Abstract. Description Logic Programs (DLP) have been described as a description logic (DL) that is in the “expressive intersection” of DL and datalog. This is a very weak guideline for defining DLP in a way that can be claimed to be optimal or maximal in any sense. Moreover, other DL fragments such as EL and Horn-SHIQ have also been “expressed” using datalog. So is DLP just one out of many equal DLs in this “expressive intersection”? This paper attempts to clarify these issues by characterising DLP with various design principles that clearly distinguish it from other approaches. A consequent application of the introduced principles leads to the definition of a significantly larger variant of DLP which we show to be maximal in a concrete sense. While DLP is used as a concrete (and remarkably complex) example in this paper, we argue that similar approaches can be applied to find canonical definitions for other fragments of logical languages, such as the “maximal” fragment of SWRL rules that can be expressed in the DL SROIQ.

1 Introduction

Description Logic Programs (DLP) were introduced as a family of fragments of description logic (DL) that can be expressed in first-order Horn-logic [1,2]. Since common reasoning tasks are still undecidable for first-order Horn-logic, its function-free fragment datalog is of particular interest, and the term “DLP” today is most commonly used to refer to tractable DLs that can be translated to equisatisfiable datalog.

This statement is slightly more concrete than describing DLP as a subset of the “expressive intersection” of DL and datalog [1], but it is still insufficient to characterise DLP. In particular, it is well-known that other tractable DLs such as $\mathcal{EL}$ can also be translated to equisatisfiable datalog programs [3,4]. It is known that the union of DLP and $\mathcal{EL}$ is an intractable DL (for some discussion, see [3]), but one may still wonder whether DLP is merely one among several equivalent subsets of the “expressive intersection” of DL and datalog.

But tractability was not among the original design goals of DLP, and one might also weaken this principle to require merely a polytime transformation to datalog. Since reasoning in datalog is still ExpTime complete, this would not preclude intractable logics. Could the union of DLP and $\mathcal{EL}$ then be considered as an extended version of DLP?

* There have been non-editorial updates since version 1.0; an overview of the changes is given at the end of the paper.
Possibly yes, since it is contained in the DL Horn-$SHIQ$ for which a satisfiability-preserving datalog transformation is known \cite{5}. However, $\mathcal{EL}$ and DLP can be translated to datalog axiom-by-axiom, i.e. in a modular fashion, while the known datalog transformation for Horn-$SHIQ$ needs to consider the whole knowledge base. But how can we be sure that there is no simpler transformation given that both data-complexity and combined complexity of datalog and Horn-$SHIQ$ agree? The answer is given in Proposition 1 below.

In any case, it is obvious that the design principles for DLP – but also for $\mathcal{EL}$ and Horn-$SHIQ$ – are not sufficiently well articulated to clarify the distinction between those formalisms. This paper thus approaches an explicit characterisation of DLP, not in terms of concrete syntax but in terms of general design principles, which captures the specifics of the known DLP for datalog. An essential principle is structurality of the language: a formula should be in DLP based on its term structure, not based on concrete entity names that it uses. Moreover, we ask whether DLP could be defined as a larger, or even as the largest, DL language that satisfies our design principles. A significantly larger variant of DLP is introduced, and its translation to datalog is provided. This paper does not give a conclusive answer on whether or not this extended DLP is the largest possible, but we conjecture this to be true, and we sketch the proof currently under construction.

This paper begins with some preliminary definitions in Section 2. In Section 3, we discuss the problems of characterising DLP and provide some fundamental results. Section 4 provides a simplified version of the main results by restricting attention to the smaller description logic $ALC$, where it is significantly easier to define a DLP fragment and prove its maximality. These simplifications allow us to outline the general proof structure and some relevant methods, but they do not cover all relevant parts of earlier DLP definitions nor all relevant proof techniques needed in the general case. A full definition for an extended language $DLP$ is then provided in Section 5. In Section 6, we show how $DLP$ can be expressed using datalog. Section 7 discussed some important model-theoretic constructions for characterising fragments of first-order logic that can be expressed in datalog. These constructions are then used as a basis for showing maximality of $DLP$ in Section 8. Section 9 provides a short outlook on further application areas for the presented approaches.

2 Preliminaries

We consider the well-known description logic $SROIQ$ as defined in \cite{6}. As we are mainly interested in a $SROIQ$ish syntax of the language to be defined, we consider $SROIQ^{\text{free}}$ denoting $SROIQ$ without simplicity and regularity constraints. In particular $SROIQ^{\text{free}}$ allows arbitrary role inclusion axioms $R_1 \circ \ldots \circ R_n \sqsubseteq R$. Clearly, the semantics of $SROIQ^{\text{free}}$ follows mutatis mutandis from that of $SROIQ$. As usual, $SROIQ^{\text{free}}$ knowledge bases are defined over finite sets of individual names $I$, concept names $A$, and role names $R$. For the purpose of this paper, we assume that $R$ includes inverse roles, i.e. that for each $R \in R$ there is an inverse $\text{Inv}(R) \in R$ such that $\text{Inv} : R \to R$ is bijective, symmetric, and irreflexive. We call $\mathcal{I} = (I, A, R)$ signature, and all signa-
tures are assumed to be finite in this paper. A signature \( \mathcal{S}' = \langle I', A', R' \rangle \) is called an extension of \( \mathcal{S} \), if \( I \subseteq I' \) and \( A \subseteq A' \) and \( R \subseteq R' \).

