EPiC Series in Computing
Volume 73, 2020, Pages 32–67
LPAR23. LPAR-23: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning

Finding Small Proofs for Description Logic Entailments: Theory and Practice

Christian Alrabbaa, Franz Baader, Stefan Borgwardt, Patrick Koopmann, and Alisa Kovtunova

Institute of Theoretical Computer Science, TU Dresden, Germany

Abstract

Logic-based approaches to AI have the advantage that their behaviour can in principle be explained by providing their users with proofs for the derived consequences. However, if such proofs get very large, then it may be hard to understand a consequence even if the individual derivation steps are easy to comprehend. This motivates our interest in finding small proofs for Description Logic (DL) entailments. Instead of concentrating on a specific DL and proof calculus for this DL, we introduce a general framework in which proofs are represented as labeled, directed hypergraphs, where each hyperedge corresponds to a single sound derivation step. On the theoretical side, we investigate the complexity of deciding whether a certain consequence has a proof of size at most n along the following orthogonal dimensions: (i) the underlying proof system is polynomial or exponential; (ii) proofs may or may not reuse already derived consequences; and (iii) the number n is represented in unary or binary. We have determined the exact worst-case complexity of this decision problem for all but one of the possible combinations of these options. On the practical side, we have developed and implemented an approach for generating proofs for expressive DLs based on a non-standard reasoning task called forgetting. We have evaluated this approach on a set of realistic ontologies and compared the obtained proofs with proofs generated by the DL reasoner ELK, finding that forgetting-based proofs are often better w.r.t. different measures of proof complexity.

1 Introduction

Embedded or cyber-physical systems that interact autonomously with the real world, or with users they are supposed to support, must continuously make decisions based on sensor data, user input, knowledge they have acquired during runtime as well as knowledge provided during design-time. To make the behavior of such systems comprehensible, they need to be able to explain their decisions to the user or, after something has gone wrong, to an accident investigator. While systems that use Machine Learning (ML) to interpret sensor data are very fast and usually quite accurate, their decisions are notoriously hard to explain, though huge efforts are currently being made to overcome this problem [1]. In contrast, decisions made by reasoning about symbolically represented knowledge are in principle easy to explain. In particular, if the knowledge is represented in (some fragment of) first-order (FO) logic, and a decision is made based on the result of an FO reasoning process, then one can in principle use a formal proof in

an appropriate calculus to explain an entailment. In practice, however, things are not so rosy also in the symbolic setting. On the one hand, proofs may be very large, and thus it may be hard to comprehend why the overall entailment holds even if each single derivation step is easy to follow. On the other hand, single proof steps may also be hard to understand, in particular for users that are not experts in logic. The problem of explaining why a certain consequence follows from a given set of sentences has been considered for full FO automated theorem proving since at least four decades. Since the resolution proofs generated by high-performance automated theorem provers are not appropriate for human consumption, approaches transforming them into proofs in more human-oriented calculi (such as Gentzen's natural deduction calculus [29]) have been developed. To alleviate the tedious task of following a huge number of these steps, abstractions of proofs that use definitions, lemmas, and more abstract deduction rules have been investigated [21]. A more detailed description of the huge body of research in this area is beyond the scope of this paper.

Here we concentrate on explaining the results of DL reasoning using formal proofs, and in particular on how complex it is to find small proofs. The first work on explaining DL entailments is probably the PhD thesis of McGuinness [31], where the results obtained by the structural subsumption algorithm of the CLASSIC system [11] are translated into proofs in a formal calculus. The thesis also investigates how to create shorter, better understandable proofs by pruning away unimportant parts. In [10], proofs of subsumptions generated by a tableau-based system are translated into sequent proofs. The authors then investigate how to make the sequent proofs shorter. More recent work on explaining DL entailment was often focused on computing so-called justifications, i.e., minimal subsets of the knowledge base (KB) from which the consequence in question follows (see, e.g., [7, 18, 38]). The basic assumption is here that, whereas KBs may be very large and have many consequences, a single consequence often follows from a small subset of the KB by an easy derivation. While this is true in certain applications [9], in general it may be quite hard for a user to see without help why a consequence follows from a given justification [20]. On the one hand, this has triggered research into assessing the complexity of a justification, i.e., how hard it is to derive the given consequence from the justification [19, 32]. On the other hand, it has rekindled the interest in generating proofs appropriate for human consumption. For example, the explanation plugin for the ontology editor Protégé described in [24] cannot only produce justfications, but can also display proofs, provided that proofs of an appropriate form are returned by the employed reasoner, an assumption that is, e.g., satisfied by the reasoner ELK [25]. While these proofs are represented in the formal syntax employed by Protégé, the work reported in [32, 36, 37] uses ontology verbalization techniques to translate proofs into natural language text.

Since most of the previous work on generating proofs for DL entailments emphasizes the importance of small proofs, we investigate in this paper how hard it is to find such proofs. Instead of concentrating on a specific DL and proof calculus or reasoner for this DL, we introduce a general framework in which proofs are represented as labeled, directed hypergraphs whose hyperedges correspond to single sound derivation steps. To be more precise, we assume that a reasoner (called *deriver* in this paper, to distinguish it from an actual implemented system) generates a so-called derivation structure, which consists of possible proof steps, from which actual proofs can be derived. For example, if we consider the consequence-based reasoning approaches for the DLs $\mathcal{EL}$ and $\mathcal{ELI}$ described in [5], then a derivation structure for a given KB and consequence consists of the (finitely many) instantiated classification rules. The $\mathcal{EL}$ reasoner ELK actually returns such a derivation structure, but this structure only contains the rule instances that have actually been used during the reasoning process.

On the theoretical side, we investigate the complexity of deciding whether a certain conse-

quence has a proof of size at most n along three orthogonal dimensions. First, we distinguish between derivers that produce derivation structures of polynomial or exponential size. For example, there is a polynomial deriver for $\mathcal{EL}$ (e.g., the one implemented by ELK), whereas $\mathcal{ELI}$ has an exponential, but no polynomial deriver. Second, we distinguish between arbitrary proofs and tree-shaped proofs, which may not reuse already derived consequences, but must re-derive them each time they are needed. Finally, we distinguish between unary and binary coding of the number n, which bounds the size of the proof. Tables summarizing the complexity results shown in this paper can be found in Section 4 (Table 1 and 2). We see that, for general proofs, the above decision problem is NP-complete even for polynomial derivers and unary coding of numbers. For exponential derivers, the complexity depends on the coding of the number n: it is NP-complete for unary coding, but NExpTime-complete for binary coding. Interestingly, for tree-shaped proofs the complexity is considerably lower, which is due to the fact that we can use a Dijkstra-like greedy algorithm to compute minimal tree-shaped proofs.

On the practical side, we compare the time it requires to extract a smallest proof vs. a smallest tree-shaped proof from derivation structures extracted from ELK, and our findings match the theoretical complexity results, i.e., finding small tree-shaped proofs is easier. Moreover, we have developed and implemented an approach for generating proofs for expressive DLs based on *forgetting* [26, 30]. We have evaluated this approach on a set of realistic ontologies and compared the proofs obtained using the forgetting tools LETHE [27] and FAME [41] with proofs generated by the DL reasoner ELK, finding that forgetting-based proofs are often better w.r.t. different measures of proof complexity. In addition to measuring the size of proofs, we also consider measures obtained by applying the justification complexity measure of [19] to the proof steps.

To ease presentation, some supplementary material can be found in the two appendices. Appendix A ("Theory") contains the missing proofs from the main part of this paper, while Appendix B ("Practice") provides additional examples of automatically generated proofs.

2 Preliminaries

Most of our theoretical discussion applies to arbitrary *logics* $\mathcal{L} = (S_\mathcal{L}, \models_\mathcal{L})$, which consist of a set $S_\mathcal{L}$ of *$\mathcal{L}$-sentences* and a *consequence relation* $\models_\mathcal{L} \subseteq P(S_\mathcal{L}) \times S_\mathcal{L}$ between *$\mathcal{L}$-theories*, i.e., subsets of $\mathcal{L}$-sentences, and single $\mathcal{L}$-sentences. We assume that $\models_\mathcal{L}$ has a model-based definition, i.e., for some definition of "model", $\mathcal{T} \models_\mathcal{L} \alpha$ holds iff every model of all elements in $\mathcal{T}$ is also a model of α. We also assume that the *size* $|\alpha|$ of an $\mathcal{L}$-sentence α is defined in some way, e.g. by the number of symbols in α. Since $\mathcal{L}$ is usually fixed, we drop the prefix "$\mathcal{L}$-" from now on. For example, $\mathcal{L}$ could be *first-order logic*. However, we are mainly interested in proofs for DLs, which can be seen as decidable fragments of first-order logic [4]. In particular, we use specific DLs to show our hardness results.

The syntax of DLs is based on disjoint, countably infinite sets $\mathsf{N_C}$ and $\mathsf{N_R}$ of *concept names* $A, B, \ldots$ and *role names* $r, s, \ldots$, respectively. Sentences of the DL $\mathcal{EL}$, called *general concept inclusions (GCIs)*, are of the form $C \sqsubseteq D$, where C and D are *$\mathcal{EL}$-concepts*, which are built from concept names by applying the constructors $\top$ (*top*), $C \sqcap D$ (*conjunction*), and $\exists r.C$ (*existential restriction* for a role name r). The DL $\mathcal{ALC}$ extends $\mathcal{EL}$ by the constructors $\bot$ (*bottom*), $C \sqcup D$ (*disjunction*), $\forall r.C$ (*value restriction*), and $\neg C$ (*negation*). Both $\mathcal{EL}$ and $\mathcal{ALC}$ can be extended by a new kind of sentences $r \sqsubseteq s$, called *role inclusions*, where r and s are role names, and by additional constructors such as *nominals* or *inverse roles*. These extensions are denoted by the additional letters $\mathcal{H}$, $\mathcal{O}$, and $\mathcal{I}$, respectively, appended to the name of the logic, resulting in DLs such as $\mathcal{ALCH}$, $\mathcal{ALCOI}$ or $\mathcal{ELI}$. In DLs, finite theories are called

$$\mathsf{R}_0\ \frac{}{C \sqsubseteq C} \qquad \mathsf{R}_\top\ \frac{}{C \sqsubseteq \top} \qquad \mathsf{R}_\sqsubseteq\ \frac{C \sqsubseteq D \quad D \sqsubseteq E}{C \sqsubseteq E} \qquad \mathsf{R}^-_{\sqcap,1}\ \frac{C \sqsubseteq D \sqcap E}{C \sqsubseteq D}$$

$$\mathsf{R}^-_{\sqcap,2}\ \frac{C \sqsubseteq D \sqcap E}{C \sqsubseteq E} \qquad \mathsf{R}^+_{\sqcap}\ \frac{C \sqsubseteq D \quad C \sqsubseteq E}{C \sqsubseteq D \sqcap E} \qquad \mathsf{R}_\exists\ \frac{C \sqsubseteq \exists r.D \quad D \sqsubseteq E}{C \sqsubseteq \exists r.E}$$

Figure 1: The inference rules for $\mathcal{EL}$ used in ELK.

$$\dfrac{A \sqsubseteq B \qquad \dfrac{\dfrac{}{B \sqsubseteq B} \qquad B \sqsubseteq \exists r.A}{B \sqsubseteq B \sqcap \exists r.A}}{A \sqsubseteq B \sqcap \exists r.A}$$

Figure 2: A tree-shaped proof in $\mathcal{EL}$

TBoxes or *ontologies*. We use $\mathsf{sig}(X)$ to denote the *signature* of some ontology or sentence X, i.e., the set of concept and role names occurring in X.

The semantics of DLs is based on first-order interpretations; for details, see [4]. In Figure 1, we depict a simplified version of the inference rules for $\mathcal{EL}$ from [25]. For example, $\{A \sqsubseteq \exists r.B,\ B \sqsubseteq C,\ \exists r.C \sqsubseteq D\} \models A \sqsubseteq D$ is a valid inference in $\mathcal{EL}$. Deciding consequences in $\mathcal{EL}$ is P-complete [2], and in $\mathcal{ELI}$ and $\mathcal{ALC}$ it is ExpTime-complete [3, 35]. Given a TBox $\mathcal{T}$ and sentence α, a *justification* for α in $\mathcal{T}$ is a minimal subset $\mathcal{J} \subseteq \mathcal{T}$ such that $\mathcal{J} \models \alpha$. Already for $\mathcal{EL}$-ontologies, finding a single justification is possible in polynomial time, but there may be exponentially many justifications, and furthermore finding a justification of size $\leq n$ is NP-complete [8].

3 Proofs, Derivers and Derivation Structures

While justifications are a popular tool for pinpointing the reasons for an entailment in an ontology, they do not provide deeper information on the reasoning behind the entailment. In contrast, consequence-based reasoners such as Cel [6] and Elk [25] derive new consequences using a fixed set of inference rules, and hence could provide proofs like in Figure 2, which uses the inference rules from Figure 1. Of course, the precise structure of a proof depends on the inference rules that are allowed to draw conclusions. Moreover, provers usually output a consequence only once, even if it is used in several inference steps. To be able to study abstract properties of such proofs, we view proofs as directed hypergraphs, in which hyperedges represent inference steps that connect a set of sentences (the premises) to another sentence (the conclusion). In the following, we fix a logic $\mathcal{L}$.

Definition 1 (Hypergraph). *A* (directed, labelled) hypergraph *[33] is a triple* $H = (V, E, \ell)$, *where*

- V *is a finite set of* vertices,
- E *is a set of* hyperedges (S, d), *where* $S \subseteq V$ *and* $d \in V$, *and*
- $\ell \colon V \to S_{\mathcal{L}}$ *is a* labelling function *that assigns sentences to vertices.*

The *size* of H, denoted $|H|$, is measured by the size of the labels of its hyperedges:

$$|H| := \sum_{(S,d) \in E} \Big(|\ell(d)| + \sum_{v \in S} |\ell(v)| \Big).$$

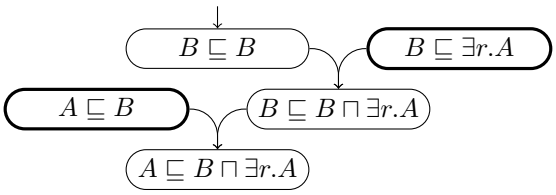

Figure 3: A derivation structure (which is a proof)

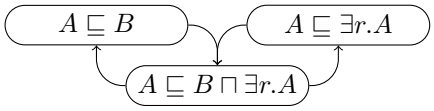

Figure 4: A cyclic derivation structure

A vertex $v \in V$ is called a *leaf* if it has no incoming hyperedges, i.e., there is no $(S, v) \in E$, and v is a *sink* if it has no outgoing hyperedges, i.e., there is no $(S, d) \in E$ such that $v \in S$.

Definition 2 (Derivation Structure)**.** *A* derivation structure $\mathcal{D} = (V, E, \ell)$ *over a finite theory* $\mathcal{T}$ *is a hypergraph, where*

- $\mathcal{D}$ *is* grounded*, i.e., every leaf* v *in* $\mathcal{D}$ *is labeled by* $\ell(v) \in \mathcal{T}$*; and*
- $\mathcal{D}$ *is* sound*, i.e., for every* $(S, d) \in E$*, the entailment* $\{\ell(s) \mid s \in S\} \models \ell(d)$ *holds.*

A derivation structure $\mathcal{D}' = (V', E', \ell')$ *is called a* substructure *of* $\mathcal{D} = (V, E, \ell)$ *if* $V' \subseteq V$*,* $E' \subseteq E$ *and* $\ell' = \ell|_{V'}$*. In this case, we also say that* $\mathcal{D}$ contains $\mathcal{D}'$*.*

In such a structure, the hyperedges are called *inference steps*. There can be inference steps of the form $(\emptyset, v)$ if v is labeled by a tautology, for example in Figure 3, where the leafs are marked with a thick border and labeled by the $\mathcal{EL}$ sentences from $\mathcal{T} = \{A \sqsubseteq B, B \sqsubseteq \exists r.A\}$ and the hyperedges represent valid $\mathcal{EL}$ entailments, in particular $\emptyset \models \{B \sqsubseteq B\}$. This hypergraph correponds to the tree-shaped proof in Figure 2. On the other hand, Figure 4 depicts a derivation structure that does not correspond to a proof since it contains cyclic inferences.

Definition 3 (Cycle, Tree)**.** *Given a hypergraph* $H = (V, E, \ell)$ *and* $s, t \in V$*, a* path P *of length* q *in* H *from* s *to* t *is a sequence of vertices and hyperedges*

$$P = (d_0, (S_1, d_1), d_1, (S_2, d_2), \ldots, d_{q-1}, (S_q, d_q), d_q),$$

where $d_0 = s$*,* $d_q = t$*, and* $d_{j-1} \in S_j$ *for all* j*,* $1 \leq j \leq q$*. If there is such a path in* H*, we say that* t *is* reachable *from* s *in* H*. If* $t = s$*, then* P *is called a* cycle*. A hypergraph is* acyclic *if it does not contain a cycle.*

A tree $H = (V, E, \ell)$ *with* root $t \in V$ *is a hypergraph in which* t *is a sink and is reachable from every vertex* $v \in V \setminus \{t\}$ *by exactly one path.*

In particular, the root is the only sink in a tree, and all trees are acyclic. We can now define proofs as special derivation structures that derive a goal sentence.

