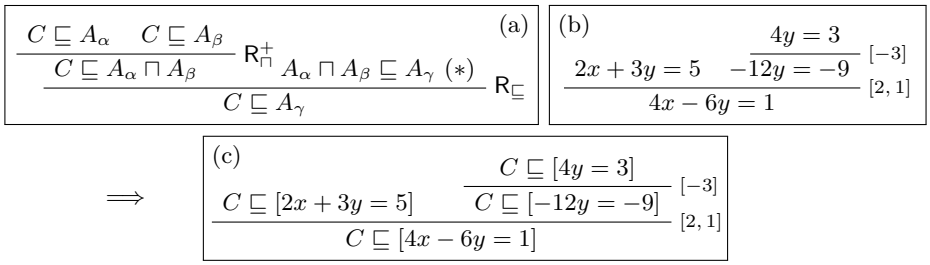

Fig. 2. (a) $\mathcal{EL}_\bot$ proof over $\mathcal{O}'$, (b) $\mathcal{D}_{\mathbb{Q},lin}$ proof and (c) integrated $\mathcal{EL}_\bot[\mathcal{D}_{\mathbb{Q},lin}]$ proof.

and 5 in this algorithm are captured by the following additional rules:

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

For $\mathcal{D}_{\mathbb{Q},lin}$, inferences are Gaussian elimination steps that derive $\beta + c\alpha$ from β and α, and we label them with $[1, c]$ to indicate that β is multiplied by 1 and α 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 $\bigwedge \mathbf{D} \to \beta$ need to be treated differently. Since we use $\mathbf{D}$ to eliminate the variables from β to show that β is a linear combination of $\mathbf{D}$, our approach would yield a proof with final conclusion $0 = 0$. In our example, we could get the following "proof" for $\mathcal{D} \models \alpha \wedge \beta \to \gamma$:

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

To obtain a proof with γ as conclusion, we reverse the proof direction by recursively applying the following transformation to make a premise α_1 into a conclusion.

$$\frac{\alpha_1 \qquad \alpha_2}{\alpha_3}\,[c,d] \qquad \rightsquigarrow \qquad \frac{\alpha_2 \qquad \alpha_3}{\alpha_1}\,[-\tfrac{d}{c}, \tfrac{1}{c}]$$

We then transform the next inference to obtain an inference that has α_3 as the conclusion, and continue this process until $0 = 0$ becomes a leaf, which we then remove from the proof. The result for our example is shown 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 [12,24] are not suited. However, existing consequence-based classification methods for $\mathcal{ALC}$ [28] use complicated calculi that are not needed for our purposes. Instead, we use a modified version of a calculus from [19], 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 rules. Those rules are the main difference to the original calculus in [19], where a more expensive, incremental mechanism was used instead. To transform this

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

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

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

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 [18].

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 $\mathbf{N}$ be 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 $\mathbf{N}$ 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 the appendix 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 the concrete domain. We can thus extend the approaches from [1,2] to obtain complexity bounds for finding proofs of small size and depth.

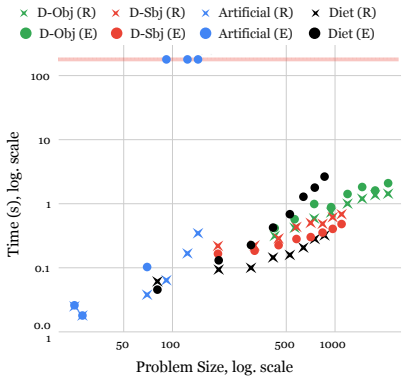

Fig. 4. $\mathcal{EL}_\bot[\mathcal{D}]$: time for reasoning (R) and explanation (E) vs. problem size

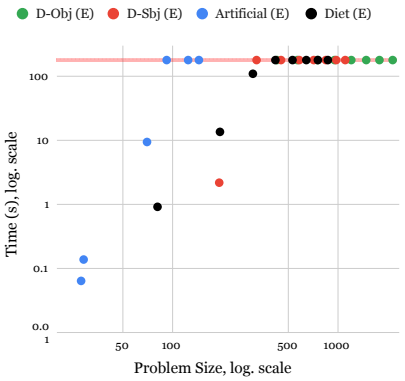

Fig. 5. $\mathcal{ALC}[\mathcal{D}]$: total reasoning and explanation time vs. problem size

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 the appendix.

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 [25], 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* [8], 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 [26], 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 $\mathcal{D}$ [7,21], i.e. CDs that are closed under negation and for which satisfiability of sets of constraints is decidable.

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., 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
4. Alrabbaa, C., Koopmann, P., Turhan, A.: Practical query rewriting for DL-Lite with numerical predicates. In: GCAI (2019). https://doi.org/10.29007/gqll
5. 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
6. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: IJCAI (2005), http://ijcai.org/Proceedings/05/Papers/0372.pdf
7. 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
8. Baader, F., Horrocks, I., Lutz, C., Sattler, U.: An Introduction to Description Logic. Cambridge Univ. Press (2017). https://doi.org/10.1017/9781139025355
9. 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
10. 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
11. 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
12. 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
13. Dutertre, B., de Moura, L.M.: A fast linear-arithmetic solver for DPLL(T). In: CAV (2006). https://doi.org/10.1007/11817963_11

14. 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
15. 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
16. Kazakov, Y., Klinov, P., Stupnikov, A.: Towards reusable explanation services in Protege. In: DL (2017), https://ceur-ws.org/Vol-1879/paper31.pdf
17. 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
18. 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
19. 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
20. 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
21. 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
22. Lutz, C.: Description logics with concrete domains - A survey. In: Adv. in Modal Logic 4 (2002), http://www.aiml.net/volumes/volume4/Lutz.ps
23. Lutz, C.: NExpTime-complete description logics with concrete domains. ACM TOCL **5**(4), 669–705 (2004). https://doi.org/10.1145/1024922.1024925
24. Motik, B., Shearer, R.D.C., Horrocks, I.: Hypertableau reasoning for description logics. JAIR **36**, 165–228 (2009). https://doi.org/10.1613/jair.2811
25. 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
26. 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
27. 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
28. 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
29. Turner, P.R.: Gauss elimination: Workhorse of linear algebra (1995), https://apps.dtic.mil/sti/pdfs/ADA313547.pdf, NAWCADPAX-96-194-TR

A Omitted Proofs in Sections 3 and 4

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

Proof. We first observe that, in each iteration, the while loop adds at most polynomially many axioms to $\mathcal{O}'$ (at most one for each concept $C \in \mathsf{sub}(\mathcal{O}^{-\mathcal{D}})$ and constraint $\beta \in \mathcal{C}(\mathcal{O})$), which are of polynomial size. Therefore, $\mathsf{CL}(\mathcal{O}')$ can always be computed in polynomial time [6] in Lines 3 and 5. Moreover, there are at most polynomially many iterations since we only add axioms to $\mathcal{O}'$, and thus due to the monotonicity of entailment, each set $\mathbf{D}_C$ in Line 5 monotonically increases from one iteration to the next, and is bounded by $\mathcal{C}(\mathcal{O})$. Since the loop terminates once all sets $\mathbf{D}_C$ remain the same, there can be at most $|\mathsf{sub}(\mathcal{O}^{-\mathcal{D}})| \cdot |\mathcal{C}(\mathcal{O})|$ iterations of the while loop.