Our work leads to rather complex syntactic descriptions of DL languages, so it is desirable to simplify syntax as much as possible early on. Unfortunately, expressive features that can be considered as syntactic sugar in SROIQ may not be syntactic sugar in the restricted DL fragments we study. For example, the symbol \( \top \) cannot be expressed by \( A \sqcup \neg A \) in DLP. Thus, in general, the precise set of available operators influences the definition and expressivity of DLP. Yet, we do assume within this paper that the universal role \( U \) is not a specific logical symbol, but that it is only available through axiomatisation.

Omitting \( U \) as a language construct significantly simplifies the complexity of the definitions we arrive at. Some further syntactic simplification can be assumed without any reservations: we always write \( \exists R. A \) and \( \forall R. A \) as \( \geq 1 R. A \) and \( \leq 0 R. \neg A \), respectively, and we omit syntactic forms that can be derived by exchanging operators in conjunctions and disjunctions, i.e. we specify grammars only up to commutativity of \( \sqcap \) and \( \sqcup \).

Every SROIQfree GCI \( C \subseteq D \) can be expressed by \( \top \subseteq \neg C \sqcup D \) (i.e. by stating that the concept \( \neg C \sqcup D \) is universally valid). In the following, we will often tacitly assume that GCIs are expressed as universally valid concepts. For further simplification, we consider various syntactic normal forms. We write NNF(KB) for the negation normal form (NNF) of a knowledge base KB, defined as usual. By DNF(KB) we denote the disjunctive normal form, which is obtained by exhaustively replacing subconcepts of the form \( (C \sqcup D) \sqcap E \) with \( (C \sqcap E) \sqcup (D \sqcap E) \). Note that we do not distribute Boolean concept constructors over role restrictions, i.e. our DNF may still contain complex nested concepts. Our canonical definition of DLP is not generally closed under such transformations: stronger normalisation reduces the amount of knowledge bases that the definition can cover.

Obviously, SROIQfree knowledge bases KB can be expressed as semantically equivalent theories of first-order logic with equality (FOL=), where \( I, A, R \) take the roles of constants, unary predicates and binary predicates, respectively. We will use \( \pi(KB) \) to denote one (arbitrary) such translation, and we will consider signatures of SROIQ as FOL= signatures when convenient, but we will assume that only individual names (constants), concept names (unary predicates), and role names (binary predicates) are present in any considered FOL= signature.

Datalog can be defined as syntactic fragment of FOL=. A datalog program over a first-order signature \( \mathcal{S} \) is a first-order theory over \( \mathcal{S} \) which contains only Horn clauses, i.e. FOL= formulae of one of the forms

\[
\forall x(A_1 \land \ldots \land A_n \rightarrow A) \\
\forall x(A_1 \land \ldots \land A_n \rightarrow false) \\
\forall x(true \rightarrow A),
\]

where \( A_{ij} \) are atoms over \( \mathcal{S} \), and \( \forall x \) quantifies over all variables occurring in the implications. We will follow the common practice of omitting the quantifiers as well as true and false.

1 This is easy in SROIQ since \( U \) is not required to be simple: \( \top \subseteq \exists R.[a], \top \subseteq \exists S^-.[a], R \circ S \subseteq U \); we will see later that the same can be done in DLP.
An essential tool for defining DLP is the possibility of exchanging arbitrary signature symbols in an expression to obtain a **renaming**:

**Definition 1.** Let \( F \) be a \( \text{FOL} \) formula, a \( \text{SROIQ}^{\text{free}} \) axiom, or a \( \text{SROIQ}^{\text{free}} \) concept expression, and let \( \mathcal{I} \) be a signature. An expression \( F' \) is a renaming of \( F \) in \( \mathcal{I} \) if \( F' \) can be obtained from \( F \) by replacing each occurrence of a role/concept/individual name with some role/concept/individual name in \( \mathcal{I} \). Multiple occurrences of the same entity name in \( F \) need not be replaced by the same entity name of \( \mathcal{I} \) in this process. A knowledge base \( \mathsf{KB}' \) is a renaming of a knowledge base \( \mathsf{KB} \) if it is obtained from \( \mathsf{KB} \) by replacing each axiom with a renaming.