Definition 4 (Proof)**.** *Given a sentence* η *and a finite theory* $\mathcal{T}$*, a* proof for $\mathcal{T} \models \eta$ *is a derivation structure* $\mathcal{P} = (V, E, \ell)$ *over* $\mathcal{T}$ *that*

- *contains exactly one sink* $v_\eta \in V$*, which is labelled by* η*, and*
- *is acyclic.*

A tree proof *is a proof that is a tree.*

$$\dfrac{A \sqsubseteq B \qquad \dfrac{A \sqsubseteq B \qquad B \sqsubseteq \exists r.A}{A \sqsubseteq \exists r.A}}{A \sqsubseteq B \sqcap \exists r.A}$$

Figure 5: A tree-shaped proof

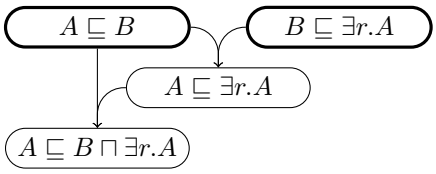

Figure 6: A proof with a reused vertex

For example, the derivation structure in Figure 3 is a proof according to this definition, but the one in Figure 4 is not. Figures 5 and 6 depict two more ways to prove $A \sqsubseteq B \sqcap \exists r.A$ from $\mathcal{T} = \{A \sqsubseteq B,\ B \sqsubseteq \exists r.A\}$ (following the inference rules in Figure 1). The first is presented in classical notation as a tree and the second one as a hypergraph without sentence label repetition. Both of them can be seen as proofs in the sense of Definition 4 and they use the same inference steps, but have different numbers of vertices.

To compare these different kinds of proofs, we recall the notion of homomorphism.

Definition 5 (Homomorphism)**.** *Let $H = (V, E, \ell)$ and $H' = (V', E', \ell')$ be two hypergraphs. A* homomorphism *from H to H', denoted $h\colon H \to H'$, is a mapping $h\colon V \to V'$ such that for all $(S, d) \in E$, one has $h(S, d) := (\{h(v) \mid v \in S\}, h(d)) \in E'$ and, for all $v \in V$, it holds that $\ell'(h(v)) = \ell(v)$. The homomorphism h is an* isomorphism *if it is a bijection and, for all $S \subseteq V$ and $d \in V$, we have $(S, d) \in E$ iff $h(S, d) \in E'$.*

Recall that our goal is to find, among all possible proofs of a sentence using a specific set of inference rules, a "minimal" one. Before we define what exactly we mean by "minimal," we first need to specify what "all possible proofs" means. For this, we assume that we are given a *deriver* (e.g. a first-order theorem prover or a DL reasoner) that, given a theory $\mathcal{T}$ and a goal sentence η, produces a derivation structure $\mathcal{D}$. This structure describes all instances of inference rules that are relevant for constructing proofs for η, without us having to know the details of these rules. The quest for a minimal proof thus becomes a search for a minimal proof expressible by the inference steps in $\mathcal{D}$. Since systems of inference rules may not be complete for proving arbitrary sentences of a given logic $\mathcal{L}$, we restrict η in the following definition to a subset $C_\mathcal{L} \subseteq S_\mathcal{L}$ of allowed consequences. For example, in DLs a common reasoning task is *classification*, which amounts to checking entailment of atomic concept subsumptions of the form $A \sqsubseteq B$, where A and B are concept names.

Definition 6 (Deriver)**.** *A* deriver $\mathfrak{D}$ *is given by a set $C_\mathcal{L} \subseteq S_\mathcal{L}$ and a function that assigns derivation structures to pairs $(\mathcal{T}, \eta)$ of finite theories $\mathcal{T} \subseteq S_\mathcal{L}$ and sentences $\eta \in C_\mathcal{L}$, such that $\mathcal{T} \models \eta$ iff $\mathfrak{D}(\mathcal{T}, \eta)$ contains a proof for $\mathcal{T} \models \eta$. A proof $\mathcal{P}$ for $\mathcal{T} \models \eta$ is called* admissible w.r.t. *$\mathfrak{D}(\mathcal{T}, \eta)$ if there is a homomorphism $h\colon \mathcal{P} \to \mathfrak{D}(\mathcal{T}, \eta)$.*

We call $\mathfrak{D}$ a polynomial deriver *if there exists a polynomial $p(x)$ such that the size of $\mathfrak{D}(\mathcal{T}, \eta)$ is bounded by $p(|\mathcal{T}| + |\eta|)$.* Exponential derivers *are defined similarly by the restriction $|\mathfrak{D}(\mathcal{T}, \eta)| \leq 2^{p(|\mathcal{T}|+|\eta|)}$.*

ELK [25] can be seen as a polynomial deriver for $\mathcal{EL}$ in this sense, because it allows to instantiate the inference rules in Figure 1 with only a polynomial number of $\mathcal{EL}$-concepts, namely the subconcepts appearing in $\mathcal{T}$ or η. The derivation structure $\textsc{Elk}(\mathcal{T}, \eta)$ thus contains all allowed instances of these rules, where each sentence is represented by a unique vertex. Since the number of premises in each rule is bounded by 2, the overall size of this structure is polynomial. ELK is complete only for some goal sentences of the form $C \sqsubseteq D$, where D

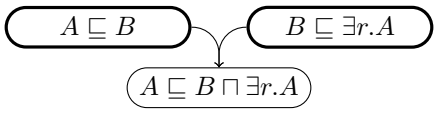

Figure 7: A justification proof

is a concept name and C is a subconcept from $\mathcal{T}$.[1] An example of an exponential deriver is given by the inference rules for $\mathcal{ALC}$ [39], which are complete for entailments of the form $\mathcal{T} \models M \sqsubseteq \bot$, where $\mathcal{T}$ is normalized (see [39] for details) and M is a conjunction of literals (concept names or negated concept names). For each such $\mathcal{T}$ and M, it is guaranteed that there are only exponentially many valid inference steps. Other examples include a triple-exponential inference system for the very expressive DL $\mathcal{ALCHOIQ}$ [12] and an exponential deriver for $\mathcal{ELI}$ [5]. For a survey on consequence-based reasoning in DLs, see [13].

Note that the generality of our definitions also admits proofs as in Figure 7, where the consequence is derived directly from its justification in one inference step. Indeed, such a short proof can sometimes be the most effective explanation. In this paper, we leave the choice of which granularity of inference steps is most appropriate to the chosen deriver.

4 The Complexity of Finding Minimal Proofs

We focus on two kinds of proof representations (cf. Figures 5 and 6). In Section 4.2, we consider tree-shaped proofs in the classical form that is naturally suited for a two-dimensional layout. However, by allowing an arbitrary hypergraphs as in Section 4.1, one may obtain more concise proofs. This is closer to the usual output of an automated inference system, which reports each derived consequence only once. These two approaches have important differences which result in different complexity bounds for the problem of finding a minimal proof. For the sake of readability, we include only proof sketches and defer the complete proofs to the appendix.

4.1 Minimal Proofs

We formally define our main problem: finding minimal admissible proofs w.r.t. $\mathfrak{D}(\mathcal{T}, \eta)$. We consider a proof minimal if it has a minimal number of vertices. Given a proof $\mathcal{P} = (V, E, \ell)$, we define $\mathsf{vert}(\mathcal{P}) := |V|$ for any proof $\mathcal{P} = (V, E, \ell)$.

Definition 7 (Minimal Proof)**.** *Let $\mathfrak{D}$ be a deriver. Given a theory $\mathcal{T}$ and a sentence $\eta \in C_{\mathcal{L}}$, an admissible proof $\mathcal{P}$ w.r.t. $\mathfrak{D}(\mathcal{T}, \eta)$ is called* minimal *if $\mathsf{vert}(\mathcal{P})$ is minimal among all such proofs. The associated decision problem, denoted $\mathsf{MP}(\mathfrak{D})$, is to decide, given $\mathcal{T}$ and η as above and a natural number $n \in \mathbb{N}$, whether there is an admissible proof $\mathcal{P}$ w.r.t. $\mathfrak{D}(\mathcal{T}, \eta)$ with $\mathsf{vert}(\mathcal{P}) \leq n$.*

If $\mathfrak{D}$ is a polynomial deriver, we add the superscript poly to MP, in case of an exponential deriver we add exp. If n is encoded in unary representation, we add a subscript unary, and for binary encoding binary, e.g. $\mathsf{MP}^{\mathsf{poly}}_{\mathsf{unary}}(\mathfrak{D})$.

Note that an admissible proof $\mathcal{P}$ has no less vertices than its homomorphic image in $\mathfrak{D}(\mathcal{T}, \eta)$. We can show that if $\mathcal{P}$ is minimal, then this image must be isomorphic to $\mathcal{P}$, and therefore is also a proof. Thus, to decide $\mathsf{MP}(\mathfrak{D})$ we can focus on proofs which are substructures of $\mathfrak{D}(\mathcal{T}, \eta)$.

Lemma 8. *There exists a substructure in $\mathfrak{D}(\mathcal{T}, \eta)$ which is a minimal proof.*

Our complexity results for $\mathsf{MP}(\mathfrak{D})$ are summarized in Table 1.

[1] To prove other kinds of entailments $\mathcal{T} \models \eta$, one first has to *normalize* $\mathcal{T}$ and η [2].

Table 1: The complexity of $\mathsf{MP}(\mathfrak{D})$.

	poly	exp
unary	NP [Th. 11]	NP [Th. 10]
binary	NP [Th. 9]	NExpTime [Th. 9, Th. 12]

Table 2: The complexity of $\mathsf{MTP}(\mathfrak{D})$.

	poly	exp
unary	P [Th. 17]	NP [Th. 18, Th. 19]
binary	P [Th. 16]	$\leq$ ExpTime [Th. 16]

To show the upper bounds for the second row of the table, we can simply guess a substructure $\mathcal{P}$ of $\mathfrak{D}(\mathcal{T}, \eta)$ and check the conditions of Definitions 2 and 4 and that $\mathsf{vert}(\mathcal{P}) \leq n$.

Theorem 9. $\mathsf{MP}^{\mathsf{poly}}_{\mathsf{binary}}(\mathfrak{D})$ *is in* NP *and* $\mathsf{MP}^{\mathsf{exp}}_{\mathsf{binary}}(\mathfrak{D})$ *is in* NExpTime.

For the case of $\mathsf{MP}^{\mathsf{exp}}_{\mathsf{unary}}(\mathfrak{D})$, we can show an NP-upper bound if we make some additional (reasonable) assumptions. In practice, in order to search for a proof of polynomially bounded size, it does not make sense to construct a complete derivation structure of exponential size first. Instead, a reasoning system would request relevant parts of the derivation structure on demand, for instance guided by a calculus. As a simple abstraction of this, for our complexity upper bounds, we assume that we can access derivation structures by means of *oracle functions* that can instantly check whether a given pair (S, d) is a hyperedge in $\mathfrak{D}(\mathcal{T}, \eta)$, and whether a given vertex has a certain label.[2]

Theorem 10. $\mathsf{MP}^{\mathsf{exp}}_{\mathsf{unary}}(\mathfrak{D})$ *is in* NP.

The NP-hardness of $\mathsf{MP}^{\mathsf{poly}}_{\mathsf{unary}}(\mathfrak{D})$ can be shown by a reduction of the hitting set problem [16] using Elk with the inference rules in Figure 1 as a polynomial deriver. The hardness argument is inspired by a similar result for justifications in [8].

Theorem 11. *There is a polynomial deriver* $\mathfrak{D}$ *such that* $\mathsf{MP}^{\mathsf{poly}}_{\mathsf{unary}}(\mathfrak{D})$ *is* NP*-hard.*

It remains to show one lower bound for Table 1. For $\mathsf{MP}^{\mathsf{exp}}_{\mathsf{binary}}(\mathfrak{D})$, we can use the deriver given by the calculus for $\mathcal{ELI}$ from [5], and perform a reduction from a NExpTime-hard tiling problem. While the reduction is relatively complex, the main idea can be briefly sketched as follows: we use the sentences in the derivation structure to refer to possible combinations of coordinates in the tiling with associated tile types, where two roles are used to refer to possible neighbours. The entailment to be proven requires to infer at least the sentences in a possible tiling. The bound on the size of the proof makes sure that the same coordinates cannot be used twice, thus identifying coordinates with tile types. A special challenge arises from the fact that even auxiliary sentences and inferences contribute to the proof size, which should not depend on specifics of the tiling solutions such as whether all tiling conditions are used or not.

Theorem 12. *There is an exponential deriver* $\mathfrak{D}$ *such that* $\mathsf{MP}^{\mathsf{exp}}_{\mathsf{binary}}(\mathfrak{D})$ *is* NExpTime*-hard.*

4.2 Minimal Tree Proofs

We now introduce a variant of $\mathsf{MP}(\mathfrak{D})$ that considers tree-shaped proofs.

Definition 13 (Minimal Tree Proof)**.** *Let* $\mathfrak{D}$ *be a deriver. Given a theory* $\mathcal{T}$ *and a sentence* $\eta \in C_{\mathcal{L}}$ *such that* $\mathcal{T} \models \eta$*, an admissible tree proof w.r.t.* $\mathfrak{D}(\mathcal{T}, \eta)$ *is called* minimal *if* $\mathsf{vert}(\mathcal{P})$ *is minimal among all such tree proofs. We use* $\mathsf{MTP}(\mathfrak{D})$ *analogously to* $\mathsf{MP}(\mathfrak{D})$.

[2] This is similar to how a random-access Turing machine can access its input tape by specifying cell addresses.

Table 2 summarizes our results regarding $\mathsf{MTP}(\mathfrak{D})$. Surprisingly, for polynomial derivers the complexity drops from NP to P when allowing only tree proofs. The reason is that we can use a greedy, Dijkstra-like algorithm [15, 33] to find a minimal tree proof. The original algorithm is designed to find shortest hyperpaths, which are defined as follows [33].

Definition 14 (Hyperpath)**.** *Given two vertices s and t, a* hyperpath Π *is an acyclic hypergraph $\Pi = (V, E, \ell)$ such that:*

- $s, t \in V$,
- *for any $v \in V \setminus \{s\}$, v is reachable from s in Π, and*
- *there exists no proper substructure of Π with the above properties.*

The second condition implies that for each $v \in V \setminus \{s\}$ there is an incoming hyperedge (S, v) in Π, while the third implies that hyperedge (S, v) is unique, i.e., there is no other incoming hyperedge for v.

Given a derivation structure $\mathcal{D} = (V, E, \ell)$ over a theory $\mathcal{T}$, we define $\mathcal{D}^* = (V \cup \{s\}, E^*, \ell^*)$ by extending $\mathcal{D}$ with an auxiliary vertex s, whose label is arbitrary, and new edges connecting s to every vertex labelled by a tautology or an element of $\mathcal{T}$:

$$E^* = E \cup \{(\{s\}, v) \mid v \in V,\, \ell(v) \in \mathcal{T}\} \cup \{(\{s\}, v) \mid (\emptyset, v) \in E\}.$$

Intuitively, proofs for $\mathcal{T} \models \eta$ in $\mathcal{D}$ correspond to hyperpaths in $\mathcal{D}^*$ that connect s to a target vertex t labeled by η, and tree proofs correspond to the unraveling of such hyperpaths.[3] However, we also need to be able to measure the size of a tree proof based on the corresponding hyperpath. The algorithm from [15] finds a hyperpath $\mathcal{P} = (V, E, \ell)$ in $\mathcal{D}^*$ from s to t that is minimal w.r.t. a so-called *additive weighting function*.[4] We are here interested in the following such function:

$$\begin{aligned} W_{\mathcal{P}}(s) &:= 0, && \text{for the source vertex } s,\\ W_{\mathcal{P}}(d) &:= 1 + \sum_{v \in S} W_{\mathcal{P}}(v), && \text{for every } (S, d) \in E,\\ W_{\mathcal{P}}(\mathcal{P}) &:= W_{\mathcal{P}}(t), && \text{for the target vertex } t. \end{aligned}$$

This is well-defined since a hyperpath can contain no cycles and exactly one incoming edge for each vertex except s. Intuitively, additive weighting functions (corresponding to minimal tree proofs) do not need to keep track of whether a particular vertex/sentence has already been visited before, but simply add a value—in our case 1—whenever the vertex is visited. In other words, admissible tree proofs can use multiple copies of vertices from the derivation structure $\mathfrak{D}(\mathcal{T}, \eta)$. In contrast, for minimal proofs in general (corresponding to special non-additive weighting functions) one has to keep track of which vertices/sentences have already been visited, in order to avoid counting them twice.

Lemma 15. *There is a tree proof $\mathcal{P}$ for $\mathcal{T} \models \eta$ that is admissible w.r.t. $\mathcal{D} = \mathfrak{D}(\mathcal{T}, \eta)$ with $\mathsf{vert}(\mathcal{P}) \leq m$ iff $\mathcal{D}^*$ contains a hyperpath $\mathcal{P}'$ from s to a vertex t labeled with η with $W_{\mathcal{P}'}(\mathcal{P}') \leq m$.*

[3] Similar to graph theory, by the unraveling (unfolding) of a hypergraph we mean a (subset-)minimal tree hypergraph obtained from the original hypergraph by copying vertices, such that for each path in the original hypergraph there is a corresponding path, up to copies, in the unraveled hypergraph.

[4] In the case of non-additive functions, [22] have proved that the problem of finding minimal hyperpaths is NP-hard, which is analogous to the result in Theorem 11.

Hence, the polynomial algorithm *SBT-Dijkstra* from [15] can be used to find minimal tree proofs in polynomial time in the size of the derivation structure $\mathfrak{D}(\mathcal{T}, \eta)$.

Theorem 16. $\mathsf{MTP}^{\mathsf{poly}}_{\mathsf{binary}}(\mathfrak{D})$ *is in* P *and* $\mathsf{MTP}^{\mathsf{exp}}_{\mathsf{binary}}(\mathfrak{D})$ *is in* ExpTime.

For the matching lower bound for $\mathsf{MTP}^{\mathsf{poly}}_{\mathsf{unary}}(\mathfrak{D})$, we reduce the P-hard entailment problem in $\mathcal{EL}$ without existential restrictions [14] to $\mathsf{MTP}(\mathfrak{D})$, where $\mathfrak{D}$ is Elk. The challenge here is to show that minimal tree proofs admissible w.r.t. this derivation structure are bounded polynomially (which trivially holds in the case of MP(Elk)). For that we propose a strategy for applying the inference from Figure 1 (omitting existential restrictions) which demonstrates that there is no superpolynomially large minimal tree proof admissible w.r.t. the deriver.