It remains to prove correctness. In each step, $\mathcal{O}'[A_\alpha \mapsto \alpha \mid \alpha \in \mathcal{C}(\mathcal{O})]$ contains only axioms that are entailed by $\mathcal{O}$, since the concept names A_α replace exactly the occurrences of the concrete constraints α occurring in $\mathcal{O}$. Consequently, if the output of the algorithm contains $\langle C, D\rangle$, then $\mathcal{O} \models C \sqsubseteq D$, which means that the algorithm is sound. It remains to show completeness, i.e. that the output contains all such tuples. Take two concepts $C^\dagger, D^\dagger \in \mathsf{sub}(\mathcal{O})$ such that $\langle C^\dagger, D^\dagger\rangle$ is not returned. We construct a model $\mathcal{I}$ of $\mathcal{O}$, based on the contents of $\mathcal{O}'$ and $\mathbf{N} = \mathsf{CL}(\mathcal{O}')$ in the last iteration, such that $\mathcal{I} \not\models C^\dagger \sqsubseteq D^\dagger$. We start with a model $\mathcal{I}'$ of the final ontology $\mathcal{O}'$:

- $\Delta^{\mathcal{I}'} := \{C \in \mathsf{sub}(\mathcal{O}^{-\mathcal{D}}) \mid \langle C, \bot\rangle \notin \mathbf{N}\}$,
- for all $A \in \mathsf{N_C}$, $A^{\mathcal{I}'} := \{C \in \Delta^{\mathcal{I}'} \mid \langle C, A\rangle \in \mathbf{N}\}$,
- for all $r \in \mathsf{N_R}$, $r^{\mathcal{I}'} := \{\langle C, D\rangle \in \Delta^{\mathcal{I}'} \times \Delta^{\mathcal{I}'} \mid \langle C, \exists r.D\rangle \in \mathbf{N}\}$

Since $\mathsf{CL}(\mathcal{O}')$ computes all entailed inclusions between subconcepts in $\mathcal{O}'$, one can easily show that $\mathcal{I}' \models \mathcal{O}'$ [17]: from the construction it follows by induction on the structure of $C, D \in \mathsf{sub}(\mathcal{O}^{-\mathcal{D}})$ that $C \in D^{\mathcal{I}'}$ iff $\langle C, D\rangle \in \mathbf{N}$ and $\langle C, \bot\rangle \notin \mathbf{N}$. Let $C \sqsubseteq D \in \mathcal{O}'$ and $E \in C^{\mathcal{I}'}$. Then, $\langle E, C\rangle \in \mathbf{N}$, and since $\mathcal{O}' \models C \sqsubseteq D$, also $\langle E, D\rangle \in \mathbf{N}$, and thus $E \in D^{\mathcal{I}'}$.

We argue that $\mathcal{I}'$ can be extended to an interpretation $\mathcal{I}$ such that, for each $\alpha \in \mathcal{C}(\mathcal{O})$, we have $A_\alpha^{\mathcal{I}'} = \alpha^{\mathcal{I}}$. It is possible to ensure $A_\alpha^{\mathcal{I}'} \subseteq \alpha^{\mathcal{I}}$, because, if $\mathbf{D}_C$ for some domain element $C \in \Delta^{\mathcal{I}'}$ is unsatisfiable, we would have added an axiom $\bigsqcap_{\alpha \in \mathbf{D}_C} A_\alpha \sqsubseteq \bot$ in Line 7, and by completeness of $\mathsf{CL}(\mathcal{O}')$, we would have added $\langle C, \bot\rangle$ to $\mathbf{N}$, and thus not included C in $\Delta^{\mathcal{I}'}$. Consequently, we can assign values to the concrete features such that all constraints in $\mathbf{D}_C$ are satisfied, i.e. $C \in A_\alpha^{\mathcal{I}'}$ implies $C \in \alpha^{\mathcal{I}}$, since $A_\alpha \in \mathbf{D}_C$ by construction. However, we have to ensure that for all $\alpha \in \mathcal{C}(\mathcal{O})$, also $\alpha^{\mathcal{I}} \subseteq A_\alpha^{\mathcal{I}'}$ holds, because otherwise there might be GCIs in $\mathcal{O}$ with concrete constraints on the left-hand side that are not satisfied. Intuitively, we have to make sure that we do not accidentally satisfy more concrete domain constraints than necessary. For a fixed C, let $\overline{\mathbf{D}}_C := \{\beta \in \mathcal{C}(\mathcal{O}) \mid \langle C, A_\beta\rangle \notin \mathbf{N}\}$. Due to Line 9, there is no $\beta \in \overline{\mathbf{D}}_C$ such that $\mathcal{D} \models \bigwedge \mathbf{D}_C \to \beta$. By convexity of $\mathcal{D}$, this implies $\mathcal{D} \not\models \bigwedge \mathbf{D}_C \to \bigvee \overline{\mathbf{D}}_C$. In other words, we can find an assignment to the concrete features $x^{\mathcal{I}}(C)$ such that every constraint in $\mathbf{D}_C$ is satisfied, and no constraint in $\overline{\mathbf{D}}_C$ is satisfied. Doing this for

all domain elements $C \in \Delta^{\mathcal{I}'}$, we ensure that $A_\alpha^{\mathcal{I}'} = \alpha^{\mathcal{I}}$ for all $\alpha \in \mathcal{C}(\mathcal{O})$, and thus $\mathcal{I} \models \mathcal{O}$.

We now show that $\mathcal{I} \not\models C^\dagger \sqsubseteq D^\dagger$ by proving that $C' \in (C')^{\mathcal{I}} = (C^\dagger)^{\mathcal{I}}$ and $C' \notin (D')^{\mathcal{I}} = (D^\dagger)^{\mathcal{I}}$, where C' and D' result from replacing in $C^\dagger$ and $D^\dagger$ each $\alpha \in \mathcal{C}(\mathcal{O})$ by A_α. From our assumption that $\langle C^\dagger, D^\dagger \rangle$ is not returned by Algorithm 1, we know that $\langle C', D' \rangle \notin \mathbf{N}$. Therefore, by completeness of $\mathsf{CL}(\mathcal{O}') = \mathbf{N}$, we cannot have $\langle C', \bot \rangle \in \mathbf{N}$, and thus $C' \in \Delta^{\mathcal{I}'}$ and $C' \notin (D')^{\mathcal{I}'} = (D')^{\mathcal{I}}$. Completeness of $\mathsf{CL}(\mathcal{O}')$ also yields that $\langle C', C' \rangle \in \mathbf{N}$, and thus $C' \in (C')^{\mathcal{I}'} = (C')^{\mathcal{I}}$. This shows that $\mathcal{I} \not\models C^\dagger \sqsubseteq D^\dagger$ and concludes the proof. □

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

Proof. We show that each rule can produce only quadratically many new constraints in the number of variables, and hence the algorithm terminates after polynomial time. For R_-, $\mathsf{R}_\leftrightarrow$, and R_0, this follows from the fact that these rules can be applied at most once for each variable x or each pair of variables (x, y). Using R_+, we can produce at most 2 constraints of the form $x + q = y$ for each pair (x, y), since the saturation stops as soon as $\mathsf{R}_{\neq}^+$ can be applied. At the end, $\mathsf{R}_=$ and $\mathsf{R}_>$ can also be applied only once for each pair (x, y).