We will study the semantics and expressivity formulae based on their syntactic structure, disregarding any possible interactions between signature symbols. We therefore call a \( \text{FOL} \) formula, \( \text{SROIQ}^{\text{free}} \) axiom, or \( \text{SROIQ}^{\text{free}} \) concept expression \( F \) **structural** if no signature symbols occurs more than once in \( F \). Clearly, every expression \( F \) can be renamed to a structural expression \( F' \) if a sufficiently (linearly) large signature is available. This is one example of an operation that may require us to use a larger signature than originally given. Since it is not admissible to use infinitely large signatures when studying computational complexities, we need some notation to explicitly extend a DL language by extending its signature. This is the purpose of the following definition.

**Definition 2.** A language \( L \) over a signature \( \mathcal{I} \) is a subset of all \( \text{SROIQ}^{\text{free}} \) concept expressions and axioms over \( \mathcal{I} \). A language scheme is a class \( \mathcal{L} \) of languages such that

1. for every signature \( \mathcal{I} \), there is a language \( L(\mathcal{I}) \) over \( \mathcal{I} \) in \( \mathcal{L} \) and
2. for any two signatures \( \mathcal{I} \) and \( \mathcal{I}' \), and every concept expression or axiom \( C \) from \( L(\mathcal{I}) \), we find that \( L(\mathcal{I}') \) contains every concept expression or axiom \( C' \) obtained from \( C \) by (uniformly and one-to-one) replacing all occurrences of individual, concept and role names from \( \mathcal{I} \) into such from \( \mathcal{I}' \).

We say that \( \mathcal{L} \) contains an axiom of concept expression \( C \) if there is a signature \( \mathcal{I} \) such that \( C \in L(\mathcal{I}) \). Furthermore, the notation is extended to knowledge bases (theories) \( \mathsf{KB} \): we simply write \( \mathsf{KB} \in L \) to express \( \mathsf{KB} \in 2^L \).

Concrete language schemes arising in this paper can typically be described by context-free grammars that are based on (variable) logical signatures – such language schemes can be considered as “parametrised grammars” though the above definition is more general.

Our discussion is necessarily based on a notion of semantic correspondence between different logical theories of DL and datalog. It turns out, however, that semantic equivalence is too strong – it does not allow the use of auxiliary symbols for expressing a logical relationship – while equisatisfiability is too weak – it allows complex semantic translations that are not tractable. The following notion turns out to be a more appropriate middle-ground:

**Definition 3.** Let \( T \) and \( T' \) be two first-order logic theories and let \( \mathcal{I} \) be the signature over which \( T \) is defined. We say that \( T' \) emulates \( T \) if for every \( \text{FOL}_= \) formula \( \varphi \) over \( \mathcal{I} \), \( T' \cup \{ \varphi \} \) is satisfiable if and only if \( T \cup \{ \varphi \} \) is.
Given a SROIQ knowledge base KB and a datalog program P, we say that P emulates KB if P emulates π(KB).

The underlying motivation for this definition is that the “observable” behaviour of $T'$ fully agrees with that of $T$ in all semantic contexts that restrict to the “interface” $\mathcal{I}$. To accomplish this, $T'$ might introduce auxiliary “internal” symbols that are not available in $\mathcal{I}$, and whose interpretation is not observable from the outside. Using these intuitions, emulation appears as a natural tool for enabling information exchange in distributed knowledge systems, since it defines minimal requirements for preserving a theory’s semantics even in combination with additional information.

Emulation can be generalised by parametrising the set of “test formulae” $\varphi$ w.r.t. which we demand equisatisfiability of the two theories. Also note that emulation is loosely related to the notion of conservative extensions [7] in that every conservative extension of a theory $T$ emulates $T$. However, for any conservative extension $T'$ of $T$ holds $T \subseteq T'$ which is not required in our case. On the contrary: the logical (sub)languages in which the emulating and the emulated theory are specified might differ.

The above notion of emulation takes a proof-theoretic approach based on first-order logic formulae. This is motivated by the fact that practical applications typically only consider entailments of (a subset of) first-order logic formulae. However, the following stronger, model-theoretic property is often more convenient to work with in proof.

Definition 4. Given first-order theories $T$ and $T'$ with signatures $\mathcal{I}$ and $\mathcal{I}'$, then $T'$ strongly emulates $T$ if

1. $\mathcal{I}'$ extends $\mathcal{I}$,
2. every model of $T'$ becomes a model of $T$ when restricted to the interpretations of elements from $\mathcal{I}$, and
3. for every model $J$ of $T$ there is a model $I$ of $T'$ that coincides with $J$ on $\mathcal{I}$.

Given a SROIQ knowledge base KB and a datalog program P, we say that P strongly emulates KB if P strongly emulates $\pi(KB)$.

Lemma 1. If $T'$ strongly emulates $T$, then $T'$ emulates $T$.