Theorem 17. *There exists a polynomial deriver* $\mathfrak{D}$ *such that* $\mathsf{MTP}^{\mathsf{poly}}_{\mathsf{unary}}(\mathfrak{D})$ *is* P*-hard under* LogSpace*-reductions.*

Regarding the lower bound for $\mathsf{MTP}^{\mathsf{exp}}_{\mathsf{unary}}(\mathfrak{D})$, we again use the exponential deriver for $\mathcal{ELI}$ from [5], and this time use it for a reduction of SAT. Different to the lower bounds in the last subsection, we cannot use the number of vertices to restrict the usage of sentences throughout the whole proof: the special property of tree proofs is that sub-trees cannot be shared among different parts of the proof. The solution is to build the TBox in such a way that minimal proofs are mostly linear. For SAT, we need to simulate the "guessing" of a valuation for the SAT formula, which is then checked against the different clauses of the input formula. The intuitive idea is to collect the truth values in a single concept using a linear sequence of rule applications, and use another linear sequence to check this concept against each clause one after the other. The number of vertices is used to restrict the non-deterministic choices in the proof, and allows us to enforce that, for each variable, only one valuation is picked.

Theorem 18. *There is an exponential deriver* $\mathfrak{D}$ *such that* $\mathsf{MTP}^{\mathsf{exp}}_{\mathsf{unary}}(\mathfrak{D})$ *is* NP*-hard.*

The remaining NP upper bound for Table 2 can be shown analogously to Theorem 10.

Theorem 19. $\mathsf{MTP}^{\mathsf{exp}}_{\mathsf{unary}}(\mathfrak{D})$ *is in* NP.

5 Finding Proofs in Practice

To analyze the practical aspects of finding minimal proofs, we ran several experiments for reasoning tasks in description logics.[5] We first extracted a dataset containing explanation tasks from the ontologies of the OWL 2 EL classification track of the 2015 OWL Reasoner Evaluation (ORE) [34]. Due to the second experiment described below, we filtered out all sentences that are not expressible in $\mathcal{ELH}$. To avoid re-evaluating proofs that are the same modulo renaming of concept and role names, we extracted a set of *justification patterns*. Specifically, we used Elk to compute all justifications $\mathcal{J}$ of entailments α of the form $A \sqsubseteq B$ or $A \equiv B$ in the filtered ontologies, and identified those that are the same modulo renaming, resulting in a set of 1573 unique entailments $\mathcal{J} \models \alpha$ (up to renaming of concept and role names).[6]

[5] The datasets and scripts used for the experiments can be found at https://lat.inf.tu-dresden.de/2020-finding-small-proofs .

[6] For 4 ontologies the generation of justifications caused an OutOfMemoryError (using Java OpenJDK 11.0.4 with a maximum heap size of 23.5GB), hence any unique $\mathcal{ELH}$ entailments from these ontologies are not part of our dataset.

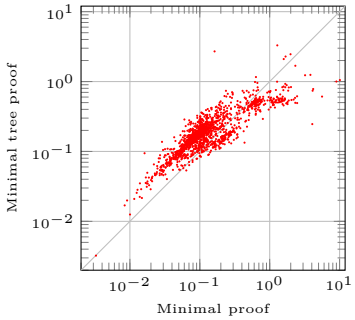

Figure 8: Times for extracting minimal (tree) proofs (in ms). Each dot represents one explanation task.

5.1 Minimal Proofs vs. Minimal Tree Proofs

Internally, ELK generates a derivation structure that contains all possible proofs of a given entailment, possibly even including cycles. ELK 0.5 provides an explanation service that can be used for directly accessing the proof steps [23, 24]: specifically, for a given entailment $C \sqsubseteq D$, it allows to request all rule applications performed by ELK that produce $C \sqsubseteq D$. Recursive requests of this kind allow to reconstruct the entire derivation structures generated by ELK. We used these derivation structures for our purposes to analyze the difference between MP and MTP in practice. We implemented the Dijkstra-based algorithm for extracting minimal tree proofs, and a brute-force recursive procedure for finding minimal (non-tree) proofs. The resulting runtimes for the 1573 explanation tasks are compared in Figure 8. One can see that extracting trees was generally slower, but gained an advantage for larger derivation structures. This is probably due to the overhead of the various data structures required for the Dijkstra algorithm, the implementation of which was not optimized very far.

This gives experimental evidence of the difference in complexity of these two tasks observed in the previous section. More surprisingly, however, it turns out that both methods always yielded equivalent results, i.e., for our dataset the minimal tree proof is always the unraveling of the minimal (non-tree) proof. Although in theory there can be a difference between the two, this does not seem to be relevant in practice, at least not in our dataset.

5.2 Forgetting-Based Proofs

Since finding a proof with a minimal number of vertices is NP-hard, we now leave the goal of optimality aside and simply compare proofs generated by different techniques using different metrics. Since there exist only few implementations of consequence-based algorithms for description logics that are able to produce proofs (ELK [24, 25] being the notable exception), we developed a black-box approach to generate proofs that are not based on a set of inference rules.

The main idea for this approach comes from the observation that the premises of a proof usually contain more symbols than the conclusion. For example, in the proof $\frac{A \sqsubseteq C \qquad C \sqsubseteq B}{A \sqsubseteq B}$ the concept name C is eliminated by the inference step. Hence, the problem of finding a proof looks like a forgetting problem [28, 30], where the symbols that do not occur in the conclusion should be removed. In the example, $\{A \sqsubseteq B\}$ can be seen as the result of forgetting C in the original TBox $\{A \sqsubseteq C,\ C \sqsubseteq B\}$. More generally, a *forgetting-based proof* is composed of multiple steps

$$(C)\ \frac{A \sqsubseteq C \qquad (D)\ \dfrac{C \sqsubseteq \exists r.D}{C \sqsubseteq \exists r.\top}}{(r)\ \dfrac{A \sqsubseteq \exists r.\top \qquad \exists r.\top \sqsubseteq B}{A \sqsubseteq B}}$$

Figure 9: A proof obtained by FBA

$$(C)\ \frac{A \sqsubseteq C \qquad (r)\ \dfrac{C \sqsubseteq \exists r.D \qquad \exists r.\top \sqsubseteq B}{C \sqsubseteq B}}{A \sqsubseteq B}$$

Figure 10: An alternative proof

that correspond to forgetting single symbols.

Definition 20. *Given a TBox $\mathcal{T}$ and a concept or role name X, the result of* forgetting X *in $\mathcal{T}$ is another TBox $\mathcal{T}'$ such that $\mathsf{sig}(\mathcal{T}') \subseteq \mathsf{sig}(\mathcal{T}) \setminus \{X\}$ and for any sentence η with $\mathsf{sig}(\eta) \subseteq \mathsf{sig}(\mathcal{T}) \setminus \{X\}$ we have $\mathcal{T}' \models \eta$ iff $\mathcal{T} \models \eta$.*

The result of forgetting might not always exist in typical description logics, and require the use of additional constructs such as fixpoint operators [28, 30, 41].

Our *forgetting-based approach (FBA)* for constructing proofs can be summarized as follows. Given a TBox $\mathcal{T}$ and a sentence η that is of the form $A \sqsubseteq B$ or $A \equiv B$, the goal is to forget all symbols except A and B from $\mathcal{T}$. Since rarely the whole TBox is needed for a proof, we first extract a justification $\mathcal{J} \subseteq \mathcal{T}$ for η, which already fixes the premises of the final proof. We then construct a sequence of TBoxes $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_n$, where $\mathcal{T}_0 = \mathcal{J}$ and each $\mathcal{T}_i$, $1 \leq i \leq n$, is the result of forgetting one symbol from $\mathcal{T}_{i-1}$ (except A or B), and then extracting a justification for η. This process finishes when $\mathsf{sig}(\mathcal{T}_n) \subseteq \{A, B\}$. If one forgetting step fails, we try to forget a different symbol instead. If this fails for all symbols in the current signature, the process terminates early and we set $\mathcal{T}_n := \{\eta\}$.

To reconstruct the actual proof from these TBoxes, we start with a vertex labeled with η and select a justification $\mathcal{J}_\eta$ for η in $\mathcal{T}_{n-1}$, which gives us a first proof step that derives η from $\mathcal{J}_\eta$. We add a new vertex for each element of $\mathcal{J}_\eta$, and a hyperedge connecting these vertices to the sink. We then recursively justify each element of $\mathcal{J}_\eta$ in $\mathcal{T}_{n-2}$, and continue in this manner until we reach the sentences from the original TBox in $\mathcal{T}_0 = \mathcal{J} \subseteq \mathcal{T}$. Each justification step for an intermediate sentence then corresponds to one inference step in the proof, which can be further annotated by the symbol that was forgotten in the corresponding forgetting operation.

Example 1. *We start with $\mathcal{T}_0 = \{C \sqsubseteq \exists r.D,\ A \sqsubseteq C,\ \exists r.\top \sqsubseteq B\}$ and want to prove that $\mathcal{T}_0 \models A \sqsubseteq B$. We first forget the symbol D, resulting in $\mathcal{T}_1 = \{C \sqsubseteq \exists r.\top,\ A \sqsubseteq C,\ \exists r.\top \sqsubseteq B\}$, then C, which yields $\mathcal{T}_2 = \{A \sqsubseteq \exists r.\top,\ \exists r.\top \sqsubseteq B\}$. Finally, forgetting r obtains $\mathcal{T}_3 = \{A \sqsubseteq B\}$; see Figure 9. The premises of e.g. $A \sqsubseteq \exists r.\top$ in $\mathcal{T}_2$ are a justification of this sentence in $\mathcal{T}_1$.*

Note that the precise result of FBA depends on the choices of justifications in each step as well as the order in which symbols are forgotten.

Example 2. *First forgetting r in $\mathcal{T}_0$ from Example 1 would result in $\mathcal{T}_1 = \{A \sqsubseteq C,\ C \sqsubseteq B\}$ and a different proof in Figure 10.*

However, regardless of these choices, FBA is sound and complete.

Theorem 21. *Let FBA use a sound and complete reasoner to compute justifications and a sound forgetting tool. Given a sentence η and a TBox $\mathcal{T}$, FBA produces a proof $\mathcal{P}$ iff $\mathcal{T} \models \eta$.*

The main advantage of this approach is that it works in a black-box fashion, using any forgetting tool and reasoner to compute justifications (as long as they support $\mathcal{L}$). This means

$$(C)\ \frac{A \sqsubseteq C \qquad C \sqsubseteq \exists r.D}{(r)\ \dfrac{A \sqsubseteq \exists r.\top \qquad \exists r.\top \sqsubseteq B}{A \sqsubseteq B}}$$

Figure 11: A more compact proof for Example 1

that we can find proofs for inferences in expressive logics like $\mathcal{ALCH}$ or $\mathcal{ALCOI}$, for which no consequence-based proof generators exist, using forgetting tools such as Lethe [27] and Fame [41]. FBA also does not explicitly generate tautologies, as many consequence-based reasoners do, cutting down on the size of proofs, but possibly making them harder to follow. An obvious disadvantage compared to consequence-based reasoners is the lack of a predefined set of inference rules, requiring more effort to understand each individual inference step.

5.3 Optimizations for FBA

When actually implementing FBA, we considered several refinements of this method. First, it may be the case that the justification $\mathcal{J}_\alpha$ for $\alpha \in \mathcal{T}_i$ in $\mathcal{T}_{i-1}$ may be exactly $\{\alpha\}$, because α does not contain the symbol that was forgotten in this step. In such a case, we skip the corresponding trivial inference step $\frac{\alpha}{\alpha}$, and instead use a justification for α from the first TBox $\mathcal{T}_j$, $j < i$, such that $\alpha \notin \mathcal{T}_j$. We further extended this optimization: if the justification of a sentence $\alpha \in \mathcal{T}_i$ in $\mathcal{T}_{i-2}$ does not contain more sentences than the justification in $\mathcal{T}_{i-1}$, we choose the premises from the "earlier" TBox $\mathcal{T}_{i-2}$ to skip "simple" steps involving only one sentence. In Example 1, the justification $\{A \sqsubseteq C,\ C \sqsubseteq \exists r.D\}$ for $A \sqsubseteq \exists r.\top$ in $\mathcal{T}_0$ would be preferred to the justification of the same size in $\mathcal{T}_1$, resulting in the more compact proof – see Figure 11.

Regarding the order in which symbols are forgotten, we considered several aspects. Once a role name is forgotten, all proof steps that concern the fillers of existential or value restrictions over this role are hidden from the proof. This is problematic if most proof steps take place "within" a role restriction, which is why role names should be forgotten as late as possible to make the proofs as fine-grained as possible. On the other hand, once a role r does not have non-trivial fillers anymore, i.e., only occurs in expressions like $\exists r.\top$ or $\forall r.\bot$, we should immediately forget it, to focus on the remaining parts of the sentence instead of keeping $\exists r.\top$ around in all intermediate steps. More precisely, the priority with which we try to forget symbols in each step is the following: first role names without non-trivial fillers, then concept names that do not occur nested within existential or value restrictions, then any other concept names, and finally the remaining role names.

5.4 Evaluating FBA

In our second experiment, we compared the minimal proofs extracted from Elk with the ones from our forgetting-based approach, using both Lethe [27] and Fame [41] as black-box forgetters[7] (and HermiT [17] as reasoner). Of Lethe, we used the latest version 0.6,[8] and for Fame, we used the $\mathcal{ALCOI}$-variant of Fame 1.0,[9] since other versions often fail to produce forgetting results that can be handled by the OWL API. Apart from the supported DL, a major

[7] Both Lethe and Fame may generate fixpoint expressions that are not expressible in usual DLs. We omit any such steps from the generated proofs.

[8] https://lat.inf.tu-dresden.de/~koopmann/LETHE

[9] http://www.cs.man.ac.uk/~schmidt/sf-fame

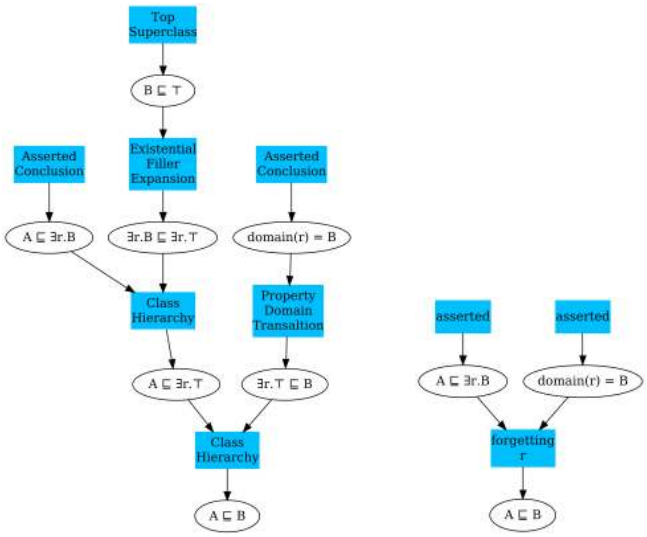

Figure 12: The ELK proof (left) makes several inference steps, which are summarized by LETHE and FAME (right) into a single step.

difference between LETHE and FAME is that LETHE is theoretically guaranteed to compute a sound and complete forgetting result, while FAME is incomplete and may fail to forget a given name. Note that LETHE supports $\mathcal{ALCH}$ and FAME supports $\mathcal{ALCOI}$, but we only consider $\mathcal{ELH}$ reasoning tasks since we want to compare with ELK.

To demonstrate the output of the proof generators, in Figure 12 we provide two alternative proofs, one generated by ELK and the other by FBA. The automatically generated images use blue boxes to denote hyperedges (S, d), with incoming and outgoing arrows indicating S and d, respectively, and a label indicating the type of inference rule (for ELK) or the forgotten symbol(s) (for FBA). In contrast to our previous definition of hypergraphs, both ELK and our implementation of FBA treat sentences from the TBox not as leaves, but as being derived by a special kind of hyperedges $(\emptyset, v)$ labeled by "asserted" or "Asserted Conclusion".

We generated such proofs for each explanation task from our dataset using the 3 tools and compared these proofs according to different metrics. In Figure 13 we compare the *hypergraph size* (number of vertices) and *tree size* (number of vertices of the unraveling) of the generated proofs, where each dot may represent multiple entailments $\mathcal{J} \models \alpha$. FAME did not generate proofs for 656 of the 1573 tasks, which are assigned a score of 0 in all diagrams (290 contained role inclusions, which are not supported by the version of FAME that we used, and in 366 cases it returned an incorrect forgetting result[10]). In contrast, LETHE is complete for $\mathcal{ALCH}$ forgetting and always yields a proof using the preferred order of forgetting symbols. We can see that most of the forgetting-based proofs were of similar size or smaller than the corresponding ones generated by ELK. However, recall that the proofs generated using LETHE and FAME have the advantage that they can use inference steps in more expressive logics than $\mathcal{ELH}$, which may result in more compact proofs. Moreover, the proofs generated using FAME were usually larger than the ones from LETHE, which can be explained by additional simplification steps performed by LETHE when computing the forgetting result [27].

Next we wanted to find out how hard it would be for a user to actually understand the

[10] As this points at a bug in the tool that might not have been detected before, we have informed the developer.