It is clear that each of the rules is sound. Hence, it remains to show that $\mathcal{D}_{\mathbb{Q},diff} \not\models \bigwedge \mathbf{D} \to \beta$ whenever the Algorithm returns false. In this case, $\mathbf{D}'$ cannot contain β nor $\bot$, i.e. the rules $\mathsf{R}_{\neq}^+$, $\mathsf{R}_<$, and $\mathsf{R}_{\neq}^+$ are not applicable to $\mathbf{D}'$. We construct an assignment f to show that $\bigwedge \mathbf{D} \to \beta$ is not valid, i.e. which satisfies all constraints in $\mathbf{D}$, but not $D\beta$. For any $x = q \in \mathbf{D}'$, we set $f(x) := q$. Consider now the remaining unsatisfied constraints of the form $x + q = y$ and $x > q$ in $\mathbf{D}'$, for which there cannot exist constraints $x = p$ nor $y = p$ in $\mathbf{D}'$. Due to $\mathsf{R}_\leftrightarrow$ and R_+, the directed graph G with edges $\{(x, y) \mid x + q = y \in \mathbf{D}'\}$ consists of unconnected cliques. If we fix one variable x in each clique C and some value $f(x)$, then the values $f(y)$ of the other variables $y \in C$ are determined by the unique constraints $x + q = y$ that must exist in $\mathbf{D}'$. Such an assignment also satisfies all constraints $y + p = z$ with $y, z \in C$ since they are implied by corresponding constraints $x + q = y$ and $x + r = z$, for which we must have $r - q = p$ due to $\mathsf{R}_\leftrightarrow$, R_+, and $\mathsf{R}_{\neq}^+$. We now set $f(x)$ to an arbitrary value that only has to satisfy the constraint $x > q$ in case one exists in $\mathbf{D}'$, and fix the values of all other $y \in C$ accordingly. All constraints $y > p \in \mathbf{D}'$ for $y \in C$ will be satisfied due to $\mathsf{R}_>$.

If β is either of the form $x + q = y$ or $x = q$, it could happen that we have chosen values $f(x), f(y)$ that satisfy β by accident. However, since we assumed that $\beta \notin \mathbf{D}'$, this is only possible if there is no constraint $x + p = y$ or $x = p$, respectively, in $\mathbf{D}'$. In particular, x and y cannot be in the same clique. Thus, we can increase the value $f(x)$ by an arbitrary amount (and the values of all variables connected to x in G accordingly) in order to ensure that β is not satisfied, while $\mathbf{D} \subseteq \mathbf{D}'$ remains satisfied.

If β is of the form $x > q$, then we know that $\mathbf{D}'$ contains neither $x = p$ with $p > q$ nor $x > p$ with $p \geq q$. If $\mathbf{D}'$ contains $x = p$ with $p \leq q$, then

$f(x) = p$ does not satisfy β. If $\mathbf{D}'$ contains $x > p$ with $p < q$, then it is possible to choose $f(x) := q$ (and adjust the values of x's clique accordingly), which also does not satisfy β, but still satisfies $\mathbf{D}$. Finally, if $\mathbf{D}'$ contains no constraint of the form $x = p$ or $x > p$, the value of $f(x)$ can be chosen arbitrarily while still satisfying $\mathbf{D}$, and so we can again choose $f(x) := q$ (and adjust the connected variables accordingly). □

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 $\mathcal{O} \models C \sqsubseteq D$.*

Proof. Given a $\mathcal{D}$-proof $\mathcal{P}$ and a concept C (the *context*), we construct the $\mathcal{EL}_\bot[\mathcal{D}]$-proof $\mathcal{P}_C$ by replacing each inference in $\mathcal{P}$ as follows:

$$\frac{\gamma_1 \quad \cdots \quad \gamma_m}{\delta}\,\mathsf{R} \quad \rightsquigarrow \quad \frac{C \sqsubseteq A_{\gamma_1} \quad \cdots \quad C \sqsubseteq A_{\gamma_m}}{C \sqsubseteq A_\delta}\,\mathsf{R}$$

In this transformation, all steps remain sound, and a $\mathcal{D}$-proof of β becomes an $\mathcal{EL}_\bot$-proof of $C \sqsubseteq A_\beta$.

To integrate this into the original $\mathcal{EL}_\bot$-proof $\mathcal{P}'$ from ELK, we need to analyze in which contexts the newly introduced axioms $A_{\alpha_1} \sqcap \cdots \sqcap A_{\alpha_n} \sqsubseteq A_\beta$ can appear in $\mathcal{P}'$ (the case with $\bot$ instead of A_β is similar). The relevant inference rules of ELK are

$$\frac{C \sqsubseteq D \quad D \sqsubseteq E}{C \sqsubseteq E}\,\mathsf{R}_\sqsubseteq \qquad \frac{C \sqsubseteq D \quad C \sqsubseteq E}{C \sqsubseteq D \sqcap E}\,\mathsf{R}^+_\sqcap \qquad \frac{C \sqsubseteq \exists r.D \quad D \sqsubseteq E}{C \sqsubseteq \exists r.E}\,\mathsf{R}_\exists$$

with the side condition that the axiom $D \sqsubseteq E$ in $\mathsf{R}_\sqsubseteq$ is always an element of the input ontology (here, $\mathcal{O}'$). We distinguish three cases in the following.

– In the best case, $A_{\alpha_1} \sqcap \cdots \sqcap A_{\alpha_n} \sqsubseteq A_\beta$ is used in $\mathcal{P}'$ in an inference step

$$\frac{C \sqsubseteq A_{\alpha_1} \sqcap \cdots \sqcap A_{\alpha_n} \qquad A_{\alpha_1} \sqcap \cdots \sqcap A_{\alpha_n} \sqsubseteq A_\beta}{C \sqsubseteq A_\beta}\,\mathsf{R}_\sqsubseteq \tag{1}$$

as in our example proof in Fig. 2(a). Then, we replace (1) by $\mathcal{P}_C$, where $\mathcal{P}$ is the $\mathcal{D}$-proof of $\alpha_1 \wedge \cdots \wedge \alpha_n \to \beta$. The proof $\mathcal{P}_C$ already has the same conclusion $C \sqsubseteq A_\beta$ as (1). However, since the leafs of $\mathcal{P}_C$ are of the form $C \sqsubseteq A_{\alpha_i}$, in general we also need to add the inferences

$$\frac{C \sqsubseteq A_{\alpha_1} \sqcap \ldots \sqcap A_{\alpha_n}}{C \sqsubseteq A_{\alpha_i}}\,\mathsf{R}^-_\sqcap$$

to $\mathcal{P}'$ to connect $\mathcal{P}_C$ to the original premise $C \sqsubseteq A_{\alpha_1} \sqcap \cdots \sqcap A_{\alpha_n}$ of (1). In Fig. 2(c), this was not necessary since the nodes labeled by $C \sqsubseteq A_\alpha$ and $C \sqsubseteq A_\beta$ already existed in (a), and hence we could use them directly and omit the step $\mathsf{R}^+_\sqcap$ in (a).

– An axiom $A_{\alpha_1} \sqcap \cdots \sqcap A_{\alpha_n} \sqsubseteq A_\beta$ could also be used in $\mathsf{R}_\exists$:

$$\frac{C \sqsubseteq \exists r.\big(A_{\alpha_1} \sqcap \cdots \sqcap A_{\alpha_n}\big) \qquad A_{\alpha_1} \sqcap \cdots \sqcap A_{\alpha_n} \sqsubseteq A_\beta}{C \sqsubseteq \exists r.A_\beta}\,\mathsf{R}_\exists \tag{2}$$

In this case, we cannot use C as the context. Though it would be tempting to translate $\mathcal{D}$-inferences $\frac{\gamma_1 \quad \dots \quad \gamma_m}{\delta}$ into $\frac{C \sqsubseteq \exists r.A_{\gamma_1} \quad \dots \quad C \sqsubseteq \exists r.A_{\gamma_m}}{C \sqsubseteq \exists r.A_\delta}$ to arrive at the conclusion $C \sqsubseteq \exists r.A_\beta$, such inferences would not be sound since $\emptyset \not\models \exists r.A_{\gamma_1} \sqcap \cdots \sqcap \exists r.A_{\gamma_m} \sqsubseteq \exists r.(A_{\gamma_1} \sqcap \cdots \sqcap A_{\gamma_m})$ in $\mathcal{EL}_\bot$. Hence, we simply use $D = A_{\alpha_1} \sqcap \cdots \sqcap A_{\alpha_n}$ itself as the context, i.e. we translate the $\mathcal{D}$-proof $\mathcal{P}$ of $\alpha_1 \wedge \cdots \wedge \alpha_n \to \beta$ into $\mathcal{P}_D$, which provides a proof of the second premise $D \sqsubseteq A_\beta$ of (2). The leafs of $\mathcal{P}_D$ have labels of the form $A_{\alpha_1} \sqcap \cdots \sqcap A_{\alpha_n} \sqsubseteq A_{\alpha_i}$, for which we introduce new inferences $\frac{}{A_{\alpha_1} \sqcap \cdots \sqcap A_{\alpha_n} \sqsubseteq A_{\alpha_i}}$ since they are tautologies that require no further explanation. Using $D = A_{\alpha_1} \sqcap \cdots \sqcap A_{\alpha_n}$ as context here is not ideal, though, since in general n could be quite large, which can make the proof cluttered.