Proof. An induction on the structure of $\text{FOL}_e$ formulae shows that in a model, the validity of a $\text{FOL}_e$ formula $\varphi$ is independent of the interpretation of the signature elements not occurring in $\varphi$.

Suppose both conditions 2 and 3 hold but $T$ does not emulate $T'$. Hence, there is a $\text{FOL}_e$ formula $\varphi$ over $\mathcal{I}$ such that $T \cup \{\varphi\}$ and $T' \cup \{\varphi\}$ are not equisatisfiable. We consider two cases

- $T \cup \{\varphi\}$ is satisfiable but $T' \cup \{\varphi\}$ is not. Yet, choosing a model of $T \cup \{\varphi\}$ and extending it appropriately by condition ?? yields a model of $T' \cup \{\varphi\}$, which gives a contradiction.
- $T' \cup \{\varphi\}$ is satisfiable but $T \cup \{\varphi\}$ is not. Then, we can select a model of $T' \cup \{\varphi\}$ and restrict it by condition ??, obtaining a model of $T \cup \{\varphi\}$, which again gives a contradiction. □
The semantic conditions used in the previous lemma are in fact stronger than the original requirement of emulation: there are \( \text{FOL} \) theories without this direct correspondence between models, but which still cannot be distinguished by any test formula of \( \text{FOL}_{\text{eq}} \), but only by higher-order theories. However, all emulations considered in this paper feature the stronger semantic correspondence. The earlier definition of emulation is motivated by the intuition that the semantic differences that are captured by first-order logic are those that are observable in practice.

3 Considerations for Defining DLP

In this section, we discuss why defining DLP is not straightforward, and we specify various design principles to guide our subsequent definition. The goal is to arrive at a notion of DLP that is characterised by these principles, as opposed to DLP being some \textit{ad hoc} fragment of description logic that happens to be expressible in datalog without being maximal or canonical in any sense. The first design principle fixes our choice of syntax and underlying DL:

\textit{DLP 1 (DL Syntax)} DLP knowledge bases should be \( \text{SROIQ}^\text{free} \) knowledge bases.

The second principle states that the semantics of every DLP knowledge base can be expressed in datalog. We will see below that it is sometimes useful to introduce auxiliary predicates during the translation to datalog. If this is done, the datalog program can no longer be semantically equivalent to the original knowledge base, even if all consequences with respect to the original predicates are still the same. Yet, \textit{equivsatisfiability} – the requirement that a DLP knowledge base is satisfiable iff its datalog translation is – turns out to be too weak for many purposes. A suitable compromise is the notion of \textit{emulation} as introduced in Definition 3:

\textit{DLP 2 (Semantic Correspondence)} There should be a transformation function \( \text{datalog} \) that maps a DLP knowledge base \( \text{KB} \) to a datalog program \( \text{datalog}(\text{KB}) \) such that \( \text{datalog}(\text{KB}) \) emulates \( \text{KB} \).

It turns out that DLP 2 is a strong requirement with many useful consequences. For example, it ensures us that instance retrieval queries can directly be answered over datalog, without needing to know the details of the datalog transformation: to find out whether \( \text{KB} \) entails \( C(a) \), it suffices to check if \( \text{datalog}(\text{KB}) \) entails \( C(a) \). But DLP 2 is much stronger than the requirement of preserving such atomic consequences, since the entailment of any \( \text{FOL}_{\text{eq}} \) formula over the signature of \( \text{KB} \) can be checked in \( \text{datalog}(\text{KB}) \).

The principles DLP 1 and DLP 2 set the stage for defining DLP but they do not yet provide sufficient details to attempt a definition. The description of DLP as the “intersection” of DL and datalog is not a useful basis for defining DLP: the syntactic intersection of the two formalisms contains no terminological axioms at all. This raises the question of how to define DLP in a canonical way. A naive approach would be to define a DL ontology to belong to DLP if it can be expressed by a semantically
equivalent datalog program. Such a definition would be of little practical use: every inconsistent ontology can trivially be expressed in datalog, and therefore a DL reasoner is needed to decide whether or not a knowledge base should be considered to be in DLP. This is certainly undesirable from a practical viewpoint. It is therefore preferable to give a definition that can be checked without complex semantic computations:

**DLP 3 (Tractability)** Containment of a knowledge base KB in a DLP language over some signature \( \mathcal{S} \) should be decidable in polynomial time with respect to the size of KB and \( \mathcal{S} \).

Note that typical syntactic language definitions are often subpolynomial, e.g. if they can be decided in logarithmic space (which leads to a linear time algorithm that can be parallelised). Yet, polynomial time language definitions might still be acceptable: for example, every decidable DL with transitive roles, number restrictions, and role hierarchies already requires polynomial time for checking simplicity of roles.