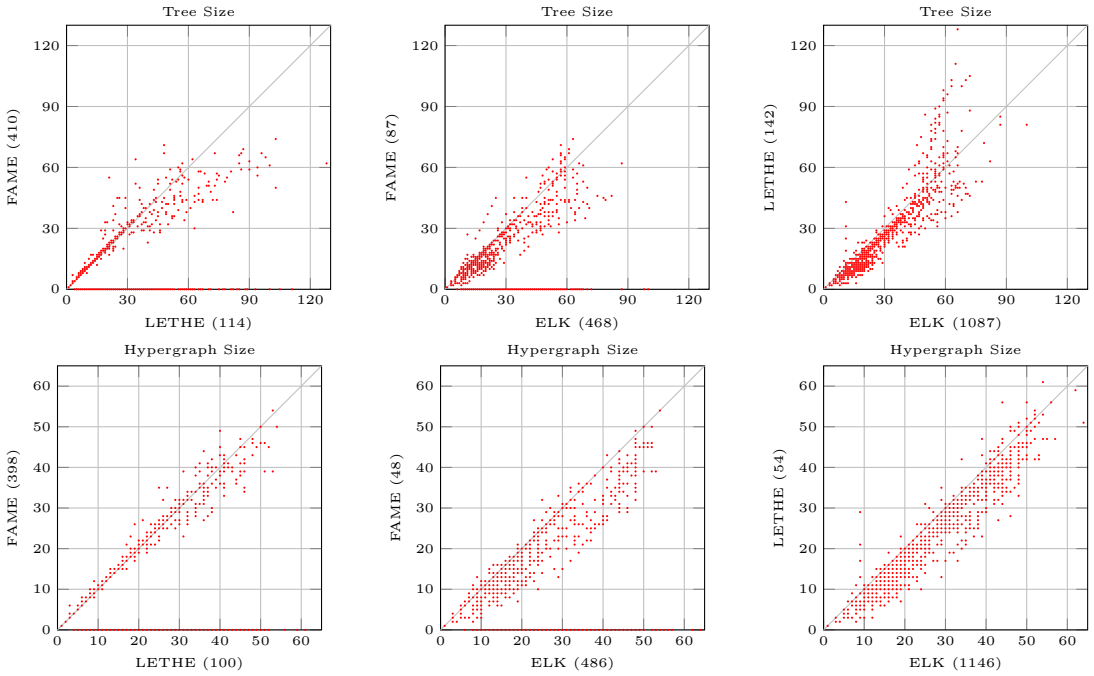

Figure 13: Comparison of the tree size and hypergraph size of the proofs generated by the three tools. In parentheses are the numbers of entries that lie strictly above or below the diagonals (excluding cases where one of the tools did not produce a proof at all, which are evaluated to 0).

generated proofs. This is hard to evaluate empirically, but was already attempted for plain justifications [19], resulting in a *justification complexity model* that assigns a numerical score to an entailment task $\mathcal{J} \models \alpha$, indicating its difficulty. We evaluated all individual inference steps in the generated proofs according to this model and report in Figure 14 the sum and the maximum of the justification complexities of all inference steps in each proof. Here, the advantage of the forgetting-based approach over ELK was less pronounced. Although the sum of justification complexities is correlated with the hypergraph size, the difficulty of the individual steps now also plays a role. For maximum justification complexity, LETHE scored worse than ELK more often than vice versa, but for FAME and ELK the situation is reversed (on a restricted dataset, because FAME did not generate proofs in all cases).

Looking at individual examples, however, the simplicity of the ELK inference rules is clearly an advantage when one wants to understand the generated proofs. In total, we could find 362 distinct "inference rules" used in the proofs generated using LETHE (up to renaming of concept and role names). For FAME, we extracted 281 inference rules. This large diversity is obviously not captured by the hypergraph size, but also not accurately represented by the justification complexity, which only measures single inference steps. Moreover, understanding inference steps formulated in $\mathcal{ALCH}$ or $\mathcal{ALCOI}$ is inherently harder than for pure $\mathcal{ELH}$ proofs. In the appendix, we show some more examples of the generated proofs.

We also attempted some experiments on $\mathcal{ALCH}$ ontologies from the ORE dataset, but were not able to extract many explanation tasks (that were not already in the $\mathcal{ELH}$ dataset), mostly due to time and memory problems when computing all possible justifications for all $\mathcal{ALCH}$ entailments using a black-box approach based on HermiT (for the $\mathcal{ELH}$ dataset we

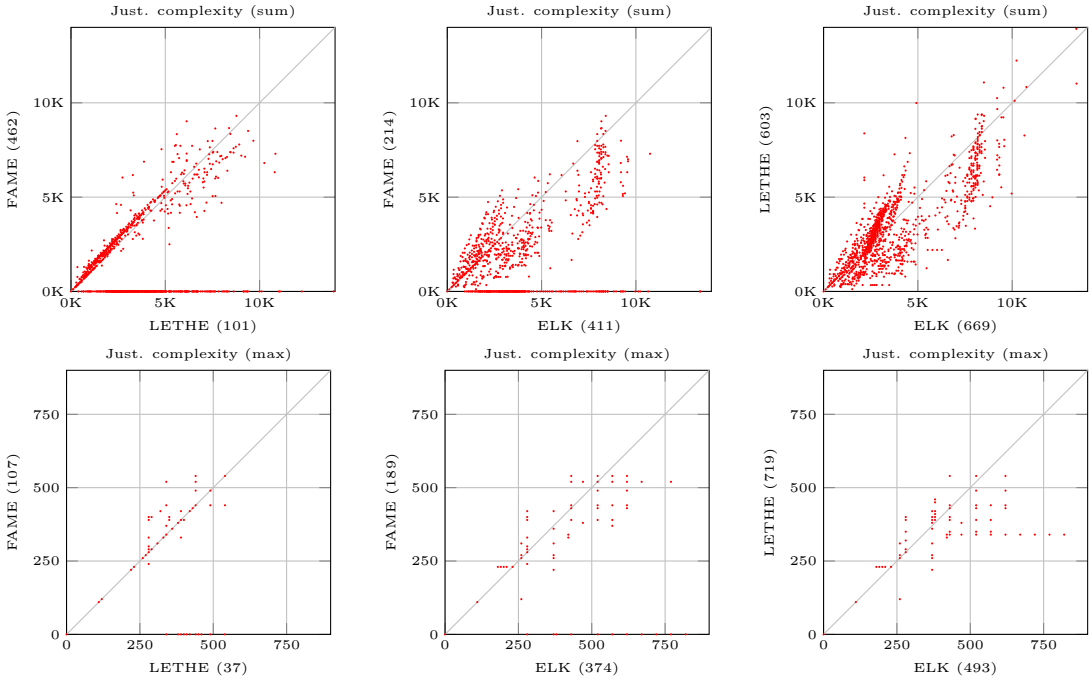

Figure 14: The total and maximal justification complexity [19] of proofs.

could directly use Elk to compute all justifications).

6 Conclusion

We have investigated the complexity as well as the practical aspects of finding small proofs for description logic entailments. Obviously, size is not enough to evaluate the difficulties users face when trying to understand proofs, and even existing measures such as the justification complexity model from [19] cannot capture all possible aspects of this problem. Hence, in future work we will extend our results to more complex measures of understandability for proofs. We conjecture that many of our results can be transferred, e.g. when the goal is to minimize the sum of justification complexities we also obtain an NP-complete decision problem, whereas for the maximum of justification complexities we expect a P-upper bound since such maxima can also be computed by a greedy, Dijkstra-like approach.

Acknowledgements This work was partially supported by DFG grant 389792660 as part of TRR 248 (https://perspicuous-computing.science), and the DFG Research Training Group QuantLA, GRK 1763 (https://lat.inf.tu-dresden.de/quantla).

References

[1] Alejandro Barredo Arrieta, Natalia Diaz-Rodriguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador Garcia, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja

Chatila, and Francisco Herrera. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58:82–115, 2020.

[2] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ envelope. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI'05)*, pages 364–369. Professional Book Center, 2005. URL: http://ijcai.org/Proceedings/09/Papers/053.pdf.

[3] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ envelope further. In Kendall Clark and Peter F. Patel-Schneider, editors, *Proc. of the 4th Workshop on OWL: Experiences and Directions*, pages 1–10, 2008. URL: http://webont.org/owled/2008dc/papers/owled2008dc_paper_3.pdf.

[4] Franz Baader, Ian Horrocks, Carsten Lutz, and Uli Sattler. *An Introduction to Description Logic*. Cambridge University Press, 2017. URL: http://dltextbook.org/.

[5] Franz Baader, Ian Horrocks, Carsten Lutz, and Uli Sattler. Reasoning in the $\mathcal{EL}$ family of description logics. In *An Introduction to Description Logic*, pages 140–167. Cambridge University Press, 2017. doi:10.1017/9781139025355.006.

[6] Franz Baader, Carsten Lutz, and Boontawee Suntisrivaraporn. CEL—A polynomial-time reasoner for life science ontologies. In Ulrich Furbach and Natarajan Shankar, editors, *Proc. of the 3rd Int. Joint Conf. on Automated Reasoning (IJCAR'06)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 287–291. Springer-Verlag, 2006.

[7] Franz Baader, Rafael Peñaloza, and Boontawee Suntisrivaraporn. Pinpointing in the description logic $\mathcal{EL}^+$. In *Proc. of the 30th German Annual Conf. on Artificial Intelligence (KI'07)*, volume 4667 of *Lecture Notes in Computer Science*, pages 52–67, Osnabrück, Germany, 2007. Springer.

[8] Franz Baader, Rafael Peñaloza, and Boontawee Suntisrivaraporn. Pinpointing in the description logic $\mathcal{EL}^+$. In *KI 2007: Advances in Artificial Intelligence, 30th Annual German Conference on AI, KI 2007, Osnabrück, Germany, September 10-13, 2007, Proceedings*, pages 52–67, 2007. doi:10.1007/978-3-540-74565-5_7.

[9] Franz Baader and Boontawee Suntisrivaraporn. Debugging SNOMED CT using axiom pinpointing in the description logic $\mathcal{EL}^+$. In *Proc. of the 3rd Conference on Knowledge Representation in Medicine (KR-MED'08): Representing and Sharing Knowledge Using SNOMED*, volume 410 of *CEUR-WS*, 2008.

[10] Alexander Borgida, Enrico Franconi, and Ian Horrocks. Explaining $\mathcal{ALC}$ subsumption. In *ECAI 2000, Proceedings of the 14th European Conference on Artificial Intelligence, Berlin, Germany, August 20-25, 2000*, pages 209–213, 2000. URL: http://www.frontiersinai.com/ecai/ecai2000/pdf/p0209.pdf.

[11] Ronald J. Brachman, Deborah L. McGuinness, Peter F. Patel-Schneider, Lori Alperin Resnick, and Alexander Borgida. Living with CLASSIC: When and how to use a KL-ONE-like language. In John F. Sowa, editor, *Principles of Semantic Networks*, pages 401–456. Morgan Kaufmann, 1991.

[12] David Tena Cucala, Bernardo Cuenca Grau, and Ian Horrocks. Consequence-based reasoning for description logics with disjunction, inverse roles, number restrictions, and nominals. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 1970–1976, 2018. doi:10.24963/ijcai.2018/272.

[13] David Tena Cucala, Bernardo Cuenca Grau, and Ian Horrocks. 15 years of consequence-based reasoning. In *Description Logic, Theory Combination, and All That - Essays Dedicated to Franz Baader on the Occasion of His 60th Birthday*, pages 573–587, 2019. doi:10.1007/978-3-030-22102-7_27.

[14] William F. Dowling and Jean H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *J. Log. Program.*, 1(3):267–284, 1984. doi:10.1016/0743-1066(84)90014-1.

[15] Giorgio Gallo, Giustino Longo, and Stefano Pallottino. Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42(2):177–201, 1993. doi:10.1016/0166-218X(93)90045-P.

[16] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1990.

[17] Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang. HermiT: an OWL 2 reasoner. *Journal of Automated Reasoning*, 53(3):245–269, 2014.

[18] Matthew Horridge. *Justification Based Explanation in Ontologies*. PhD thesis, University of Manchester, UK, 2011. URL: https://www.research.manchester.ac.uk/portal/files/54511395/FULL_TEXT.PDF.

[19] Matthew Horridge, Samantha Bail, Bijan Parsia, and Uli Sattler. Toward cognitive support for OWL justifications. *Knowledge-Based Systems*, 53:66–79, 2013. doi:10.1016/j.knosys.2013.08.021.

[20] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Justification oriented proofs in OWL. In *The Semantic Web - ISWC 2010 - 9th International Semantic Web Conference, ISWC 2010, Shanghai, China, November 7-11, 2010, Revised Selected Papers, Part I*, pages 354–369, 2010. doi:10.1007/978-3-642-17746-0_23.

[21] Xiaorong Huang. Reconstruction proofs at the assertion level. In Alan Bundy, editor, *Proc. of 12th International Conference on Automated Deduction (CADE-12)*, volume 814 of *Lecture Notes in Computer Science*, pages 738–752. Springer, 1994.

[22] Giuseppe F Italiano and Umberto Nanni. Online maintenance of minimal directed hypergraphs. *Proceedings 3" Convegno Italiano di Informatica Teorica, Mantova*, pages 335–349, 1989.

[23] Yevgeny Kazakov and Pavel Klinov. Goal-directed tracing of inferences in $\mathcal{EL}$ ontologies. In Peter Mika, Tania Tudorache, Abraham Bernstein, Chris Welty, Craig A. Knoblock, Denny Vrandecic, Paul T. Groth, Natasha F. Noy, Krzysztof Janowicz, and Carole A. Goble, editors, *Proc. of the 13th International Semantic Web Conference (ISWC 2014)*, volume 8797 of *Lecture Notes in Computer Science*, pages 196–211. Springer, 2014.

[24] Yevgeny Kazakov, Pavel Klinov, and Alexander Stupnikov. Towards reusable explanation services in protege. In Alessandro Artale, Birte Glimm, and Roman Kontchakov, editors, *Proc. of the 30th Int. Workshop on Description Logics (DL'17)*, volume 1879 of *CEUR Workshop Proceedings*, 2017. URL: http://www.ceur-ws.org/Vol-1879/paper31.pdf.

[25] Yevgeny Kazakov, Markus Krötzsch, and Frantisek Simancik. The incredible ELK – from polynomial procedures to efficient reasoning with $\mathcal{EL}$ ontologies. *J. Autom. Reasoning*, 53(1):1–61, 2014. doi:10.1007/s10817-013-9296-3.

[26] Boris Konev, Dirk Walther, and Frank Wolter. Forgetting and uniform interpolation in extensions of the description logic $\mathcal{EL}$. In Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, and Ulrike Sattler, editors, *Proceedings of the 22nd International Workshop on Description Logics (DL 2009), Oxford, UK, July 27-30, 2009*, volume 477 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009. URL: http://ceur-ws.org/Vol-477/paper_28.pdf.

[27] Patrick Koopmann. LETHE: Forgetting and uniform interpolation for expressive description logics. *KI - Künstliche Intelligenz*, 2020. doi:10.1007/s13218-020-00655-w.

[28] Patrick Koopmann and Renate A. Schmidt. Forgetting concept and role symbols in $\mathcal{ALCH}$-ontologies. In *Logic for Programming, Artificial Intelligence, and Reasoning - LPAR-19*, volume 8312 of *LNCS*, pages 552–567. Springer, 2013. doi:10.1007/978-3-642-45221-5_37.

[29] Christoph Lingenfelder. Structuring computer generated proofs. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence. Detroit, MI, USA, August 1989*, pages 378–383, 1989. URL: http://ijcai.org/Proceedings/89-1/Papers/060.pdf.

[30] Carsten Lutz and Frank Wolter. Foundations for uniform interpolation and forgetting in expressive description logics. In *Proceedings of IJCAI 2011*, pages 989–995. IJCAI/AAAI, 2011. doi:10.5591/978-1-57735-516-8/IJCAI11-170.

[31] Deborah L. McGuinness. *Explaining Reasoning in Description Logics*. PhD thesis, Rutgers University, NJ, USA, 1996. doi:10.7282/t3-q0c6-5305.

[32] Tu Anh Thi Nguyen, Richard Power, Paul Piwek, and Sandra Williams. Measuring the understand-

ability of deduction rules for OWL. In *Proceedings of the First International Workshop on Debugging Ontologies and Ontology Mappings, WoDOOM 2012, Galway, Ireland, October 8, 2012.*, pages 1–12, 2012. URL: http://www.ida.liu.se/~patla/conferences/WoDOOM12/papers/paper4.pdf.

[33] Lars Relund Nielsen, Kim Allan Andersen, and Daniele Pretolani. Finding the K shortest hyperpaths. *Computers & OR*, 32:1477–1497, 2005. doi:10.1016/j.cor.2003.11.014.

[34] Bijan Parsia, Nicolas Matentzoglu, Rafael S. Gonçalves, Birte Glimm, and Andreas Steigmiller. The OWL Reasoner Evaluation (ORE) 2015 competition report. *J. Autom. Reasoning*, 59(4):455–482, 2017. doi:10.1007/s10817-017-9406-8.

[35] Klaus Schild. A correspondence theory for terminological logics: Preliminary report. In John Mylopoulos and Raymond Reiter, editors, *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, pages 466–471. Morgan Kaufmann, 1991. URL: http://ijcai.org/Proceedings/91-1/Papers/072.pdf.

[36] Marvin R. G. Schiller and Birte Glimm. Towards explicative inference for OWL. In *Informal Proceedings of the 26th International Workshop on Description Logics, Ulm, Germany, July 23 - 26, 2013*, pages 930–941, 2013. URL: http://ceur-ws.org/Vol-1014/paper_36.pdf.

[37] Marvin R. G. Schiller, Florian Schiller, and Birte Glimm. Testing the adequacy of automated explanations of EL subsumptions. In *Proceedings of the 30th International Workshop on Description Logics, Montpellier, France, July 18-21, 2017.*, 2017. URL: http://ceur-ws.org/Vol-1879/paper43.pdf.

[38] Stefan Schlobach and Ronald Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In Georg Gottlob and Toby Walsh, editors, *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003)*, pages 355–362, Acapulco, Mexico, 2003. Morgan Kaufmann.

[39] Frantisek Simancik, Yevgeny Kazakov, and Ian Horrocks. Consequence-based reasoning beyond horn ontologies. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 1093–1098, 2011. doi:10.5591/978-1-57735-516-8/IJCAI11-187.

[40] Peter van Emde Boas. The convenience of tilings. *Lecture Notes in Pure and Applied Mathematics*, pages 331–363, 1997.

[41] Yizheng Zhao and Renate A. Schmidt. FAME: an automated tool for semantic forgetting in expressive description logics. In *Automated Reasoning - 9th International Joint Conference, IJCAR 2018*, volume 10900 of *LNCS*, pages 19–27. Springer, 2018. doi:10.1007/978-3-319-94205-6_2.