– The last case is when $A_{\alpha_1} \sqcap \cdots \sqcap A_{\alpha_n} \sqsubseteq A_\beta$ appears as the first premise of $\mathsf{R}_\sqsubseteq$ or any premise of $\mathsf{R}^+_\sqcap$. In this case, the left-hand side $C = A_{\alpha_1} \sqcap \cdots \sqcap A_{\alpha_n}$ is propagated to the conclusions of the form $C \sqsubseteq E$ or $C \sqsubseteq D \sqcap E$, which then potentially lead to more axioms of the form $C \sqsubseteq F$. However, since these axioms are not elements of $\mathcal{O}'$, they can never appear as the second premise of $\mathsf{R}_\sqsubseteq$, which means that we cannot find a different DL context for the $\mathcal{D}$-proof $\mathcal{P}$ of $A_{\alpha_1} \sqcap \cdots \sqcap A_{\alpha_n} \sqsubseteq A_\beta$, and we again have to use $C = A_{\alpha_1} \sqcap \cdots \sqcap A_{\alpha_n}$ itself as the context, i.e. we use $\mathcal{P}_C$ to derive $A_{\alpha_1} \sqcap \cdots \sqcap A_{\alpha_n} \sqsubseteq A_\beta$ directly.

Finally, as seen in Figure 2(c), we replace all A_α in the resulting proof by α in order to obtain an $\mathcal{EL}_\bot[\mathcal{D}]$-proof. □

B Omitted Details from Section 5

B.1 Refutational Completeness

We first show refutational completeness of the calculus for $\mathcal{ALC}$. The idea is to present a procedure that constructs a model from a saturated set of clauses that does not contain the empty clause. The construction makes use of linear orderings on clauses and literals that determine how the model is constructed. These orderings are also used in the optimized version of the calculus. While this is a common construction to show refutational completeness of resolution, the peculiarities of our method requires a more refined ordering.

Note the special role of concept names occurring under a role restriction: if a concept name A occurs in a literal $\mathsf{Q}r.A$, then we assume that there are no other positive occurrences of A, that is, literals with A are either of the form $\neg A$ or $\mathsf{Q}r.A$, where Q and r are always the same. Correspondingly, for every concept name A, positive occurrences of A in the set of clauses are either only in literals of the form A, only in literals of the form $\exists r.A$, or only in literals of the form $\forall r.A$.

We now assume a linear order $\prec_l$ on literals such that, for all $A \in \mathsf{N_C}$:

1. If A_1 occurs under an existential role restriction, and A_2 under a value restriction, then $\neg A_1 \prec \neg A_2$.
2. If A occurs under a role restriction, then $\neg A \prec_l L$ for any literal L that is not of the form $\neg A'$, where A' occurs under a role restriction.
3. If A does not occur under a role restriction, then $A \prec_l \neg A$,
4. $\exists r.A_1 \prec_l \forall s.A_2 \prec_l A_3$ for all $r, s \in \mathsf{N_R}$ and A_1, A_2, $A_3 \in \mathsf{N_C}$.

We extend $\prec_l$ to a linear order $\prec$ on clauses, such that $C_1 \prec C_2$ whenever C_2 contains a literal L_2 such that, for every literal L_1 in C_1, $L_1 \prec C_2$.

Theorem 9. *For any set of clauses* $\mathbf{N}$ *satisfying our constraints, our calculus derives a number of clauses that is at most exponential in* $\mathbf{N}$*, and derives the empty clause iff* $\mathbf{N}$ *is inconsistent.*

Proof. As usual, we will use the symbol $\bot$ to denote the empty clause. The number of distinct literals occurring in $\mathbf{N}$ is linearly bounded by its size, and each derived clause is composed of literals occurring in $\mathbf{N}$. Since we represent clauses as sets, this establishes that the algorithm has to terminate after exponentially many steps. Every rule in Figure 3 infers only clauses that are entailed by its premises. Consequently, if $\bot$ is derived, then $\mathbf{N}$ must be inconsistent. It remains to show that if $\bot$ is not derived, then $\mathbf{N}$ has a model.

Let $\mathbf{N}^*$ be the set of clauses generated by Alg_2, and assume $\bot \notin \mathbf{N}^*$. Using the order $\prec$ on clauses, we construct an interpretation $\mathcal{I}$ as fixpoint of an unbounded sequence of interpretations $\mathcal{I}_0$, $\mathcal{I}_1$, ... The first interpretation $\mathcal{I}_0$ is defined by $\Delta^{\mathcal{I}_0} = \{d_0\}$ and $X^{\mathcal{I}_0} = \emptyset$ for all $X \in \mathsf{N_C} \cup \mathsf{N_R}$. The next interpretations are built based on the previous one, where in each step, we add at most one element to the domain. We may thus speak of the *oldest domain element* (that satisfies some condition), by which we mean the domain element added first in the sequence of interpretations. $\mathcal{I}_{i+1}$ is constructed from $\mathcal{I}_i$ as follows. If $\mathcal{I}_i \models \mathbf{N}^*$, then $\mathcal{I}_i = \mathcal{I}_{i+1} = \mathcal{I}$. Otherwise, there is a clause $C \in \mathbf{N}^*$ and a domain element $d \in \Delta^{\mathcal{I}_i}$ s.t. $d \notin C^{\mathcal{I}_i}$. Choose such a pair $\langle d, C\rangle \in \Delta^{\mathcal{I}_i} \times \mathbf{N}^*$ where, among all such pairs, d is the *oldest* domain element (that is, the domain element introduced in $\mathcal{I}_j$ for the smallest index j), and, for the selected d, C is the smallest clause according to the ordering $\prec$. We then distinguish the following cases based on the maximal literal L in C according to $\prec_l$:

- If $L = A \in \mathsf{N_C}$, then $\mathcal{I}_{i+1}$ is obtained from $\mathcal{I}_i$ by adding d to $A^{\mathcal{I}_i}$,
- If $L = \exists r.D$, then $\mathcal{I}_{i+1}$ is obtained from $\mathcal{I}_i$ by adding a new domain element e and adding it to $D^{\mathcal{I}_i}$, as well as adding $\langle d, e\rangle$ to $r^{\mathcal{I}_i}$.
- If $L = \forall r.D$, then $\mathcal{I}_{i+1}$ is obtained from $\mathcal{I}_i$ by adding each r-successor of d to $D^{\mathcal{I}_i}$.