The downside of a syntactic approach is that semantically equivalent transformations on a knowledge base may change its status with respect to DLP. This is not a new problem – many DLs are not syntactically closed under semantically equivalent transformations, e.g. due to simplicity restrictions – but it imposes an additional burden on ontology engineers and implementers. To alleviate this problem, a reasonable further design principle is to require closure under at least some forms of equivalence or satisfiability preserving transformations. A particularly common transformation is the construction of negation normal form and disjunctive normal form as defined earlier.

**DLP 4 (Closure Under NNF and DNF)** A knowledge base KB should be in DLP iff its negation normal form NNF(KB) and its disjunctive normal form DNF(KB) are in DLP.

Closure under NNF will turn out to be mostly harmless, while closure under DNF imposes some real restrictions to our subsequent treatment. We still include it here since it allows us to generally present DL concepts as disjunctions, such that the relationship to datalog rules (disjunctions of literals) is more direct.

The above principles still allow DLP to be defined in such a way that some DLP knowledge base subsumes another knowledge base that is not in DLP. In other words, it might happen that adding axioms to a non-DLP knowledge base makes it into a DLP knowledge base. This "nonmonotonic" behaviour is undesirable since it requires implementations and knowledge engineers to consider all axioms of a knowledge base in order to check if it is in DLP. The following principle requires definitions to be more well-behaved:

**DLP 5 (Modularity)** Consider two knowledge bases KB₁ and KB₂. Then KB₁ ∪ KB₂ should be in DLP if and only if both KB₁ and KB₂ are. Moreover, in this case the datalog transformation should be datalog(KB₁ ∪ KB₂) = datalog(KB₁) ∪ datalog(KB₂).

Modularity ensures that one can decide for each axiom of a knowledge base whether or not it belongs to DLP without regarding any other axioms. The goal thus has changed from defining DLP knowledge bases to defining DLP axioms. Note that SROIQ with
global constraints (regularity, simplicity) does not satisfy DLP 5 (to see this, set \( \text{KB}_1 = \{ \text{Tra}(R) \} \) and \( \text{KB}_2 = \{ \top \subseteq \preceq 1 R \top \} \)) which is our actual reason to consider \( \mathcal{SROIQ}^{\text{free}} \) instead of \( \mathcal{SROIQ} \). The above principles already suffice to establish an interesting result about tractability of reasoning in DLP:

**Proposition 1.** Consider a class \( K \) of knowledge bases that belong to a language for which DLP 1 to DLP 5 are satisfied, and such that the maximal size of axioms in \( K \) is bounded. Then deciding satisfiability of knowledge bases in \( K \) is possible in polynomial time.

**Proof.** By DLP 2, satisfiability of \( \text{KB} \in K \) can be decided by checking satisfiability of \( \text{datalog}(\text{KB}) \). Assume that the size of axioms in knowledge bases in \( K \) is at most \( n \). Up to renaming of symbols, there is only a finite number of different axioms of size \( n \). We can assume without loss of generality that the transformation \( \text{datalog} \) produces structurally similar \( \text{datalog} \) for structurally similar axioms, so that there are only a finite number of structurally different \( \text{datalog} \) theories \( \text{datalog}(\{\alpha\}) \) that can be obtained from axioms \( \alpha \) in \( K \). The maximal number of variables occurring within these \( \text{datalog} \) programs is bounded by some \( m \). By DLP 5, the same holds for all programs \( \text{datalog}(\text{KB}) \) with \( \text{KB} \in K \). Satisfiability of \( \text{datalog} \) with at most \( m \) variables per rule can be decided in time polynomial in \( 2^m \) [8]. Since \( m \) is a constant, this yields a polynomial time upper bound for deciding satisfiability of knowledge bases in \( K \). \( \square \)

The previous result states that reasoning in any DLP language is necessarily “almost” tractable. Indeed, many DLs allow complex axioms to be decomposed into a number of simpler normal forms of bounded size, and in any such case tractability is obtained, but it turns out that there are arbitrarily large DLP axioms that cannot be decomposed in DLP. Yet, Proposition 1 clarifies why Horn-\( \mathcal{SHIQ} \) cannot be in DLP: \( \text{ExpTime} \) worst-case complexity of reasoning can be proven for a class \( K \) of Horn-\( \mathcal{SHIQ} \) knowledge bases as in the above proposition (see [9], noting that remaining complex axioms can be decomposed in Horn-\( \mathcal{SHIQ} \)).

Note that none of the above principles actually require DLP to contain any knowledge base at all. An obvious approach thus is to define DLP to be the largest language that adheres to all of the chosen design principles. The question to ask at this point is whether this is actually possible: is there a definition of DLP that adheres to the above principles and that includes as many DL ontologies as possible? The answer is a resounding no:

**Proposition 2.** Consider a language \( L_{\text{DLP}} \) that adheres to the principles DLP 1 to DLP 5. There is a language \( L_{\text{DLP}}' \), that adheres to DLP 1 to DLP 5 while covering more knowledge bases, i.e. \( L_{\text{DLP}} \subset L_{\text{DLP}}' \).