A Proofs Omitted from the Main Text

Since we investigative the problem of finding a minimal proof, we concentrate on proofs without redundant elements.

Definition 22 (Redundant)**.** *We call a proof $\mathcal{P}$ for $\mathcal{T} \models \eta$* redundant *if it properly contains another proof for $\mathcal{T} \models \eta$; otherwise it is* non-redundant*.*

A non-redundant proof for $\mathcal{T} \models \eta$ cannot contain several vertices labeled with η, or two inference steps that lead to the same vertex (proved by Lemma 23 (3) below). However, it is still possible that there are two vertices labelled with the same sentence.

For proofs we can show some important properties.

Lemma 23. *Let $\mathcal{P} = (V, E, \ell)$ be a proof for $\mathcal{T} \models \eta$. Then*

1. *all paths in $\mathcal{P}$ are finite and all longest paths in $\mathcal{P}$ have v_η as the target;*
2. *$\mathcal{T} \models \eta$;*
3. *if $\mathcal{P}$ is non-redundant, there is no vertex $w \in V$ s.t. there are $(S_1, w), (S_2, w) \in E$ with $S_1 \neq S_2$.*

Proof. The first statement trivially follows from the acyclicity and the only sink v_η in $\mathcal{P}$. The length of a path in $\mathcal{P}$ can be bounded by $|V|$.

The second claim can be shown by an induction on the depth of $\mathcal{P}$. Namely, for every k and every $w \in V$ s.t. all paths leading to w have length at most k it holds that $\mathcal{T} \models \ell(w)$. The induction base follows from the fact that the leafs are labelled with the sentences from $\mathcal{T}$. For the induction step, for a vertex $w \in V$, we consider an hyperedge $(S, w) \in E$. Every $s \in S$ satisfies the induction hypothesis and, thus, $\mathcal{T} \models \ell(s)$. By $\mathcal{P}$ being a derivation structure, it holds $\{\ell(s) | s \in S\} \models \ell(w)$ and, by transitivity of model-based entailment, $\mathcal{T} \models \ell(w)$.

Now, we consider a non-redundant $\mathcal{P}$. Assume that there is a vertex $w \in V$ with two incoming edges, (S_1, w) and (S_2, w). By removing one of the edges, say (S_1, w), and, if a vertex $v \in S_1$ becomes a sink, removing it and all its incoming edges (S, v), and continuing recursively in this fashion, we obtain a proper substructure of $\mathcal{P}$. The new derivation structure still has the original sink v_η from $\mathcal{P}$ and no cycles can be created by the described procedure. This contradicts $\mathcal{P}$ being non-redundant. □

Since redundant proofs will in general not be minimal, we can restrict most of our considerations to non-redundant proofs.

Lemma 8. *There exists a substructure in $\mathfrak{D}(\mathcal{T}, \eta)$ which is a minimal proof.*

Proof. Let $\mathcal{P} = (V, E, \ell)$ be a minimal proof w.r.t. $\mathfrak{D}(\mathcal{T}, \eta)$. By definition, there is a homomorphism $h \colon \mathcal{P} \to \mathfrak{D}(\mathcal{T}, \eta)$. We show that h is injective. Assume to the contrary that there are two vertices $v, v' \in V$ with $h(v) = h(v')$, which implies that $\ell(v) = \ell(v')$. If there is no path between v and v' in $\mathcal{P}$, then we can construct $\mathcal{P}'$ by renaming all occurrences of v in $\mathcal{P}$ to v'. This operation does not introduce any leafs, sinks, or cycles (in particular no cycles involving the sink of $\mathcal{P}$), and hence $\mathcal{P}'$ is still a proof for $\mathcal{T} \models \eta$. But $\mathsf{vert}(\mathcal{P}') < \mathsf{vert}(\mathcal{P})$, which contradicts minimality of $\mathcal{P}$.

It remains to deal with the case where there is a path between v and v'. In this case, we ensure that v and v' are not connected anymore before merging them. Since $\mathcal{P}$ is acyclic, we can assume that there is a path from v to v', but no path from v' to v. We now start

from v' and remove v' and all hyperedges of the form (S, v') from $\mathcal{P}$, which already destroys any paths from v to v'. However, this operation may make some vertices $w \in S$ into sinks, and we continue removing such w and hyperedges (S', w) until either no new sinks are produced or we have reached v. The only remaining loose ends are the hyperedges (S, d) in $\mathcal{P}$ that still contain $v' \in S$. In such hyperedges, we replace v' by v, effectively merging the two vertices. If there was no such hyperedge, then v' was the sink of $\mathcal{P}$, i.e., $\ell(v') = \eta$, and v will now be the new sink with $\ell(v) = \ell(v') = \eta$.

We show that the result $\mathcal{P}'$ of this procedure is still a proof, which has less vertices than $\mathcal{P}$, which again contradicts our assumption on $\mathcal{P}$. Since our procedure removes edges only together with their destination vertices, it cannot produce new leafs, and hence all leafs are still labeled by sentences from $\mathcal{T}$. Clearly, all remaining edges are sound since they were already sound in $\mathcal{P}$. As argued above, we have also kept the property that there is exactly one sink, which is labeled with η. Finally, all cycles in $\mathcal{P}'$ can be traced back to paths in $\mathcal{P}$ that involve both v and v'. But we have assumed that there are no paths from v' to v, and have destroyed all paths from v to v'.

This shows that h must be injective, and therefore its image $h(\mathcal{P})$ is isomorphic to $\mathcal{P}$. This means that $h(\mathcal{P})$ is a substructure of $\mathfrak{D}(\mathcal{T}, \eta)$ which is a proof for $\mathcal{T} \models \eta$ that is also minimal because $\mathsf{vert}(h(\mathcal{P})) = \mathsf{vert}(\mathcal{P})$. □

We first show the upper bounds for the second row of Table 1.

Theorem 9. $\mathsf{MP}^{\mathsf{poly}}_{\mathsf{binary}}(\mathfrak{D})$ *is in* NP *and* $\mathsf{MP}^{\mathsf{exp}}_{\mathsf{binary}}(\mathfrak{D})$ *is in* NExpTime.

Proof. We start by guessing a substructure $\mathcal{P}$ of $\mathfrak{D}(\mathcal{T}, \eta)$. This can be done in polynomial time in the size of $\mathfrak{D}(\mathcal{T}, \eta)$, which for $\mathsf{MP}^{\mathsf{poly}}_{\mathsf{binary}}$ is polynomial in the size of $\mathcal{T}$ and η, and for $\mathsf{MP}^{\mathsf{exp}}_{\mathsf{binary}}$ is exponential. This substructure $\mathcal{P}$ is necessarily sound and checking the remaining conditions of Definitions 2 and 4 and that $\mathcal{P}$ is not larger than n can be done in polynomial time in the number of vertices in $\mathcal{P}$. □

In the case of $\mathsf{MP}^{\mathsf{exp}}_{\mathsf{unary}}(\mathfrak{D})$, we can also show an NP-upper bound, but for this we have to be more careful about how we access the structure $\mathfrak{D}(\mathcal{T}, \eta)$, which is already of exponential size.

Namely, we make a few additional assumptions about how the deriver works. These assumptions do not make a difference for the other cases, but enable us to actually guess a structure of polynomial size (at most n vertices and one edge for each vertex) without needing to store $\mathfrak{D}(\mathcal{T}, \eta)$, which is of exponential size.

Let $\mathcal{D} = (V, E, \ell) := \mathfrak{D}(\mathcal{T}, \eta)$. We assume that the vertices $V = \{v_1, \dots, v_m\}$ are numbered consecutively, where m is an index of polynomial size (in binary notation). Hence, in order to address a single vertex v_i, we only need access to its index i, $1 \leq i \leq m$, instead of to the whole set V. Furthermore, we only access the structure $\mathcal{D}$ by means of two oracle functions

$$[\mathcal{D}](v_1, \dots, v_l) := \begin{cases} \mathsf{true} & \text{if } (\{v_1, \dots, v_{l-1}\}, v_l) \in E, \\ \mathsf{false} & \text{otherwise;} \end{cases}$$

$$[\mathcal{D}](v, \alpha) := \begin{cases} \mathsf{true} & \text{if } \ell(v) = \alpha, \\ \mathsf{false} & \text{otherwise.} \end{cases}$$

where $v, v_1, \dots, v_l$ are vertices (represented by their indices) and α is an sentence. These functions hide any details of the internal representation of $\mathcal{D}$. For example, if $v_1, \dots, v_m$ is an efficient indexing scheme for the (exponentially many) sentences in $\mathcal{D}$, then the call to

$[\mathcal{D}](v_1, \ldots, v_l)$ can be answered by retrieving the sentences $\alpha_1, \ldots, \alpha_l$ corresponding to $v_1, \ldots, v_l$ in polynomial time and checking whether $\frac{\alpha_1 \quad \ldots \quad \alpha_{l-1}}{\alpha_l}$ is a valid rule instance.

Having fixed a theory $\mathcal{T}$ and a sentence η, by this restricted view on the exponentially large output of a deriver, we can show the following result.

Theorem 10. $\mathsf{MP}^{\mathsf{exp}}_{\mathsf{unary}}(\mathfrak{D})$ *is in* NP.

Proof. Using the setup described above, we can guess $n' \leq n$ vertices $v_{i_1}, \ldots, v_{i_{n'}}$ and for each vertex at most one incoming edge (S_{i_j}, v_{i_j}) with $S_{i_j} \subseteq \{v_{i_1}, \ldots, v_{i_{n'}}\}$, in polynomial time in the sizes of n, $\mathcal{T}$, and η. Recall that we can restrict our search to non-redundant proofs here since we are looking for minimal proofs anyway. We can then check the following conditions to ensure that we have guessed a valid proof.

First, we make sure that every guessed edge $(\{v_1, \ldots, v_{l-1}\}, v_l)$ is actually an edge in $\mathcal{D}$ by calling $[\mathcal{D}](v_1, \ldots, v_l)$; this also guarantees soundness of the proof since $\mathcal{D}$ is a derivation structure. To check completeness, we use $[\mathcal{D}](v, \alpha)$ to ensure that any vertex v for which we have not guessed an incoming edge is labeled by some $\alpha \in \mathcal{T}$. Finally, we use several reachability checks in the guessed structure to make sure that there is a unique sink, which is labeled by η, and that there are no cycles. □

As a consequence of the theorem above, $\mathsf{MP}^{\mathsf{poly}}_{\mathsf{unary}}(\mathfrak{D})$ is also in NP. Moreover, the NP-hardness can be shown by a reduction of the hitting set problem [16], which is inspired by a proof about minimal justifications in [8].

Theorem 11. *There is a polynomial deriver* $\mathfrak{D}$ *such that* $\mathsf{MP}^{\mathsf{poly}}_{\mathsf{unary}}(\mathfrak{D})$ *is* NP*-hard.*

Proof. Given a finite set $\mathcal{U} = \{u_1, \ldots, u_l\}$, sets $s_i \subseteq \mathcal{U}$, $1 \leq i \leq k$, and a number $m \in \mathbb{N}$, we need to decide whether there is a set $s \subseteq \mathcal{U}$ of size less than m such that $s_i \cap s \neq \emptyset$, $1 \leq i \leq k$. We use an instance of $\mathsf{MP}^{\mathsf{poly}}_{\mathsf{unary}}$(ELK) with an $\mathcal{EL}$-TBox $\mathcal{T}$, sentence η and number n as follows.

Let $A, B, U_1, \ldots, U_l, S_1, \ldots, S_k$ be concept names. $\mathcal{T}$ contains the following sentences:

$$\begin{aligned} A &\sqsubseteq U_i, && \text{for every } i,\ 1 \leq i \leq l; \\ U_i &\sqsubseteq S_j, && \text{for every } u_i \in s_j, 1 \leq j \leq k, 1 \leq i \leq l; \\ S_1 \sqcap \cdots \sqcap S_k &\sqsubseteq B. \end{aligned}$$

A proof of $\eta := A \sqsubseteq B$ over $\mathcal{T}$ using the inference rules of ELK [25] first derives $A \sqsubseteq S_j$ for each j, $1 \leq j \leq k$, from $A \sqsubseteq U_i$ and $U_i \sqsubseteq S_j$ by rule $\mathsf{R}_{\sqsubseteq}$ (see Figure 1) for some i, $1 \leq i \leq l$, where the same sentence $A \sqsubseteq U_i$ can be used for different S_j. Then it composes the sentences $A \sqsubseteq S_j$ into $A \sqsubseteq S_1 \sqcap \cdots \sqcap S_k$ with $\mathsf{R}^+_{\sqcap}$ using $k-1$ inference steps, before finally deriving $A \sqsubseteq B$ with the help of $S_1 \sqcap \cdots \sqcap S_k \sqsubseteq B$ and $\mathsf{R}_{\sqsubseteq}$. The number of sentences in this proof is $m' + 3k + 1$, where m' is the number of sentences $A \sqsubseteq U_i$, which represent the hitting set s. Hence, there is a hitting set of size $\leq m$ iff there a proof as described above with $\leq n := m + 3k + 1$ vertices. □

The result in Theorem 11 also provides the lower bound for all problems in Table 1. However, for an exponential deriver and binary encoding, we are able to strengthen it to NExpTime.

We first state a result on tiling problems to simplify the proof of the theorem. We consider the following *exponential tiling problem*: given a set T of tiles, two sets $h, v \subseteq T \times T$ of horizontal and vertical tiling conditions, an initial sequence $t_0, \ldots, t_s \in T$ of tiles, and a number n encoded in unary, decide whether there exists a function $f\colon [0, 2^n - 1] \times [0, 2^n - 1] \to T$ such that $f(i, 0) = t_i$ for $0 \leq i \leq s$, $\langle f(i, j), f(i+1, j)\rangle \in h$ and $\langle f(j, i), f(j, i+1)\rangle \in v$ for all $i \in [0, 2^n - 2]$ and $j \in [0, 2^n - 1]$. This problem is known to be NExpTime-complete [40]. Furthermore, it is sufficient to consider a variant for which $s = 0$.

$$\mathsf{CR1}\ \frac{}{K \sqsubseteq A}\ \text{if } A \in K \text{ and } K \text{ appears in } \mathcal{T}'$$

$$\mathsf{CR2}\ \frac{M \sqsubseteq A \text{ for all } A \in K,\ K \sqsubseteq C}{M \sqsubseteq C}\ \text{if } M \text{ appears in } \mathcal{T}'$$

$$\mathsf{CR3}\ \frac{M \sqsubseteq \exists r.L \quad L \sqsubseteq \forall r^-.A}{M \sqsubseteq A} \qquad \mathsf{CR4}\ \frac{L \sqsubseteq \exists r.M \quad L \sqsubseteq \forall r.A}{L \sqsubseteq \exists r.(M \sqcap A)}$$

Figure 15: The inference rules for $\mathcal{ELI}$ [5]. Given a TBox $\mathcal{T}$ in a normal form, the rules produce a saturated TBox $\mathcal{T}'$. Here, K, L, M are conjunctions of concept names, A is a concept name, C is an $\mathcal{ELI}$ concept of the form A, $\exists r.M$, or $\forall r.A$, and r is a role name or the inverse of a role name. In this calculus conjunctions are implicitly viewed as sets, i.e., the order and multiplicity of conjuncts is ignored.

Lemma 24. *Let T, $h, v \subseteq T \times T$, n and $t_0, \dots, t_s \in T$ be a tiling problem. Then, there exists a tiling problem T', $h', v' \subseteq T' \times T'$ with just one initial tile $t'_0 \in T$ that has a solution iff the original tiling problem has a solution.*

Proof. We assign to every tile t_i, $1 \leq i \leq s$, a unique fresh tile t'_i, and then define the new tiling problem as follows.

$$\begin{aligned} h' = h \cup \{&\langle t'_0, t'_1\rangle, \dots, \langle t'_{s-1}, t'_s\rangle\} \\ &\cup \{\langle t'_s, t\rangle \mid \langle t_s, t\rangle \in h\} \\ v' = v \cup \{&\langle t'_i, t\rangle \mid 0 \leq i \leq s, \langle t_i, t\rangle \in v\}. \end{aligned}$$

It is easy to verify that this tiling problem satisfies the requirements. □

Theorem 12. *There is an exponential deriver $\mathfrak{D}$ such that $\mathsf{MP}^{\mathsf{exp}}_{\mathsf{binary}}(\mathfrak{D})$ is* NExpTime*-hard.*

Proof. We use the deriver generated by the calculus for $\mathcal{ELI}$ shown in Figure 15, assuming that no sentence labels more than one vertex. The calculus is taken from [5] with small modifications:

- While the authors assume the input to use at most two conjuncts on the left-hand side, we do not impose such a restriction. It is not hard to see that lifting this restriction neither affects soundness nor completeness of the method, and does not have an impact on the complexity of the problem considered.

- The authors of [5] use a special set notation to represent n-ary conjunctions in inferred sentences. We just use n-ary conjunctions instead of sets, and assume that those are treated silently as sets when rules are applied: that is, conjunctions never have duplicate conjuncts and the order is not important.

To show NExpTime-hardness for this deriver, we describe a reduction from the exponential tiling problem with one initial tile. In the following, let T, $h, v \in T \times T$ and $t_0 \in T$ be such a tiling problem. For i, $1 \leq i \leq |h|$, let h_i denote the ith element of h, and for i, $1 \leq i \leq |v|$, let v_i denote the ith element of v. Furthermore, let $T_I \subseteq T$ be the set of tiles that occur at least once as first component in h or v.