We observe that those steps ensure that $d \in C^{\mathcal{I}_i}$. We argue later that the case where $L = \neg A$ is not possible, so that those cases are exhaustive. But before that, we would like to show the following *monotonicity property* of our construction: if d is the domain element selected for creating $\mathcal{I}_i$, and for any clause $C' \in \mathbf{N}^*$ s.t. $C' \prec C$, we have $d \in (C')^{\mathcal{I}_i}$, then we also have $d \in (C')^{\mathcal{I}_j}$ for all $j > i$. This is clearly the case if d satisfies a literal of the form B or $\exists r.B$ in C', since

our construction does not remove elements from the interpretations of concepts and roles. If d satisfies a literal of the form $\forall r.D$ in C', we observe that our ordering ensures that in no clause larger than C', a literal of the form $\exists r.D'$ can be maximal, so that no further successors can be added to d. Finally, if d satisfies in C' a literal of the form $\neg A$, we first observe that our ordering ensures that A cannot be maximal in a clause larger than C', since $A \prec \neg A$. Moreover, since in each step, we select the *oldest* domain element that still has unsatisfied clauses in $\mathbf{N}^*$, it is not possible that a predecessor of d gets selected for some $j > i$, since any predecessor would be older. Consequently, d cannot be added to A due to a literal of the form $\forall r.D$ for any subsequent interpretation.

We now show that the case where $L = \neg A$ is not possible: since $d \notin C^{\mathcal{I}_i}$, $L = \neg A$ would mean that $d \in A^{\mathcal{I}_i}$. Assume such a case is possible, and let i be the smallest index for which this happens. d must have been added to $A^{\mathcal{I}_i}$ in an earlier iteration for one of the following reasons.

1. There is some $j < i$ such that $d \notin C_1^{\mathcal{I}_j}$, where $C_1 \in \mathbf{N}^*$, and A is the maximal literal in C_1. Rule **A1** is applicable on C_1 and C, yielding a clause C_2 that is also in $\mathbf{N}^*$. This clause is smaller than both C and C_1, since $A/\neg A$ is maximal in these clauses and does not occur in C_2. Consequently, C_2 would have been processed before $\mathcal{I}_j$ by this procedure, which would have ensured that for some $k < j$, $d \in C_2^{\mathcal{I}_k}$. By the monotonicity property of our construction, this means that $d \in C_2^{\mathcal{I}_j}$. Now C_2 is composed exactly of the literals in C/C_1 except $\neg A/A$, but $d \notin C_1^{\mathcal{I}_j}$, which means that $d \in C^{\mathcal{I}_j}$. Due to the monotonicity property of our construction, then also $d \in C^{\mathcal{I}_i}$. But this contradicts that d and C are selected to obtain $\mathcal{I}_{i+1}$.
2. d is an r-successor of another domain element and was added due to some $j < i$ and selected clause C_1 not satisfied in $\mathcal{I}_j$ in which the maximal literal is $\exists r.A$. In this case, we have $d \neq d_0$. The maximal literal in C is then of the form $\neg A$, with A under an existential role restriction in $\mathbf{N}$. By our ordering, this means that all other literals in C must be negated (Condition 2) and their concept names occur in $\mathbf{N}$ under existential role restrictions (Condition 1). We can indeed conclude from this that $\neg A$ is the only literal in C. Otherwise, there would be another literal $\neg B$ in C, where B does not occur in a value restriction. (Recall that by our assumptions, if B occurs positively under an existential role restriction, it cannot occur positively in a different way). Since $d \notin C^{\mathcal{I}_i}$, also $d \in B^{\mathcal{I}_i}$, but there is no step in the construction that would add an existing domain element to a concept occurring under an existential role restriction.
   We obtain that $\neg A \in \mathbf{N}^*$. But then, the **r2**-rule applies on this clause and C_1 for the case of $n = 0$, resulting in a clause C_2 that is obtained from C_1 by removing its maximal literal. Thus, $C_2 \prec C_1$, which means that C_2 must have been processed before C_1. This in turn means that $\mathcal{I}_j \models C_2$, and thus $\mathcal{I}_j \models C_1$, so C_1 could not have been selected in Step j.
3. d is an r-successor of another domain element e and was added to $A^{\mathcal{I}_i}$ due to some $j < i$ and clause C_1 in which the maximal literal is $\forall r.A$. As before, we can argue that all literals in C are of the form $\neg D$, with D occurring

under a role restriction, and C contains at most one literal $\neg D_\exists$, where $D_\exists$ occurs under an existential role restriction. In particular, d was created as an r-successor of e due to a clause in which $\exists r.D_\exists$ is the maximal literal, and for every $\neg D$ in C, d was added to the interpretation of D due to a clause in which the maximal literal is $\forall r.D$. We observe that one of the rules **r1** or **r2** is applicable on those clauses together with C, resulting in a clause that is smaller and consequently must have been processed before all the other clauses, making at least one of these clauses satisfied for e, and contradicting that this clause was used to add d to the interpretation of some D such that $\neg D$ occurs in C.

As a consequence, we obtain that, in each step, one clause is satisfied for one domain element for which it was not satisfied before. In the limit, we obtain that $\mathcal{I}$ satisfies all clauses in $\mathbf{N}^*$, and thus is a model of $\mathbf{N}$. □

B.2 Optimizations

From the construction in the proof of Theorem 9, we see that some clauses in $\mathbf{N}^*$ are not relevant for the refutational completeness, and can thus be discarded:

1. *Tautologies*, that is, clauses containing both A and $\neg A$ for some $A \in \mathsf{N_C}$ are always satisfied and will thus never trigger an adaptation of the current interpretation. In our implementation, tautologies are never added to the current set of clauses.
2. *Subsumed clauses*, i.e. clauses C_1 such that, for some other clause $C_2 \in \mathbf{N}^*$, $C_2 \neq C_1$ and every literal in C_2 also occurs in C_1. By our ordering, $C_2 \prec C_1$, so that our model construction will consider C_2 before it considers C_1. By the monotonicity property, satisfiying C_2 furthermore ensures that C_1 remains satisfied, and is thus never considered by the model construction. In our implementation, we use both *forward subsumption*, that is, newly derived clauses that are subsumed by previously derived clauses are not added to the current set of clauses, and *backward subsumption*, that is, after adding a new clause, we remove from the current set of clauses all clauses that are subsumed by it.

We furthermore observe that the arguments in the proof only consider the maximal literals in a clause according to the literal ordering $\prec_l$. For this reason, our method remains refutationally complete if we only perform inferences on the maximal literal.

B.3 Generating $\mathcal{ALC}$ Proofs

Fix an $\mathcal{ALC}$ ontology $\mathcal{O}$ and concepts C, D. To compute a proof for $\mathcal{O} \models C \sqsubseteq D$, we would first introduce concept names A_C, A_D for those concepts, and add the axioms $A_C \sqsubseteq C$, $D \sqsubseteq A_D$ (that is, we reduce to subsumption between concept names). As mentioned in the text, we use concept names A_{LHS}, A_{RHS} to track inferences for the proof. After normalizing, we add the clauses $A_C \sqcup A_{\text{LHS}}$ and

$\neg A_D \sqcup A_{\text{RHS}}$, which we add to the set of clauses. We make sure that A_{LHS} and A_{RHS} are minimal in the ordering used, so that the clause $A_{\text{LHS}} \sqcup A_{\text{RHS}}$ or a subclause is derived if $\mathcal{O} \models C \sqsubseteq D$ (a subclause in the cases where the ontology is inconsistent (empty clause), C is unsatisfiable, or $\mathcal{O} \models \top \sqsubseteq D$). By tracking all inferences, we obtain a proof for that entailment from $\mathbf{N}$, in which nodes are labeled with clauses. This proof still has to be transformed into a more readable DL proof for $\mathcal{O} \models C \sqsubseteq D$. For this, we