**Proof.** We first need to argue that, even with unlimited resources for the datalog translation, it is not possible that DLP supports all \( \mathcal{SROIQ} \) axioms. We show that, if the concept expression \( C \) is satisfiable and does not contain the symbols \( R, A_1, A_2, c \) and \( d \), then the axiom \( \alpha := \{ c \subseteq C \cap \exists R.(d \cap (A_1 \sqcup A_2)) \} \) cannot be emulated by any datalog program. For a contradiction, suppose that \( \alpha \) is emulated by a datalog theory \( \text{datalog}(\alpha) \). By construction, \( \alpha \) is satisfiable, and so is \( \{ \alpha, A_i \subseteq \bot \} \) for each \( i = 1, 2 \). By Definition 3, we find that \( \text{datalog}(\alpha) \cup \{ A_i \subseteq \bot \} \) is satisfiable, too. Thus, there are models \( I_i \)
of $\text{datalog}(a)$ such that $A_1^I = \emptyset$. By the least model property of $\text{datalog}$, there is also a model $I$ of $\text{datalog}(a)$ such that $A_1^I = A_2^I = \emptyset$. But then $\text{datalog}(a) \cup \{A_1 \sqcup A_2 \subseteq \bot\}$ is satisfiable although \{$\alpha, A_1 \sqcup A_2 \subseteq \bot\}$ is not, contradicting the supposed emulation.

We can now show that there is some unsatisfiable axiom that is not in $L_{DLP}$. To this end, recall that deciding (un)satisfiability of $\text{SHOIQ}$ concept expressions is $\text{NExpTime}$ hard. This follows from $\text{NExpTime}$ hardness of deciding consistency of $\text{SHOIQ}$ knowledge bases [10] together with the fact that knowledge base satisfiability in $\text{SROIQ}$ can be reduced to concept satisfiability [11]. However, we just showed that, if the axiom $\alpha = \{c\} \subseteq C \cap \exists R.(\{d\} \cap (A_1 \sqcup A_2))$ is in $L_{DLP}$ with symbols $R, A_1, A_2, c, d$ not in $C$, then the concept $C$ is unsatisfiable. Thus, if $L_{DLP}$ contains all unsatisfiable $\text{SHOIQ}$ axioms of the form of $\alpha$, then deciding whether $\alpha \in L_{DLP}$ is equivalent to deciding whether $C$ is unsatisfiable (since one can clearly construct $\alpha$ from $C$ in polynomial time). By DLP 3, this would yield a polynomial decision procedure for $\text{SHOIQ}$ concept satisfiability – a contradiction.

Therefore, there is an unsatisfiable axiom $\alpha$ with $\alpha \notin L_{DLP}$. Now let $L'_{DLP}$ be defined as $L_{DLP} \cup \{\text{KB} \mid \text{NNF(KB)} \setminus \{\text{NNF}(\alpha)\} \in L_{DLP}\}$. The transformation is given by $\text{datalog}(\text{NNF}(\text{KB})) = \text{datalog}(\text{KB})$ if $\text{KB} \in L_{DLP}$, and $\text{datalog}(\text{NNF}(\text{KB})) = \{\rightarrow A(x), A(x) \rightarrow \} \cup \text{datalog}(\text{NNF}(\text{KB})) \setminus \{\text{NNF}(\alpha)\}$ otherwise, where $A$ is a new predicate symbol. It is immediate that this defines a DL fragment (DLP 1), and that this definition is tractable (DLP 3). Equisatisfiability (DLP 2) follows since any knowledge base containing an axiom that is equivalent to $\alpha$ is unsatisfiable. Closure under negation normal form (DLP 4) and modularity (DLP 5) are immediate. \hfill $\Box$

This shows that any attempt to arrive at a maximal definition of DLP based on the above design principles must fail. Summing up, the above design principles are still too weak for characterising DLP: any concrete definition requires further choices that, lacking concrete guidelines, are necessarily somewhat arbitrary. Thus, while it is certainly useful to capture some general requirements in explicit principles, the resulting approach of defining DLP would not be a significant improvement over existing ad hoc approaches.

Analysing the proof of Proposition 2 reveals the reason why DLP 1 to DLP 5 are still insufficient. Intuitively, a definition of DLP cannot reach the desired maximum since the computations that were required in this case would no longer be polynomial (DLP 3). Even DLP 5 does not ameliorate the situation, since expressive DLs can encode complex semantic relationships even within single axioms. The core of the argument underlying Proposition 2 in this sense is the fact that there is no polynomial time procedure for deciding whether or not a single $\text{SROIQ}$ axiom can be expressed in $\text{datalog}$.