For the first reduction, consider the number of sentences $m := m_1 + m_2 + m_3 + m4$, which will be the size of the proof, where

$$\begin{aligned} m_1 &:= 4n^2 + 6n + |h| + |v| + 7, \\ m_2 &:= (2^n - 1)^2(10n + 3) + 2(2^n - 1)(6n + 2) - (2n + 1), \\ m_3 &:= 6(2^n - 1) + 5(2^n - 1)^2 + 1, \text{ and} \\ m_4 &:= |T_I| + 7 \cdot (|v| + |h|) + 2, \end{aligned}$$

which can be computed in polynomial time when encoded in binary, and the TBox $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$ shown in Figure 16 and 18. In the reduction, we enforce the grid-shape of the tiling using the size restriction of the proof. This means that we have to be very careful with the use of auxiliary sentences: if those are not needed in every proof, this could introduce degrees of freedom in other parts of the proof, so that we fail to enforce this grid-shape. Therefore, we have to make sure that the number of auxiliary sentences used is always the same independent of the tiling solution. Specifically, not all of the vertical or horizontal tiling conditions might be relevant for a tiling solution, but the axioms used to encode them must be used in every proof of our entailment. $\mathcal{T}_1$ in Figure 16 contains the main reduction of the tiling problem, while $\mathcal{T}_2$ in Figure 16 makes sure that all of the sentences used to encode the tiling conditions are necessary for the proof.

We use concept names A_i, $\overline{A}_i$, B_i, $\overline{B}_i$ $1 \leq i \leq n$, to encode two binary counters for the x- and the y-coordinates in the tiling, respectively, where A_i denotes that the i-th bit has a value of 1, and $\overline{A}_i$ that the i-th bit has a value of 0, and correspondingly for B. For a number $i \in [0, 2^n - 1]$, we use $[A = i]$ as an abbreviation of the conjunction of concepts A_i, $\overline{A}_i$ that corresponds to a counter value of i, and correspondingly for $[B = i]$. We prove that $\mathcal{T} \models \mathsf{Start} \sqsubseteq \mathsf{Success}$ has a proof using $\leq m$ sentences iff the given tiling problem has a condition-complete solution.

($\Leftarrow$): Assume that the tiling problem has a solution f. We describe how a proof of $\mathcal{T} \models \mathsf{Start} \sqsubseteq \mathsf{Success}$ can be constructed. In this proof, we use concepts of the form

$$C(i, j) := f(i, j) \sqcap [A = i] \sqcap [B = j]$$

on the left-hand side of GCIs to identify individual grid cells (i, j), and the right-hand side of such GCIs to derive properties of these cells. The way the tiling is represented in the proof is sketched in Figure 17, which we will explain step-by-step. An arrow labeled with r_x from (i, j) to $(i + 1, j)$ represents a sentence of the form

$$C(i, j) \sqsubseteq \exists r_x.C(i + 1, j),$$

where $i \in [0, 2^n - 2]$ and $j \in [0, 2^n - 1]$, unless $i = j = 0$, in which case it represents a sentence of the form $\mathsf{Start} \sqsubseteq \exists r_x.C(1, 0)$. We abbreviate these sentences by $H(i, j)$, and do likewise for the sentences $V(i, j)$ connecting cells (i, j) and $(i, j + 1)$ using r_y in the vertical direction:

$$C(i, j) \sqsubseteq \exists r_y.C(i, j + 1),$$

where $i \in [0, 2^n - 1]$ and $j \in [0, 2^n - 2]$, unless $i = j = 0$ in which case it represents $\mathsf{Start} \sqsubseteq \exists r_y.C(0, 1)$.

The concepts X on the cells (i, j) shown in Figure 17 are those for which we will derive sentences $C(i, j) \sqsubseteq X$. For clarity, this figure does not include the sentences where X is one of the conjuncts of $C(i, j)$.

We construct the proof in four phases, using m_1, m_2, m_3 and m_4 sentences, respectively. Phases (I)–(III) encode the tiling solution based on $\mathcal{T}_1$ in a proof for $\mathsf{Start} \sqsubseteq \mathsf{Finish}$, Phase (IV)

$$\mathsf{Start} \sqsubseteq \overline{A}_i, \qquad \text{for all } 1 \leq i \leq n \tag{1}$$

$$\mathsf{Start} \sqsubseteq \overline{B}_i, \qquad \text{for all } 1 \leq i \leq n \tag{2}$$

$$\mathsf{Start} \sqsubseteq t_0 \tag{3}$$

$$\overline{A}_i \sqsubseteq \forall r_y.\overline{A_i} \qquad \text{for all } 1 \leq i \leq n \tag{4}$$

$$A_i \sqsubseteq \forall r_y.A_i \qquad \text{for all } 1 \leq i \leq n \tag{5}$$

$$\overline{B}_i \sqsubseteq \forall r_x.\overline{B_i} \qquad \text{for all } 1 \leq i \leq n \tag{6}$$

$$B_i \sqsubseteq \forall r_x.B_i \qquad \text{for all } 1 \leq i \leq n \tag{7}$$

$$\overline{A_i} \sqcap \bigsqcap_{j<i} A_j \sqsubseteq \forall r_x.A_i \qquad \text{for all } 1 \leq i \leq n \tag{8}$$

$$A_i \sqcap \bigsqcap_{j<i} A_j \sqsubseteq \forall r_x.\overline{A_i} \qquad \text{for all } 1 \leq i \leq n \tag{9}$$

$$\overline{A_i} \sqcap \overline{A_j} \sqsubseteq \forall r_x.\overline{A_i} \qquad \text{for all } 1 \leq i \leq n, 1 \leq j < i \tag{10}$$

$$A_i \sqcap \overline{A_j} \sqsubseteq \forall r_x.A_i \qquad \text{for all } 1 \leq i \leq n, 1 \leq j < i \tag{11}$$

$$\overline{B_i} \sqcap \bigsqcap_{j<i} B_j \sqsubseteq \forall r_y.B_i \qquad \text{for all } 1 \leq i \leq n \tag{12}$$

$$B_i \sqcap \bigsqcap_{j<i} B_j \sqsubseteq \forall r_y.\overline{B_i} \qquad \text{for all } 1 \leq i \leq n \tag{13}$$

$$\overline{B_i} \sqcap \overline{B_j} \sqsubseteq \forall r_x.\overline{B_i} \qquad \text{for all } 1 \leq i \leq n, 1 \leq j < i \tag{14}$$

$$B_i \sqcap \overline{B_j} \sqsubseteq \forall r_x.B_i \qquad \text{for all } 1 \leq i \leq n, 1 \leq j < i \tag{15}$$

$$t \sqsubseteq \exists r_x.t' \qquad \text{for all } \langle t, t' \rangle \in h \tag{16}$$

$$t \sqsubseteq \exists r_y.t' \qquad \text{for all } \langle t, t' \rangle \in v \tag{17}$$

$$t \sqcap [A = 2^n{-}1] \sqcap [B = 2^n{-}1] \sqsubseteq \mathsf{Finish} \qquad \text{for all } t \in T \tag{18}$$

$$\mathsf{Finish} \sqsubseteq \forall r_x^-.\mathsf{Finish}_x \tag{19}$$

$$\mathsf{Finish} \sqsubseteq \forall r_y^-.\mathsf{Finish}_y \tag{20}$$

$$\mathsf{Finish}_x \sqcap [B = 2^n - 1] \sqsubseteq \mathsf{Finish} \tag{21}$$

$$\mathsf{Finish}_y \sqcap [A = 2^n - 1] \sqsubseteq \mathsf{Finish} \tag{22}$$

$$\mathsf{Finish}_x \sqcap \mathsf{Finish}_y \sqsubseteq \mathsf{Finish} \tag{23}$$

Figure 16: The TBox $\mathcal{T}_1$ used in the reduction for Theorem 11.

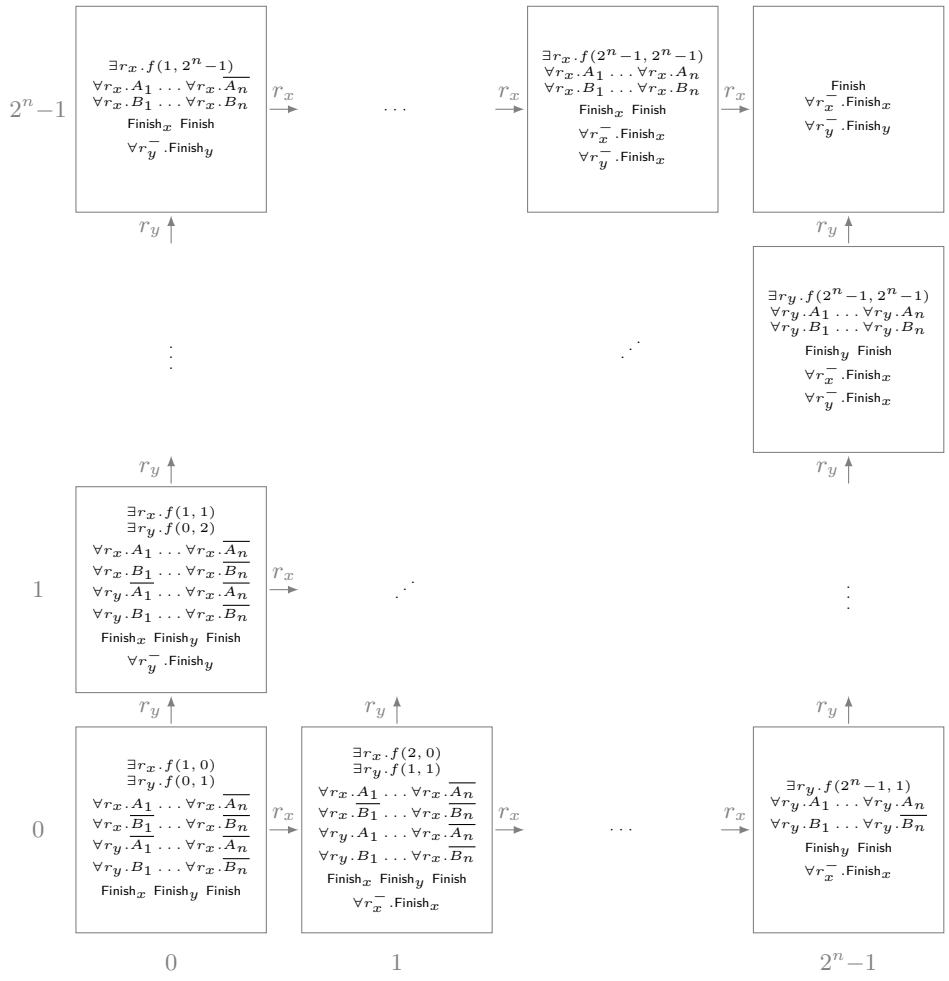

Figure 17: The sentences encoding a solution of the tiling problem.

takes care of $\mathcal{T}_2$, completing the proof to a proof for $\mathsf{Start} \sqsubseteq \mathsf{Success}$. In Phase (I), we add all sentences of $\mathcal{T}_1$ to the proof, except that for (18), we choose only the sentence with $t = f(2^n - 1, 2^n - 1)$, i.e., the tile on the upper right cell of the grid (see Figure 17). This amounts to exactly $m_1 = 4n^2 + 6n + |h| + |v| + 7$ sentences.

In Phase (II), we derive all necessary sentences $H(i,j)$ and $V(i,j)$ for the whole grid. For $H(i,j)$, we first consider the special case where $i = j = 0$. In this case, we have the sentences $\mathsf{Start} \sqsubseteq X$ for every X occurring in $C(0,0)$. Using one of the sentences (4)–(15) introduced in Phase (I), we can then use $\mathsf{CR2}$ to obtain exactly $2n$ sentences of the form

$$\mathsf{Start} \sqsubseteq \forall r_x.Y,$$

where Y ranges over the conjuncts in $[A = 1] \sqcap [B = 0]$, and one sentence

$$\mathsf{Start} \sqsubseteq \exists r_x.f(1,0)$$

via (16) since $\langle f(0,0), f(1,0)\rangle \in h$. Finally, we use a sequence of applications of $\mathsf{CR4}$ to obtain

$H(0,0)$ through $2n$ intermediate sentences that are also unique to this part of the proof. Thus, $H(0,0)$ is proven using $4n+1$ additional sentences.

For the remaining $H(i,j)$, we proceed similarly. We first derive exactly $2n+1$ sentences of the form

$$C(i,j) \sqsubseteq X,$$

where X ranges over the concept names in $C(i,j)$, using $\mathsf{CR1}$. These sentences may be shared with the proof of $V(i,j)$ in case that $j \leq 2^n - 2$. We note that $C(1,0)$ syntactically occurs in $H(0,0)$, which makes this application of $\mathsf{CR1}$ possible for $C(1,0)$. The remaining rule applications in this phase make sure that the remaining conceps $C(i,j)$ are all introduced, one after the other, so that $\mathsf{CR1}$ is applicable for all $C(i,j)$. Using one of the sentences (4)–(15) introduced in Phase (I), we can then use $\mathsf{CR2}$ to obtain exactly $2n$ sentences of the form

$$C(i,j) \sqsubseteq \forall r_x.Y,$$

where Y ranges over the conjuncts in $[A = i+1] \sqcap [B = j]$, and one sentence

$$C(i,j) \sqsubseteq \exists r_x.f(i+1,j)$$

via (16) since $\langle f(i,j), f(i+1,j)\rangle \in h$. These sentences are unique to $H(i,j)$. Using a sequence of applications of $\mathsf{CR4}$ we then obtain $H(i,j)$ through $2n$ intermediate sentences that are also unique to this part of the proof. Thus, for each $H(i,j)$, where we do not have $i = j = 0$, we use $6n+2$ sentences. The sentences $V(i,j)$ are obtained in a similar way, but using $2n+1$ sentences already used for $H(i,j)$. In total, Phase (II) uses

$$m_2 = (2^n-1)^2(10n+3) + 2(2^n-1)(6n+2) - (2n+1)$$

sentences, namely $(2n+1) + 2(4n+1) = (10n+3)$ for each inner cell $(i,j \in [0, 2^n-2])$, where the sentences derived using $\mathsf{CR1}$ are shared between $H(i,j)$ and $V(i,j)$, and

$$(2n+1) + (2n+1) + 2n = (6n+2)$$

for each "border" cell with either $i = 2^n - 1$ or $j = 2^n - 1$ (but not both), where only one of $H(i,j)$ or $V(i,j)$ needs to be derived. To get $H(0,0)$ and $V(0,0)$, we need $8n+2$ sentences in complete, which are $2n+1$ less than for the other cells, which is why this number is subtracted in the end.

In Phase (III), we derive all sentences of the form

$$C(i,j) \sqsubseteq \mathsf{Finish}.$$

We start with (18) and (19), which can be combined into

$$C(2^n-1, 2^n-1) \sqsubseteq \forall r_x^-.\mathsf{Finish}_x$$

using $\mathsf{CR2}$, which together with $H(2^n-2, 2^n-1)$ yields

$$C(2^n-2, 2^n-1) \sqsubseteq \mathsf{Finish}_x$$

by $\mathsf{CR3}$, and finally

$$C(2^n-2, 2^n-1) \sqsubseteq \mathsf{Finish}$$

by $\mathsf{CR2}$ and (21) together with the sentences $C(2^n-2, 2^n-1) \sqsubseteq B_i$, $1 \leq i \leq n$, from Phase (I). We can use this sentence in analogy to (18) to continue this process until we reach

$$\mathsf{Start} \sqsubseteq \exists r.t \qquad \text{for every } t \in T_I \tag{24}$$

$$t \sqsubseteq \forall r_x.H_i \qquad \text{for every } 1 \leq i \leq |h|, h_i = \langle t, t' \rangle \tag{25}$$

$$t \sqsubseteq \forall r_y.V_i \qquad \text{for every } 1 \leq i \leq |v|, v_i = \langle t, t' \rangle \tag{26}$$

$$H_i \sqcap t' \sqsubseteq \forall r_x^-.H_i \qquad \text{for every } 1 \leq i \leq |h|, h_i = \langle t, t' \rangle \tag{27}$$

$$V_i \sqcap t' \sqsubseteq \forall r_y^-.V_i \qquad \text{for every } 1 \leq i \leq |h|, v_i = \langle t, t' \rangle \tag{28}$$

$$H_i \sqsubseteq \forall r^-.H_i \qquad \text{for every } 1 \leq i \leq |h| \tag{29}$$

$$V_i \sqsubseteq \forall r^-.V_i \qquad \text{for every } 1 \leq i \leq |v| \tag{30}$$

$$\mathsf{Finish} \sqcap \bigsqcap_{i \leq |h|} H_i \sqcap \bigsqcap_{i \leq |v|} V_i \sqsubseteq \mathsf{Success} \tag{31}$$

Figure 18: TBox $\mathcal{T}_2$ enforcing the use of sentences (16) and (17).

$C(0, 2^n - 1) \sqsubseteq \mathsf{Finish}$. Thus, we have labeled all cells in the last row of Figure 17 with Finish, and we can proceed similarly with the last column.

For the remaining cells we can proceed similarly, but need to derive both Finish_x and Finish_y each time. Let $0 \leq i, j \leq 2^n - 2$, $i \neq 0$ or $j \neq 0$, and assume that we have already derived $C(i+1, j) \sqsubseteq \mathsf{Finish}$ and $C(i, j+1) \sqsubseteq \mathsf{Finish}$ for the right and upper neighbor of cell (i, j), respectively. We next derive

$$C(i+1, j) \sqsubseteq \forall r_x^-.\mathsf{Finish}_x \quad \text{and} \quad C(i, j+1) \sqsubseteq \forall r_y^-.\mathsf{Finish}_y$$

using CR2 and Sentences (19) and (20), respectively. We then obtain

$$C(i, j) \sqsubseteq \mathsf{Finish}_x \quad \text{and} \quad C(i, j) \sqsubseteq \mathsf{Finish}_y$$

via CR3 using $H(i, j)$ and $V(i, j)$, respectively. Finally, CR2 allows us to derive

$$C(i, j) \sqsubseteq \mathsf{Finish}$$

together with (23).

For the case of $i = j = 0$, similar applications result in the desired sentence $\mathsf{Start} \sqsubseteq \mathsf{Finish}$. The total number of new sentences Phase (III) is $m_3 = 6(2^n - 1) + 5(2^n - 1)^2$, namely 3 for each cell in the last row or the last column of the grid and 5 for each inner cell.