- regard clauses C as GCIs $\top \sqsubseteq C$,
- replace A_{LHS} everywhere by $\neg C$ and A_{RHS} by everywhere by D, where C and D are the concepts in the subsumption $C \sqsubseteq D$ to be proved,
- replace A_C by C and A_D by D, where C and D are the concepts of the subsumption to be proved,
- replace any concept names A under role restrictions $\mathsf{Q}r.A$, which were used during normalization to flatten expressions $\mathsf{Q}r.C$, again by C,
- add additional inferences to link the non-tautologigal leafs of the proof to axioms from the ontology (from which they were obtained during normalization),
- apply the following transformations exhaustively on each clause to make them more human-readable, where we treat $\top$ as empty conjunction and $\bot$ as empty disjunction:
  - $C \sqsubseteq \neg A \sqcup D \quad \Longrightarrow \quad C \sqcap A \sqsubseteq D$
  - $C \sqsubseteq \forall r.\bot \sqcup D \quad \Longrightarrow \quad C \sqcap \exists r.\top \sqsubseteq D$
  - $C \sqsubseteq \exists r.\neg D \sqcup E \quad \Longrightarrow \quad C \sqcap \forall r.D \sqsubseteq E$
  - $C \sqsubseteq \forall r.\neg D \sqcup E \quad \Longrightarrow \quad C \sqcap \exists r.D \sqsubseteq E$

The last step is not strictly necessary, but the final goal of the proof is to explain the inference to the user, for which we want to minimize the number of negations used. As a result of the transformation, the clauses $A_C \sqcup A_{\text{LHS}}$ and $\neg A_D \sqcup A_{\text{RHS}}$ that were added to the initial clause set get transformed into tautologies $C \sqsubseteq C$ and $D \sqsubseteq D$.

B.4 Integrating the Concrete Domain

The reasoning algorithm keeps a set $\mathbf{D}$ of currently relevant constraints. To compute all relevant implications $\alpha_1 \wedge \ldots \wedge \alpha_n \rightarrow \beta$, where $\alpha_1, \ldots, \alpha_n \in \mathbf{D}$, we use the proof procedure for the concrete domain to compute the set of possible inferences starting from $\mathbf{D}$. For each derived constraint β for which A_β occurs negatively in the set of clauses, we follow those inferences back to the constraints in $\mathbf{D}$ that were used to derive it, creating all such possible sets. This way, we ensure that all implications $\alpha_1 \wedge \ldots \wedge \alpha_n \rightarrow \beta$, where $\{\alpha_1, \ldots, \alpha_n\} \subseteq \mathbf{D}$ is subset-minimal, are discovered, so that we can add the corresponding clauses to the current set of clauses. The procedure might also create some implications where the set of constraints on the left-hand side is not subset-minimal. However, those clauses are immediately removed due to forward-subsumption deletion.

Theorem 7. *Let $\mathcal{O}$ be an $\mathcal{ALC}[\mathcal{D}]$ ontology and $\mathbf{N}$ be 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 $\mathbf{N}$ iff $\mathcal{O} \models C \sqsubseteq D$.*

Proof. The complexity of the method does not change, since we only add exponentially many new clauses. Soundness follows from the fact that the added clauses all correspond to valid entailments for the original ontology $\mathcal{O}$ if we replace the fresh concept names again by the corresponding concrete constraints. For completeness, we use the construction as in the proof of Theorem 9 to construct a model $\mathcal{I}$ for $\mathbf{N}^*$, the final set of clauses, in case $\bot \notin \mathbf{N}^*$. $\mathcal{I}$ is a model for $\mathcal{O}^{-\mathcal{D}}$, but we still have to modify it into a model of $\mathcal{O}$. We argue that we can do that by ensuring that for every $d \in \Delta^{\mathcal{I}}$:

1. if $d \in A_D^{\mathcal{I}}$, then $d \in D^{\mathcal{I}}$,
2. if $d \notin A_D^{\mathcal{I}}$, and there is no clause $t : C \in \mathbf{N}^*$ s.t. t can be interpreted by d (i.e., either $t = x$, or $t = a$ and $d = a^{\mathcal{I}}$), the maximal literal of C is $\neg A_D$, and $d \notin L^{\mathcal{I}}$ for any other literal in C, then $d \notin D^{\mathcal{I}}$.

If we can modify $\mathcal{I}$ like that, then we obtain a model of $\mathcal{O}$.

For a domain element $d \in \Delta^{\mathcal{I}}$, we collect the sets $\mathbf{D}$ and $\overline{\mathbf{D}}$, where $\mathbf{D} = \{D \in \mathcal{C}(\mathcal{O}) \mid d \in A_D^{\mathcal{I}}\}$ and $\overline{\mathbf{D}}$ contains all the $D \in \mathcal{C}(\mathcal{O})$ for which A_D satisfies the second condition above.

We first observe that $\mathcal{D} \not\models \bigwedge \mathbf{D} \to \bot$, since the concept names corresponding to the constraints in $\mathbf{D}$ must occur as maximal literals in some clauses in $\mathbf{N}^*$, and we would thus have added a clause $x : \bigsqcup_{D \in \mathbf{D}'} \neg A_D$ for some subset $\mathbf{D}' \subseteq \mathbf{D}$ if $\mathbf{D}$ was inconsistent. This clause would not be satisfied by $\mathcal{I}$, contradicting that $\mathcal{I} \models \mathbf{N}^*$. We can thus extend $\mathcal{I}$ in such a way that, for each $D \in \mathbf{D}$, $d \in D^{\mathcal{I}'}$. It remains to show that we can also ensure that for every $D \in \overline{\mathbf{D}}$, $d \notin D^{\mathcal{I}'}$. We first observe that for no $D \in \overline{\mathbf{D}}$, $\mathcal{D} \models \bigwedge \mathbf{D} \to D$. This is because $\overline{\mathbf{D}}$ was carefully chosen to ensure that if $\mathcal{D} \models \bigwedge \mathbf{D} \to D$, there would be some subset $\mathbf{D}' \subseteq \mathbf{D}$ for which we would have added the clause $x : \bigsqcup_{D' \in \mathbf{D}'} \neg A_{D'} \sqcup A_D$. Since d does not satisfy the concept, this would contradict that $\mathcal{I}$ is a model of $\mathbf{N}^*$. Because $\mathcal{D}$ is convex, it follows furthermore that $\mathcal{D} \not\models \bigwedge \mathbf{D} \to \bigvee \overline{\mathbf{D}}$. Consequently, we can find an assignment of the concrete features that ensures that $d \in D^{\mathcal{I}'}$ for all $D \in \mathbf{D}$ and $d \notin D^{\mathcal{I}'}$ for all $D \in \overline{\mathbf{D}}$. It follows that we can extend $\mathcal{I}$ to a model of $\mathcal{O}$ as required. □

To produce a combined proof in $\mathcal{ALC}[\mathcal{D}]$, we again follow the approach described in Section 4.2, but for now we always use the left-hand side $A_{\alpha_1} \sqcap \cdots \sqcap A_{\alpha_n}$ as the context for the CD proof of $\alpha_1 \wedge \cdots \wedge \alpha_n \to \beta$, since it is not trivial to find other meaningful DL contexts for proofs generated from the calculus in Fig. 3.

B.5 The Complexity of Finding Good Proofs

To investigate the complexity of our approaches and prove Theorem 8, we use the formal framework from [1,2], which we shortly introduce in the following. A *derivation structure* for an entailment $\mathcal{T} \models \eta$ in a logic $\mathcal{L}$ is a directed, labeled hypergraph (V, E, ℓ) where

1. vertices are labeled with $\mathcal{L}$-sentences,
2. every leaf is labeled by an axiom from $\mathcal{T}$, and
3. every hyperedge $(S, d) \in E$ is an inference satisfying $\{\ell(v) \mid v \in S\} \models \ell(d)$.