These considerations highlight a strategy for further constraining DLP to obtain a clearly defined canonical definition instead of infinitely many non-optimal choices. Namely, it is necessary to prevent complicated semantic effects that may arise when considering even single DL axioms from having any impact on the definition of DLP. Intuitively speaking, the reason for the high complexity of evaluating single axioms is that individual parts of an axiom, even if they are structurally separated, may semantically affect each other. In expressive DLs, individual parts of an axiom can capture the semantics of arbitrary terminological axioms: the TBox can be internalised into a single axiom. An important observation now is that the semantic interplay of parts of an axiom
usually requires entity names to be reused. For example, the axiom \( \top \sqsubseteq A \sqcap \neg A \) is unsatisfiable because the concept name \( A \) is used in both conjuncts, while the structurally similar formula \( \top \sqsubseteq A \sqcap \neg B \) is satisfiable. So, in order to disallow complex semantic effects within single axioms to affect DLP, we can require DLP to be closed under the exchange of entities as in Definition 1:

**DLP 6 (Structurality)** Consider knowledge bases \( KB \) and \( KB' \) such that \( KB' \) is an arbitrary renaming \( KB \). Then \( KB \) is in DLP iff \( KB' \) is.

Note that we do not require all occurrences of an entity name to be renamed together, so it is indeed possible to obtain \( A \sqcap \neg B \) from \( A \sqcap \neg A \). This is clearly a very strong requirement since it forces DLP to be based on the syntactic structure of axioms rather than on the semantic effects that occur for one particular axiom that has this structure. Together with modularity (DLP 5), this principle captures the essential difference between a “syntactic” and a “semantic” transformation from DL to datalog. Indeed, if DLP adheres to DLP 5 and DLP 6, then it may only include knowledge bases for which all potential semantic effects can be faithfully represented in datalog. The datalog transformation thus needs to take into account that additional axioms may be added (DLP 5) to state that certain entity names are semantically equivalent, while hardly any semantic consequences can be computed in advance without knowing about these equivalences. In consequence, the semantic computations that determine satisfiability must be accomplished in datalog, and not during the translation. This intuition will turn out to be quite accurate – but a lot more is needed to establish formal results below.

Structurality also interacts with normal form transformations. For example, the concept \((\neg A \sqcup \neg B) \sqcap C\) can be emulated in datalog using rules \( \rightarrow C(x) \) and \( A(x), \land B(x) \rightarrow \). But its DNF \((\neg A \sqcap C) \sqcup (\neg B \sqcap C)\) is only in DLP if its renaming \((\neg A \sqcap C) \sqcup (\neg B \sqcap D)\) is, which turns out to be not the case. Therefore, the knowledge base \( \{\neg A \sqcup \neg B, C \}\) is in DLP but the knowledge base \( \{(\neg A \sqcup \neg B) \sqcap C \}\) is not. We have discussed above why such effects are not avoidable in general. The more transformations are allowed for DLP, the less knowledge bases are contained in DLP. Note that such effects do not occur for negation normal forms.

## 4 The ALC Fragment of DLP

Our investigations in later sections show that the definition of a maximal DLP fragment of \( SROIQ^{\text{free}} \) is surprisingly complex, and the required proofs for showing its maximality are rather intricate. For this reason, we first characterise the maximal DLP fragment of the much simpler description logic \( ALC \). The absence of nominals and cardinality restrictions simplifies the required constructions significantly. Various basic aspects of the relationship between DL and datalog can also be found in this simpler case, but there are also a number of aspects that are not touched at all.

Throughout this section, we use \( \exists \) and \( \forall \) instead of \( \geq 1 \) and \( \leq 0 \ldots \neg \), which yields a more natural syntax for \( ALC \). Exploiting DLP 4 we can simplify the definition of DLP by giving concepts in negation normal form only.
Concepts that are necessarily equivalent to \( \top \) and \( \bot \)

<table>
<thead>
<tr>
<th>( \mathcal{L}^A_{\mathcal{R}} )</th>
<th>( { \top } \mid \forall R. \mathcal{L}^A_{\mathcal{R}} \mid \mathcal{L}^A_{\mathcal{R}} \sqcap \mathcal{L}^A_{\mathcal{R}} \sqcup \mathcal{C} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathcal{L}^A_{\mathcal{L}} )</td>
<td>( { \bot } \mid \exists R. \mathcal{L}^A_{\mathcal{R}} \mid \mathcal{L}^A_{\mathcal{L}} \sqcap C \mid \mathcal{L}^A_{\mathcal{L}} \sqcup \mathcal{L}^A_{\mathcal{L}} )</td>
</tr>
</tbody>
</table>

Body concepts: for \( C \) in normal form, \( C \in \mathcal{L}^A_{\mathcal{B}} \) iff \( C \sqcup \neg C \) (or \( \neg C \subseteq A \)) is in \( \mathcal{DLP}_{\mathcal{ALC}} \).