In Phase (IV), we now prove $\mathsf{Start} \sqsubseteq \mathsf{Success}$, making sure that really every sentence introduced in Phase (I) is required for the proof. For this, we need to prove $\mathsf{Start} \sqsubseteq H_i$ for every horizontal tiling condition h_i, and $\mathsf{Start} \sqsubseteq V_i$ for every vertical tiling condition v_i. We infer $\mathsf{Start} \sqsubseteq H_i$ by first inferring $t \sqsubseteq \exists r_x.(t' \sqcap H_i)$ for $h_i = \langle t, t' \rangle$ using (25) and (16), then $t \sqsubseteq H_i$ using (27), $t \sqsubseteq \forall r^-.H_i$ using (29), and finally $\mathsf{Start} \sqsubseteq H_i$ using (24). For $\mathsf{Start} \sqsubseteq V_i$, we proceed similarly. In complete, for both $\mathsf{Start} \sqsubseteq H_i$ and $\mathsf{Start} \sqsubseteq V_i$, $|T_I| + 7 \cdot (|v| + |h|)$ sentences are needed (the sentences (24) only need to be counted once).Note that also every sentence of the form (16) and (17) must be used in this proof. The final proof for $\mathsf{Start} \sqsubseteq \mathsf{Success}$ needs 2 additional sentences ((31) and the conclusion), so that Phase (IV) needs $m_4 = |T_I| + 7 \cdot (|v| + |h|) + 2$ additional sentences in complete. We obtain that the complete proof for $\mathsf{Start} \sqsubseteq \mathsf{Success}$ has a size of m.

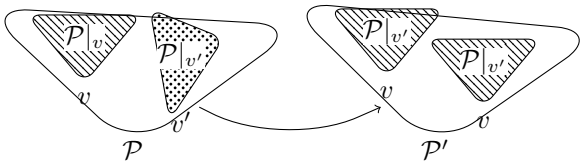

Figure 19: Transformation made in the proof of Lemma 25 for $v, v' \in V$ with $\ell(v) = \ell(v')$

($\Rightarrow$): Assume that there is a proof of $\mathcal{T} \models \mathsf{Start} \sqsubseteq \mathsf{Finish}$ that uses $m = m_1 + m_2 + m_3 + m_4$ sentences or less. Any such proof must follow the general scheme laid out above, since one cannot get from $C(2^n - 1, 2^n - 1) \sqsubseteq \mathsf{Finish}$ to $C(0, 0) \sqsubseteq \mathsf{Finish}$ without going through all tiles $(i, j) \in [0, 2^n - 1] \times [0, 2^n - 1]$. Hence, one needs to derive appropriate sentences $H(i, j)$ and $V(i, j)$ first, for some choice of tile types. It can also be verified that none of the individual derivation steps can be achieved using fewer intermediate sentences. Spcifically, every sentence (16) and (17) must be used exactly once in such a proof, due to Phase (IV). Thus, the tight bound of m requires that, for each cell (i, j), at most one concept of the form $C(i, j)$ occurs in the proof, i.e., each cell is assigned a unique tile $f(i, j) \in T$. This yields a solution of the tiling problem due to the horizontal and vertical compatibility relations encoded in (16) and (17). □

Now we can proceed to the tree-shaped proof representation (see Table 2).

Lemma 15. *There is a tree proof $\mathcal{P}$ for $\mathcal{T} \models \eta$ that is admissible w.r.t. $\mathcal{D} = \mathfrak{D}(\mathcal{T}, \eta)$ with $\mathsf{vert}(\mathcal{P}) \leq m$ iff $\mathcal{D}^*$ contains a hyperpath $\mathcal{P}'$ from s to a vertex t labeled with η with $W_{\mathcal{P}'}(\mathcal{P}') \leq m$.*

We first prove an auxiliary result that allows us to restrict the search to tree proofs of a specific form, in a way similar to Lemma 8. Given a tree proof $\mathcal{P} = (V, E, \ell)$ and $v \in V$, we denote by $\mathcal{P}|_v$ the subtree of $\mathcal{P}$ with root v (which is a proof of $\ell(v)$). Moreover, $\mathcal{P}$ is called *regular* if, whenever $\ell(v) = \ell(v')$, then $\mathcal{P}|_v$ and $\mathcal{P}|_{v'}$ are isomorphic.

Lemma 25. *There exists a minimal tree proof for $\mathcal{T} \models \eta$ in $\mathfrak{D}$ that is regular.*

Proof. Let $\mathcal{P} = (V, E, \ell)$ be a minimal tree proof that is not regular. Then there are $v, v' \in V$ such that $\mathcal{P}|_v$ and $\mathcal{P}|_{v'}$ are not isomorphic. We choose v such that $\mathsf{vert}(\mathcal{P}|_v)$ is minimal among all possible choices of v and v', and therefore $\mathsf{vert}(\mathcal{P}|_v) \leq \mathsf{vert}(\mathcal{P}|_{v'})$. We now follow a similar strategy as in the proof of Lemma 8, but instead of merging v and v' we replace $\mathcal{P}|_{v'}$ in $\mathcal{P}$ by a copy of $\mathcal{P}|_v$. By similar arguments as for Lemma 8, this results in another proof $\mathcal{P}'$ that is admissible w.r.t. $\mathfrak{D}(\mathcal{T}, \eta)$, which is clearly also still a tree. Since $\mathcal{P}$ is minimal, it is impossible that $\mathsf{vert}(\mathcal{P}|_v) < \mathsf{vert}(\mathcal{P}|_{v'})$, which means that $\mathsf{vert}(\mathcal{P}') = \mathsf{vert}(\mathcal{P})$.

We can continue in this fashion (depicted in Figure 19) until we obtain a regular proof with the same number of vertices as $\mathcal{P}$. This process terminates since by our choice of v the subtree $\mathcal{P}_v$ cannot contain another vertex that violates the regularity condition, and thus by replacing $\mathcal{P}|_{v'}$ with $\mathcal{P}|_v$ we always decrease the number of pairs (v, v') that violate the regularity condition. □

Hence, in the following we can focus on regular tree proofs. Next, we draw a connection between *non-redundant* proofs and hyperpaths.

Lemma 26. *A derivation structure $\mathcal{P} = (V, E, \ell)$ over $\mathcal{T}$ is a non-redundant proof for $\mathcal{T} \models \eta$ iff $\mathcal{P}^*$ is a hyperpath from s to a vertex v_η s.t. $\ell(v_\eta) = \eta$.*

Proof. In order to show that $\mathcal{P}^*$ is a hyperpath, we note that, trivially, $s, v_\eta \in V^*$. Now, for any $v \in V$, we show that v is reachable from s in $\mathcal{P}^*$. All leaves of $\mathcal{P}$ and those vertices v with an "empty" incoming edge $(\emptyset, v)$ are reachable from s in $\mathcal{P}^*$ by the edges we added to E^*. Moreover, all other vertices are reachable from at least one such leaf. There can be no cycle through s, because it has only outgoing edges. Hence, any cycle in $\mathcal{P}^*$ would contradict the acyclicity of $\mathcal{P}$. Finally, minimality of $\mathcal{P}^*$ follows from non-redundancy of $\mathcal{P}$.

For the opposite direction, if the hyperpath $\mathcal{P}^*$ from s to v_η contains another sink, we can build a smaller hypergraph $\mathcal{P}'$ by exhaustively removing sinks $v \neq v_\eta$ and all their incoming edges. This hypergraph is also a hyperpath from s to v_η, which contradicts the minimality condition in Definition 14 for $\mathcal{P}^*$. Acyclicity of the proof $\mathcal{P}$ follows from the acyclicity of $\mathcal{P}^*$. Non-redundancy follows from the minimality of hyperpaths. □

We can now finish the proof of Lemma 15 by noting that, for any hyperpath $\mathcal{P}^*$ as in Lemma 26, we can unravel the corresponding proof $\mathcal{P}$ into a regular tree proof $\mathcal{P}'$ (thereby recursively introducing copies of subtrees that are used in multiple inference steps). A homomorphism $h\colon \mathcal{P}' \to \mathcal{D}$ can be defined accordingly. Since the recursive definition of $W_{\mathcal{P}^*}(\mathcal{P}^*)$ adds a summand of 1 for every time a vertex occurs in an edge in $\mathcal{P}^*$, we have $\mathsf{vert}(\mathcal{P}') = W_{\mathcal{P}^*}(\mathcal{P}^*)$. Conversely, for a regular tree proof $\mathcal{P}'$ with a homomorphism $h\colon \mathcal{P} \to \mathcal{D}$, we can find a corresponding non-redundant proof $\mathcal{P}$ in $\mathcal{D}$ and obtain a hyperpath $\mathcal{P}^*$ with $W_{\mathcal{P}^*}(\mathcal{P}^*) = \mathsf{vert}(\mathcal{P}')$ by the same arguments as above.

We can show a matching lower bound by reducing a standard reasoning problem to $\mathsf{MTP}(\mathfrak{D})$.

Theorem 17. *There exists a polynomial deriver $\mathfrak{D}$ such that $\mathsf{MTP}^{\mathsf{poly}}_{\mathsf{unary}}(\mathfrak{D})$ is* P*-hard under* LogSpace*-reductions.*

Proof. We reduce the entailment problem of a GCI $A \sqsubseteq B$ from an $\mathcal{EL}$-TBox $\mathcal{T}$ in, which is P-hard already when A and B are concept names and $\mathcal{T}$ does not contain existential restrictions $\exists r.D$, because then $\mathcal{EL}$ is equivalent to propositional Horn logic (see [14] for details). We will reduce this problem to $\mathsf{MTP}(\mathrm{ELK})$, where n is encoded in unary and we can restrict our attention to the rules from Figure 1 that do not use existential restrictions. The challenge is to find $\mathcal{T}'$ and n such that $\mathcal{T}' \models A \sqsubseteq B$ holds and there is a tree proof $\mathcal{P}$ for $A \sqsubseteq B$ over $\mathcal{T}'$ with $\mathsf{vert}(\mathcal{P}) \leq n$ iff $\mathcal{T} \models A \sqsubseteq B$. According to [25], the size of $\mathrm{ELK}(\mathcal{T}, A \sqsubseteq B)$ is bounded polynomially in $|\mathcal{T}|$. However, for our purposes, we need to show that the minimal tree proof admissible w.r.t. this derivation structure is similarly bounded.

Claim. *There is a polynomial p such that, for any TBox $\mathcal{T}$ and GCI $A \sqsubseteq B$ as above, the minimal tree proof for $A \sqsubseteq B$ over $\mathcal{T}$ that is admissible w.r.t. $\mathrm{ELK}(\mathcal{T}, A \sqsubseteq B)$ satisfies $\mathsf{vert}(\mathcal{P}) \leq p(|\mathcal{T}|)$.*

Proof of claim. Since $\mathcal{T}$ does not contain existential restrictions, we can assume that all GCIs in $\mathcal{T}$ are of the form $A_1 \sqcap \cdots \sqcap A_m \sqsubseteq B_1 \sqcap \cdots \sqcap B_k$ (*), where $A_1, \ldots, A_m, B_1, \ldots, B_k$ are concept names, the order and multiplicity of concept names is irrelevant, and we view $\top$ as the "empty" conjunction. The basic idea is to use *unit propagation* [14] as for Horn clauses to derive all consequences, including $A \sqsubseteq B$. That is, starting from GCIs of the form $A \sqsubseteq B_1 \sqcap \ldots B_k$, we "propagate" the GCIs $A \sqsubseteq B_j$ to other GCIs where B_j occurs on the left-hand side. Once we have derived GCIs $A \sqsubseteq A_i$ for all concept names A_i on the left hand side of a GCI $A_1 \sqcap \cdots \sqcap A_m \sqsubseteq C_1 \sqcap \ldots C_k$, then we can also "propagate" the GCIs $A \sqsubseteq C_j$, and so on until we reach $A \sqsubseteq B$. However, we have to choose a strategy that results in a polynomially sized proof tree, which means that we need to avoid using a derived GCI in several following inference steps to avoid duplicating the whole subproof in the tree.

We use the following strategy to compute a tree proof for $\mathcal{T} \models A \sqsubseteq B$:

1. Initialize the conjunction $X := A$ of already "propagated" concept names, for which we can derive the GCI $A \sqsubseteq X$ using the inference rule R_0.
2. Select all GCIs of the form (*) from $\mathcal{T}$ for which all of $A_1, \dots, A_m$ occur in X.
3. Extend these GCIs to $X \sqsubseteq B_1 \sqcap \cdots \sqcap B_k$ by first deriving $X \sqsubseteq X$ via R_0, from this $X \sqsubseteq A_1 \sqcap \cdots \sqcap A_m$ by splitting X on the right-hand side using $\mathsf{R}_\sqcap^-$, and then combining this GCI with the original GCI of the form (*) by $\mathsf{R}_\sqsubseteq$.
4. Combine all selected GCIs, which now have the same left-hand side X, into a single sentence $X \sqsubseteq Y$, where Y is the conjunction of all concept names B_j on the right-hand sides of the selected GCIs, using the rule $\mathsf{R}_\sqcap^+$.
5. Combine this GCI with $X \sqsubseteq X$ (from R_0) to obtain $X \sqsubseteq X \sqcap Y$ via $\mathsf{R}_\sqcap^+$.
6. Derive $A \sqsubseteq X \sqcap Y$ from $A \sqsubseteq X$ and $X \sqsubseteq X \sqcap Y$ by $\mathsf{R}_\sqsubseteq$.
7. Continue with Step 2 $X \sqcap Y$ in place of X.

It can be checked that the resulting tree proof derives most sentences only once, except for ones that can be derived using a linear proof (where each hyperedge (S, d) satisfies $|S| = 1$) of polynomial size. For example, the GCI $X \sqsubseteq A_1 \sqcap \cdots \sqcap A_m$ may be derived several times if several GCIs in $\mathcal{T}$ have the same left-hand side, but this cannot cause an exponential growth of this tree w.r.t. the size of $\mathcal{T}$.

Taking the polynomial p from our claim, we set $n := p(|\mathcal{T}|)$ and

$$\mathcal{T}' := \mathcal{T} \cup \{A \sqsubseteq A_1,\ A_1 \sqsubseteq A_2, \dots, A_n \sqsubseteq B\},$$

where $A_1, \dots, A_n$ are concept names not occurring in $\mathcal{T}$. Clearly, $\mathcal{T}' \models A \sqsubseteq B$ and the existence of a tree proof $\mathcal{P}$ for $A \sqsubseteq B$ over $\mathcal{T}'$ with $\mathsf{vert}(\mathcal{P}) \leq n$ is equivalent to $\mathcal{T} \models A \sqsubseteq B$, since any proof that uses the new concept names must use at least $n+1$ sentences. Moreover, we can compute n (in binary representation) and output it in unary representation using a logarithmically space-bounded Turing machine, and similarly for $\mathcal{T}'$. Hence, the above construction constitutes the desired LogSpace-reduction. □

We conjecture that an ExpTime-lower bound for $\mathsf{MTP}^{\mathsf{exp}}_{\mathsf{binary}}(\mathfrak{D})$ could be shown in a similar fashion, by using a binary counter to enforce an artificial proof of $A \sqsubseteq B$ that has exponentially many vertices. However, so far we were not successful in finding a concrete exponential deriver where we could show an exponential upper bound on the size of minimal tree proofs.

We show NP-hardness for finding a tree proof (of size less than n) for ExpTime-complete $\mathcal{ELI}$, the extension of $\mathcal{EL}$ by inverse roles.

Theorem 18. *There is an exponential deriver $\mathfrak{D}$ such that $\mathsf{MTP}^{\mathsf{exp}}_{\mathsf{unary}}(\mathfrak{D})$ is* NP*-hard.*

Proof. We show hardness using the deriver that uses the $\mathcal{ELI}$-calculus shown in Figure 15, thus showing that the problem is already hard for that specific deriver. We will provide a translation of the SAT problem to $\mathsf{MTP}^{\mathsf{exp}}_{\mathsf{unary}}(\mathfrak{D})$. In SAT, we are given a Boolean formula of the form $\varphi = c_1 \wedge \cdots \wedge c_m$ over a finite set of variables $\{x_1, \dots, x_n\}$, where each c_i is a disjunction of literals (variables or its negations), and we want to decide whether φ is decidable.

Without loss of generality, we assume that for every variable x_i, $1 \leq i \leq n$, we have a corresponding clause $x_i \vee \neg x_i$. Given such a formula φ, we construct a TBox $\mathcal{T}_\varphi$ s.t. $\mathcal{T}_\varphi \models A \sqsubseteq F$ can be shown with a tree proof of size $2n + 4m + 9$ iff φ is satisfiable.

For every variable x_i, $1 \leq i \leq n$, we use the following sentences which are supposed to "guess" the value of x_i:

$$A \sqsubseteq \forall r.X_i, \qquad A \sqsubseteq \forall r.\overline{X}_i.$$

For every clause c_j, $1 \leq j \leq m$, we add

$$\begin{aligned} X_i &\sqsubseteq \forall r.C_j, && \text{for every literal of the form } x_i \in c_j, \text{ and} \\ \overline{X}_i &\sqsubseteq \forall r.C_j, && \text{for every literal of the form } \neg x_i \in c_j. \end{aligned}$$

In addition, we use the following sentences whose meaning will become clear below.

$$\begin{aligned} A &\sqsubseteq \exists r.B, & B &\sqsubseteq \exists r.\top, \\ C_1 \sqcap \ldots \sqcap C_m &\sqsubseteq \forall r^-.F, & F &\sqsubseteq \forall r^-.F \end{aligned}$$

We show that φ is satisfiable iff $\mathcal{T}_\varphi \models A \sqsubseteq F$ can be shown using a tree proof with $2n+4m+9$ vertices.