Here, a *leaf* is a node $v \in V$ without an incoming hyperedge $(S, v) \in E$, and a *sink* $v \in V$ has no outgoing hyperedges $(S, d) \in E$ with $v \in S$. A *proof* for $\mathcal{T} \models \eta$ is such a derivation structure that, additionally,

4. is tree-shaped, i.e. has no cycles in the relation $\{(s, d) \mid (S, d) \in E,\ s \in S\}$,
5. has a unique sink labeled by the final conclusion η, and
6. has no two hyperedges $(S, v), (S', v') \in E$ with $\ell(v) = \ell(v')$.

As in [1,2], we consider a so-called *deriver* $\mathfrak{D}$, which produces derivation structures $\mathfrak{D}(\mathcal{T}, \eta)$ that contain all inference steps relevant for a proof of $\mathcal{T} \models \eta$ (but they can encompass many possible ways of deriving η). We are interested in finding a proof that can be homomorphically mapped into $\mathfrak{D}(\mathcal{T}, \eta)$, and whose *size* (number of vertices) is below a given threshold (a "small proof"). Reasoners for $\mathcal{EL}_\bot$, such as Elk, produce derivation structures of polynomial size, and the problem of finding small proofs in such structures is NP-complete [1]. For derivation structures that are of exponential size in general, such as for $\mathcal{ALC}$, this problem is NExpTime-complete [1]. If we replace *size* by *recursive* measures such as *depth* (the length of the longest path from the sink to a leaf), the complexity drops to P and ExpTime, respectively [2].

To determine the complexity of finding small proofs in $\mathcal{EL}_\bot[\mathcal{D}]$ and $\mathcal{ALC}[\mathcal{D}]$, we additionally need to consider the size of the derivation structures for $\mathcal{D}_{\mathbb{Q},lin}$ and $\mathcal{D}_{\mathbb{Q},diff}$, which are then integrated into the pure DL proofs. For $\mathcal{D}_{\mathbb{Q},diff}$, Theorem 5 shows that the derivation structures constructed by Algorithm 2 are always of polynomial size, which does not change the complexity of finding proofs compared to the case of DLs without concrete domains. For $\mathcal{D}_{\mathbb{Q},lin}$, however, derivation structures can be of exponential size: Although Gaussian elimination produces at most quadratically many inference steps in the number of variables that occur in the constraints, there are exponentially many possible orders in which the variables could be eliminated and different choices of constraints to use for eliminating a variable, each of which yields a different proof. To alleviate this problem in our implementation, we normalize all equations after each elimination step, by reducing the coefficient of the leading variable (according to a fixed variable order) to 1, and adjusting the other coefficients accordingly. For example, the elimination step

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

from the previous example would become

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

Nevertheless, the overall size of the derivation structure stays exponential in the worst case.

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).*

Proof. In the cases involving $\mathcal{D}_{\mathbb{Q},diff}$ or $\mathcal{ALC}[\mathcal{D}]$, we obtain polynomial (exponential) structures for $\mathcal{EL}_\bot[\mathcal{D}]$ ($\mathcal{ALC}[\mathcal{D}]$) entailment problems by transforming the concrete domain derivation structures and integrating them into the classical DL derivation structures as described in Lemma 6. In $\mathcal{EL}_\bot[\mathcal{D}]$, our method considers only polynomially many $\mathcal{D}$-implications, while for $\mathcal{ALC}[\mathcal{D}]$ we obtain exponentially many $\mathcal{D}$-derivation structures (of polynomial or exponential size), which does not affect the exponential size of the derivation structures for $\mathcal{ALC}$. Hence, we can apply the classical algorithms for finding proofs of a given maximal size or depth in the combined structures [1,2].

For $\mathcal{EL}_\bot[\mathcal{D}_{\mathbb{Q},lin}]$, the idea is to guess a substructure of the combined derivation structure in polynomial time, and then verify that it is indeed a proof of the required size. Since the $\mathcal{EL}_\bot$-parts of the derivation structure are of polynomial size, we can guess those parts in NP. For all guessed axioms $\bigsqcap_{D \in \mathbf{D}_C} A_D \sqsubseteq A_E$, we then need to guess a corresponding proof of $\bigwedge \mathbf{D}_C \rightarrow E$ in $\mathcal{D}_{\mathbb{Q},lin}$. However, we know that such a proof needs at most polynomially many variable elimination steps (at most one for each variable in each involved constraint), which correspond to inference steps. Hence, we can guess in polynomial time a variable elimination order and, for each constraint α and variable x, a constraint that is used to eliminate x from α. □

C Implementation and Experiments

We implemented the algorithms described above and evaluated their performance and the produced proofs on a series of benchmarks. The implementation uses the Java-based OWL API 4 to interact with DL ontologies, but uses new data structures for representing concrete domains. Although there is a proposal for extending OWL with concrete domain predicates of arities larger than 1,[7] this is not part of the OWL 2 standard.[8] To extract proofs from the collections of inference steps produced by our concrete domain reasoning algorithms, we used a Dijkstra-like algorithm that minimizes the size of the produced proof [2]. For efficiency reasons, for proofs in $\mathcal{D}_{\mathbb{Q},lin}$, we fix a variable order for the Gaussian elimination steps, instead of considering all possible orders in which variables could be eliminated.

Returning to Example 3, we can split it into two tasks to demonstrate proofs in both concrete domains, where we have added information on the status of our current patient using the GCIs $\mathsf{CurrentPatient} \sqsubseteq [\mathsf{age} = 42]$, $\mathsf{CurrentPatient} \sqsubseteq [\mathsf{hr} = 173]$, and $\mathsf{CurrentPatient} \sqsubseteq [\mathsf{pp} = 65]$. Resulting proofs of $\mathsf{ICUpatient} \sqsubseteq \mathsf{NeedAttention}$ are shown in Fig. 6 and Fig. 7.

[7] `https://www.w3.org/2007/OWL/wiki/Data_Range_Extension:_Linear_Equations`
[8] `https://www.w3.org/TR/owl2-overview/`

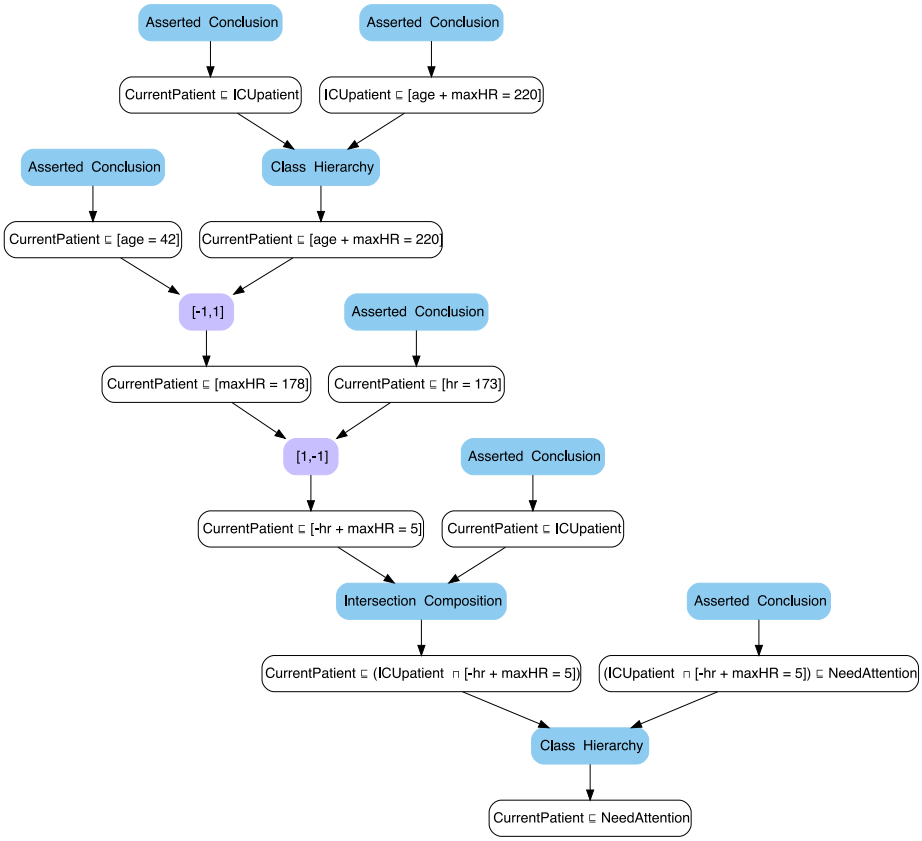

Fig. 6. Showing $\mathsf{CurrentPatient} \sqsubseteq \mathsf{NeedAttention}$ using $\mathcal{D}_{\mathbb{Q},lin}$

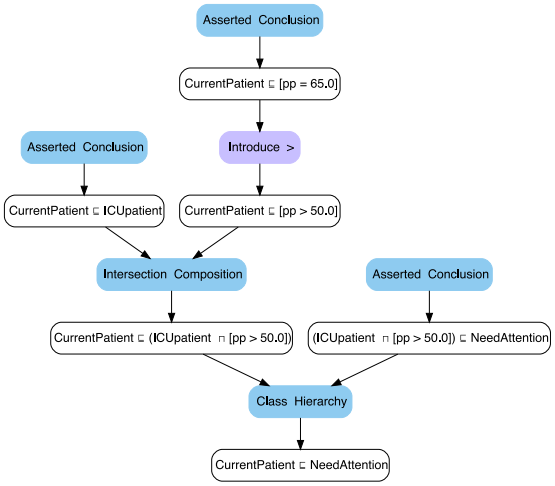

Fig. 7. Showing $\mathsf{CurrentPatient} \sqsubseteq \mathsf{NeedAttention}$ using $\mathcal{D}_{\mathbb{Q},diff}$

C.1 Benchmarks

In the following, we describe several ontologies that we developed to evaluate our implementation. Unfortunately, existing reasoning tasks for DLs with concrete domains, e.g. from Racer,[9] are not expressive enough to test our algorithms; the CD values are used only as constants and do not influence the reasoning. Some of our benchmarks are scalable in the sense that they are based on similar ontologies, but one can increase their size, e.g. by increasing the number or size of axioms or constraints. All benchmarks are formulated in $\mathcal{EL}_\bot[\mathcal{D}]$.

Simple benchmarks. For $\mathcal{D}_{\mathbb{Q},lin}$, there are two basic demonstration examples, *Drones* and *Coffee*. The main task in *Drones* is to derive the fraction of impaired sensors and propellers of a drone.

In *Coffee*, for different types of coffee such as cappuccino, ristretto, macchiato, etc., we define the proportions of components such as espresso, steamed milk, foam, etc. Consequently, we can identify a coffee drink based on the amounts of its components given in some unit like ml or oz.

Scalable benchmarks. For testing the system behavior on inputs of increasing size, we provide four benchmarks, two for $\mathcal{D}_{\mathbb{Q},lin}$ and two for $\mathcal{D}_{\mathbb{Q},diff}$.

In $Diet(n)$, given a person's daily consumption as a list of n products and their calories from fat, protein, and carbs, we check constraints about the consumption, such as "full fat", "full protein", "full carbs", "well-balanced" (55% carbs, 20% protein, 25% fat), "lower carb" (45% carbs, 25% protein, 30% fat), and "lower carb and fat" (45% carbs, 30% protein and 25% fat), expressed in $\mathcal{D}_{\mathbb{Q},lin}$. The parameter n describes the size of the linear constraints.

[9] `https://github.com/ha-mo-we/Racer`

Table 1. Results of the experiments with the $\mathcal{EL}_\bot[\mathcal{D}]$ algorithms.

Name	Axioms	Constraints	Variables	Problem Size	Time	Proof Time	#Fin/Σ
Coffee	49	21	2.3	146	329	109	1/1
Drones	93	11	2	254	195	100	1/1
Diet	26–194	15–99	2–40	81–865	61–321	45–2659	8/8
Artificial	9–24	2–13	3–5	25–144	17–347	18–	4/6
D-Sbj	55–204	20–85	1.5	191–1101	220–686	166–482	8/8
D-Obj	122–427	24–76	1.5	427–2129	315–1437	417–2099	8/8

To scale both the DL and CD parts, we created the benchmark *Artificial*(n) over $\mathcal{D}_{\mathbb{Q},lin}$. With increased n, the number of intermediate concepts in the concept hierarchy and in proofs also increases, i.e. $A \sqsubseteq C_0 \sqsubseteq \cdots \sqsubseteq C_{n-1} \sqsubseteq B$. Moreover, to show each step, the concrete domain reasoner has to derive a linear constraint from n given constraints. Thus, the size of a proof for $A \sqsubseteq B$ grows quadratically in n.

For $\mathcal{D}_{\mathbb{Q},diff}$, there are *D-Sbj*$(n)$ and *D-Obj*(n). In *D-Sbj*(n), there is one main actor, which is a drone. The parameter n quantifies the number of other objects in the world. The reasoner needs to show that all objects are at a "safe" distance from the drone.

The benchmark *D-Obj*(n) is somehow orthogonal to *D-Sbj*(n). The world contains n drones and exactly 3 other objects, e.g. humans or trees. Now the distances between all drones and objects are taken into consideration. Similarly to the subjective version, the reasoner needs to find out whether all objects are at "safe" distances.

Experiments. Our findings for the $\mathcal{EL}_\bot[\mathcal{D}]$ algorithms are summarized in Tables 1 and 2. The first two benchmarks consist of a single reasoning problem each, the remaining four represent series of small to medium-sized ontologies, for which we report ranges in each column. For the scalable benchmarks, we consider only the instances which terminated and, for each size, computed average times over three randomly generated instances of that size. The underlined benchmarks are in $\mathcal{D}_{\mathbb{Q},diff}$, the rest in $\mathcal{D}_{\mathbb{Q},lin}$. Columns 2–4 show the number of axioms, number of constraints, and average number of variables per constraint in the ontology, respectively. Column 5 aggregates the number of occurrences of any name (concept, role or feature) in the ontology. Columns 6–7 list the time (in ms) required for classification and proof generation, respectively, "Σ" denotes the total number of instances, and "#Fin" denotes the number of instances for which proof generation finished before a timeout of 3 min.

Table 2. Supplemental experiment data for the $\mathcal{EL}_\bot[\mathcal{D}]$ algorithms (cf. Table 1). "%DL" denotes the fraction of the reasoning time that was used by the calls to Elk. "%Incr" denotes the fraction of Elk reasoning time that would have been required for a single call to Elk on the final saturated ontology $\mathcal{O}'$. The last two columns describe the computed proofs in terms of their tree size and "%CD", the average fraction of the steps of the computed proof that are due to concrete domain inference steps. For the scalable benchmarks, we report either ranges or averages with standard deviations (*sd*) in each column.

Name	%DL(*sd*)	%Incr(*sd*)	Proof Size	%CD(*sd*)
Coffee	41	20	18	11
Drones	83	51	7	11
Diet	23(7)	58(6)	22–166	37(10)
Artificial	75(15)	78(16)	10–51	18(5)
D-Sbj	52(2)	76(8)	45–584	6(0)
D-Obj	63(2)	86(6)	184–1232	8(0)