($\Rightarrow$): We first show that, given an assignment $v\colon \{x_1, \ldots, x_n\} \to \{0,1\}$ that makes φ true, we can construct a tree proof for $\mathcal{T}_\varphi \models A \sqsubseteq F$ with $2n + 4m + 9$ vertices. The assignment v can be expressed as a conjunction over the set

$$\mathcal{V} = \{X_i \mid 1 \leq i \leq n,\ v(x_i) = 1\} \cup \{\overline{X}_i \mid 1 \leq i \leq n,\ v(x_i) = 0\}.$$

In the proof, we step-wise infer the sentence $A \sqsubseteq \exists r.(B \sqcap \bigsqcap \mathcal{V})$ using rule $\mathsf{CR4}$, the sentence $A \sqsubseteq \exists r.B$, and for every $1 \leq i \leq n$, the respective sentence $A \sqsubseteq \forall r.X_i$ or $A \sqsubseteq \forall r.\overline{X}_i$, depending on whether $v(x_i) = 1$ or $v(x_i) = 0$. This creates a subtree of the final proof of size $2n + 1$.

Since $A \sqsubseteq \exists r.(B \sqcap \bigsqcap \mathcal{V})$ can be derived, we can use $\mathsf{CR1}$ to infer $B \sqcap \bigsqcap \mathcal{V} \sqsubseteq B$. Using $\mathsf{CR2}$ on this sentence and $B \sqsubseteq \exists r.\top \in \mathcal{T}_\varphi$, we infer $B \sqcap \bigsqcap \mathcal{V} \sqsubseteq \exists r.\top$. Starting from this sentence, we step-wise use $\mathsf{CR4}$ to infer

$$B \sqcap \bigsqcap \mathcal{V} \sqsubseteq \exists r.(C_1 \sqcap \ldots \sqcap C_m),$$

where in each step, we use for a different clause C_j, $1 \leq j \leq m$, some sentence $\overline{X}_i \sqsubseteq \forall r.C_j \in \mathcal{T}_\varphi$ or $X_i \sqsubseteq \forall r.C_j \in \mathcal{T}_\varphi$, depending on whether $v(x_i) = 0$ or $v(x_i) = 1$. This construction results in another subtree of the proof which has a size of $4m+3$ (note that for each step, we need to derive $B \sqcap \bigsqcap \mathcal{V} \sqsubseteq X_i$ (or $B \sqcap \bigsqcap \mathcal{V} \sqsubseteq \overline{X}_i$) and $B \sqcap \bigsqcap \mathcal{V} \sqsubseteq \forall r.C_j$ as intermediate axioms to be able to apply $\mathsf{CR4}$). We further extend this tree by inferring $B \sqcap \bigsqcap \mathcal{V} \sqsubseteq F$ using $C_1 \sqcap \ldots \sqcap C_m \sqsubseteq \forall r^-.F \in \mathcal{T}_\varphi$ and $\mathsf{CR3}$. This uses another 2 vertices, so that the resulting sub-tree of the proof has size $4m + 5$. Finally, we use $\mathsf{CR2}$ on this sentence and $F \sqsubseteq \forall r^-.F \in \mathcal{T}_\varphi$, resulting in a sub-tree of size $4m + 7$ with its root labeled $B \sqcap \bigsqcap \mathcal{V} \sqsubseteq \forall r^-.F$.

We now have one sub-tree of size $2n + 1$ with its root labelled $A \sqsubseteq \exists r.(B \sqcap \bigsqcap \mathcal{V})$, and one subtree of size $4m + 7$ with its root labeled with $B \sqcap \bigsqcap \mathcal{V} \sqsubseteq \forall r^-.F$. We connect both subtrees using $\mathsf{CR3}$, resulting in a proof for $A \sqsubseteq F$ of size $2n + 4m + 9$.

($\Leftarrow$): For the other direction, assume that $\mathcal{T}_\varphi \models A \sqsubseteq F$ can be shown using a proof of size $2n + 4m + 9$. We demonstrate how we can construct from such proof an assignment $v\colon \{x_1, \ldots, x_n\} \to \{0,1\}$ that satisfies φ. A proof of $\mathcal{T}_\varphi \models A \sqsubseteq F$ needs to infer at some point sentences of the following form:

$$\begin{aligned} A &\sqsubseteq \exists r.C, & C &\sqsubseteq \forall r^-.F, \\ C &\sqsubseteq F, & C &\sqsubseteq \exists r.(C_1 \sqcap \ldots \sqcap C_m). \end{aligned}$$

The sentences in $\mathcal{T}_\varphi$ only allow to infer $A \sqsubseteq \exists r.C$ provided that C is a conjunction of B with concept names from the set $\{X_1, \overline{X}_1, \ldots, X_n, \overline{X}_n\}$. The shape of C is further restricted by the fact that we can only infer $C \sqsubseteq \exists r.(C_1 \sqcap \ldots \sqcap C_m)$ provided that C contains B as conjunct, and at least one conjunct of the form X_i or $\overline{X}_i$ for each variable x_i, $1 \leq i \leq n$. The latter is enforced since for every such variable x_i, there is some clause $c_j = x_i \vee \neg x_i$, so that C_j occurs under a universal restriction only in the sentences $X_i \sqsubseteq \forall r.C_j$ and $\overline{X}_i \sqsubseteq \forall r.C_j$ in $\mathcal{T}_\varphi$.

A proof for $C \sqsubseteq \exists r.(C_1 \sqcap \ldots \sqcap C_m)$ requires at least $4m + 3$ vertices (1 for $C \sqsubseteq B$, 2 for $C \sqsubseteq \exists r.\top$, and then we need to derive each $C \sqsubseteq \forall r.C_j$ by $\mathsf{CR2}$ from $C \sqsubseteq X_i$ / $C \sqsubseteq \overline{X}_i$ and $X_i \sqsubseteq \forall r.C_j$ / $\overline{X}_i \sqsubseteq \forall r.C_j$, and another m steps of $\mathsf{CR4}$ to obtain the goal). Inferring from this $C \sqsubseteq F$ and then $C \sqsubseteq \forall r^-.F$ requires 2 more vertices each, so that at this point, at least $4m + 7$ vertices of the proof have been assigned. As we still need to infer $A \sqsubseteq F$, this leaves $2n + 1$ vertices for proving $A \sqsubseteq \exists r.C$. For this, we need to use the sentence $A \sqsubseteq \exists r.B$ once, and 2 vertices for every additional conjunct in C, meaning that C has at most $n+1$ conjuncts. C is a conjunction of B and at least one of X_i and $\overline{X}_i$ for each x_i $1 \leq i \leq n$. As there are at most n conjuncts, this means that for every x_i, $1 \leq i \leq n$, C contains exactly one conjunct, based on which we define the valuation v: $v(x_i) = 0$ if $\overline{X}_i$ occurs as conjunct, and $v(x_i) = 1$ if X_i occurs as conjunct. We can furthermore show that v satisfies each clause c_j, $1 \leq j \leq m$: the proof used some sentence of the form $X_i \sqsubseteq \forall r.C_j$ or $\overline{X}_i \sqsubseteq \forall r.C_j$; in the first case, we have $v(x_i) = 1$ by definition of v (X_i must be a conjunct of C), and $x_i \in c_j$ by construction of $\mathcal{T}_\varphi$, and in the latter, we have $v(x_i) = 0$ by the definition of v and $\neg x_i \in c_j$ by the construction of $\mathcal{T}_\varphi$. It follows that v is a satisfying assignment of φ.

We have shown that φ is satisfiable iff $\mathcal{T}_\varphi \models A \sqsubseteq F$ has a tree proof of size at most $2n+4m+9$, which means that SAT can be reduced to $\mathsf{MTP}^{\mathsf{exp}}_{\mathsf{unary}}(\mathfrak{D})$. Consequently, $\mathsf{MTP}^{\mathsf{exp}}_{\mathsf{unary}}(\mathfrak{D})$ is NP-hard. □

We can show the remaining upper bound in a similar way as before.

Theorem 19. $\mathsf{MTP}^{\mathsf{exp}}_{\mathsf{unary}}(\mathfrak{D})$ *is in* NP.

Proof. We can use the same arguments as in the proof of Theorem 10, but allow to guess vertices of $\mathfrak{D}(\mathcal{T}, \eta)$ several times and additionally check that we obtain a tree proof. □

Finally, in the next theorem we argue that soundness and completeness of *the forgetting-based approach* (FBA) rely on soundness and completeness of the constituent components.

Theorem 21. *Let FBA use a sound and complete reasoner to compute justifications and a sound forgetting tool. Given a sentence η and a TBox $\mathcal{T}$, FBA produces a proof $\mathcal{P}$ iff $\mathcal{T} \models \eta$.*

Proof. It is not hard to see that the hypergraph $\mathcal{D} = (V, E, \ell)$ constructed this way is indeed a proof according to Definition 4. First, $\mathcal{D}$ is a *derivation structure*: it is *grounded* since every vertex $v \in V$ without an incoming edge is labeled with an sentence from $\mathcal{J} \subseteq \mathcal{T}$; and it is *sound* since for every edge $(S, d) \in E$, $\{\ell(s) \mid s \in S\}$ is a justification for $\ell(d)$. Second, every $(S, d) \in E$ is obtained from a pair $\mathcal{T}_i, \mathcal{T}_{i+1}$ from our generated sequence of TBoxes, that is, $\{\ell(s) \mid s \in S\} \subseteq \mathcal{T}_i$ and $\ell(d) \in \mathcal{T}_{i+1}$, which means that $\mathcal{D}$ is an *acyclic* derivation structure. It follows that $\mathcal{D}$ is a proof according to Definition 4. Thus, by Lemma 23, FBA is sound. Moreover, even if the forgetting process itself is not complete, FBA always generates a proof. □

B Examples of Generated Proofs

We finally discuss a few more examples that illustrate the differences between the three proof generators compared in Section 5. The images in Figures 12 and 20–22 were generated automatically from our datasets. Recall that hyperedges are denoted by blue boxes labelled with the type of inference rule (for ELK) or the forgotten symbol(s) (for FBA). Figures 20 and 21 depict two cases where the ELK proofs are arguably better than the ones generated by FBA, while Figures 12 and 22 illustrate advantages of FBA. Note that we chose quite small proofs as examples, because larger ones would not easily fit on these pages.

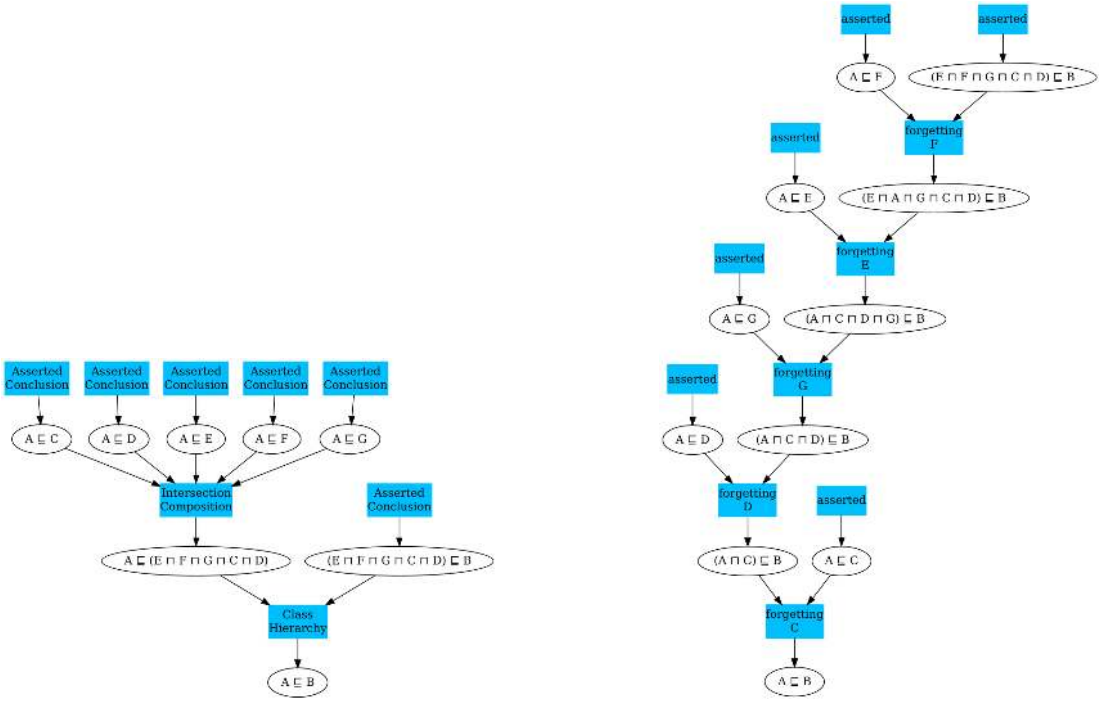

Figure 20: A proof generated by ELK (left) vs. one for the same entailment generated via FBA (right; using either LETHE or FAME). Using the n-ary "Intersection Composition" rule, ELK allows to formulate a cleaner proof, while the iterative forgetting approach requires separate steps to eliminate each symbol.

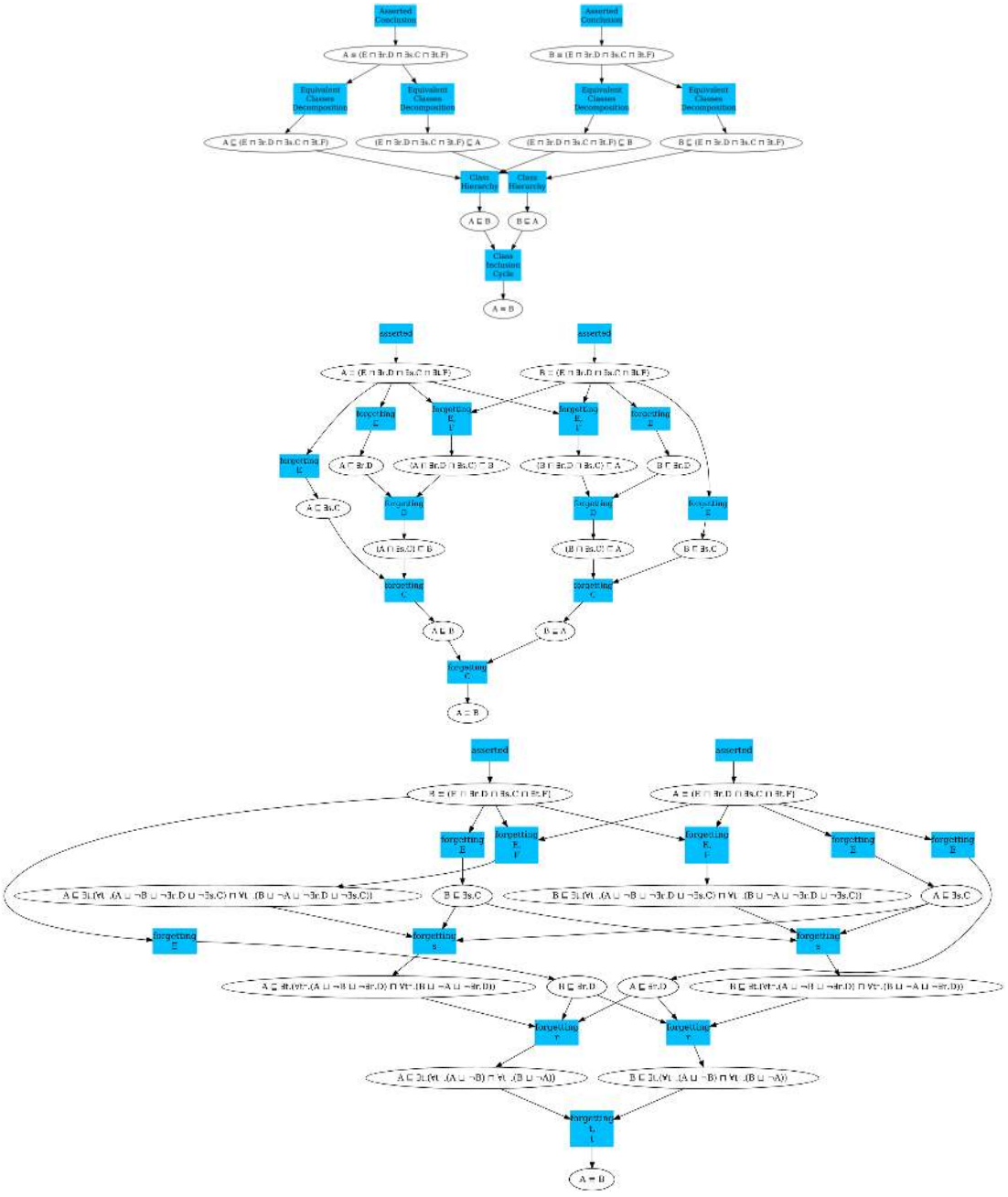

Figure 21: Again, ELK (top) allows to eliminate several symbols at the same time, while the forgetting-based proofs (center: LETHE, bottom: FAME) are more complicated. FAME makes extensive use of the expressivity of $\mathcal{ALCOI}$ because it needs to preserve all $\mathcal{ALCOI}$-consequences, which is a disadvantage in this case.

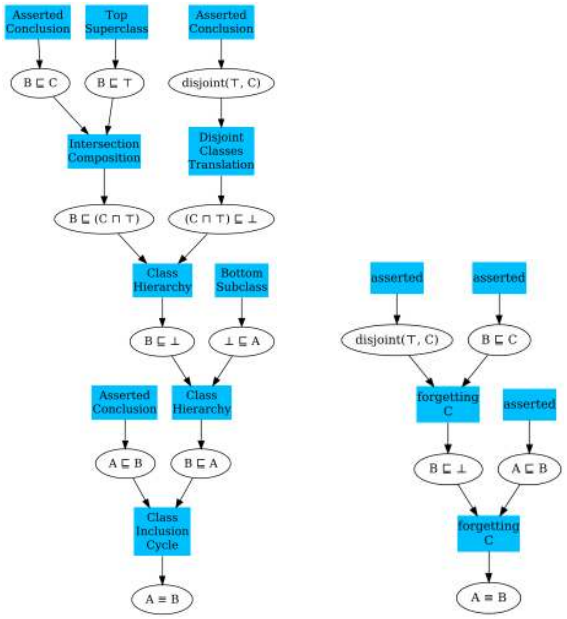

Figure 22: Elk (left) makes many simple steps, which are collapsed by FBA (right) into a simpler proof. Assuming basic knowledge about description logics, the latter is arguably preferable.