### Definition 5.**

We define the language scheme \( \mathcal{DLP}_{\mathcal{ALC}} \) as follows. For every signature \( \mathcal{A} \), the language \( \mathcal{DLP}_{\mathcal{ALC}}(\mathcal{A}) \) contains all knowledge bases consisting only of \( \mathcal{SROIQ}^\text{free} \) axioms which are

- GCIs \( C \subseteq D \) over \( \mathcal{A} \) such that \( \text{NNF}(\neg C \sqcup D) \) is a \( \mathcal{L}^A_{\mathcal{B}} \) concept as defined in Fig. 1,

or

- ABox axioms \( C(a) \) where \( \text{NNF}(C) \) is a \( \mathcal{L}^A_{\mathcal{A}} \) concept as defined in Fig. 1.

Following the grammatical structure of \( \mathcal{DLP}_{\mathcal{ALC}} \), we specify three auxiliary functions for constructing datalog programs to emulate a \( \mathcal{DLP}_{\mathcal{ALC}} \) knowledge base.

### Lemma 2.**

Given a concept name \( A \), and a concept \( C \in \mathcal{L}^A_{\mathcal{B}} \), Fig. 2 recursively defines a datalog program \( \text{dlg}^A_{\mathcal{H}}(A \subseteq C) \) that emulates \( A \subseteq C \).

**Proof.** First note that the definition of \( \text{dlg}^A_{\mathcal{H}}(A \subseteq C) \) is well. In particular, programs \( \text{dlg}^A_{\mathcal{H}}(\neg B \subseteq D) \) are only used if \( D \in \mathcal{L}^A_{\mathcal{A}} \). The claim is shown by induction over the definitions of \( \text{dlg}^A_{\mathcal{B}}(\neg A \subseteq C) \) and \( \text{dlg}^A_{\mathcal{H}}(A \subseteq C) \), where the hypothesis for the former is that it emulates \( \neg A \subseteq C \). The easy induction steps can be established by showing strong emulation: any model of the datalog program can be restricted to a model of the corresponding DL axiom, and any model of the DL axiom can be extended to an interpretation that models the datalog program. We omit further details here. Examples of a very similar argument are found in the proofs of Lemma 7 and 8. \( \Box \)

### Lemma 3.**

Given a constant \( a \) and a concept \( C \in \mathcal{L}^A_{\mathcal{A}} \), Fig. 3 recursively defines a datalog program \( \text{dlg}^A_{\mathcal{A}}(C(a), \bot) \) that emulates \( C(a) \).

**Proof.** The construction of Fig. 3 uses a “guard” concept \( E \) that is used to defer the encoding of \( \mathcal{L}^A_{\mathcal{B}} \) disjunctions. The induction claim thus is that, for every \( E \in \mathcal{L}^A_{\mathcal{B}}, C \in \mathcal{L}^A_{\mathcal{A}} \) and \( a \in \mathcal{I} \), the program \( \text{dlg}^A_{\mathcal{A}}(C(a), E) \) strongly emulates \( (C \sqcup E)(a) \).

The concept \( E \) is processed in case \( C \in \mathcal{L}^A_{\mathcal{A}} \) by using \( \text{dlg}^A_{\mathcal{A}} \). Another more interesting case is \( C = \exists R.D \). The basic encoding works by standard Skolemisation, but the guard concept is also processed and a new guard \( \neg Y \) is created for the Skolem constant \( d \). It is not hard to show strong emulation in all cases and we omit further details and refer to the full proofs given in Section 6. \( \Box \)
Definition 6. Given a structural concept \( C \notin L^A_\tau \), a datalog program \( \mathbb{P} \models C \sqsubseteq \bot \) is recursively defined as follows:

\[
\begin{array}{|l|l|}
\hline
C & \text{dlg}_{\mathbb{P}}^A(\neg A \sqsubseteq C) \\
D \in L^A_\tau & \text{dlg}_{\mathbb{P}}^A(\neg A) \\
D \in L^A_\tau & \{A(x)\} \\
\neg B & \{B(x) \rightarrow A(x)\} \\
\forall R.D & \text{dlg}_{\mathbb{P}}^A(\neg X \sqsubseteq D) \cup \{R(x,y) \land X(y) \rightarrow A(x)\} \\
\hline
D_1 \sqcap D_2 \in (L^A_{\mathbb{P}} \sqcap L^A_{\mathbb{P}}) & \text{dlg}_{\mathbb{P}}^A(\neg A \sqsubseteq D_1) \cup \text{dlg}_{\mathbb{P}}^A(\neg A \sqsubseteq D_2) \\
D_1 \sqcup D_2 \in (L^A_{\mathbb{P}} \sqcup L^A_{\mathbb{P}}) & \text{dlg}_{\mathbb{P}}^A(\neg X_1 \sqsubseteq D_1) \cup \text{dlg}_{\mathbb{P}}^A(\neg X_2 \sqsubseteq D_2) \cup \{X_1(x) \land X_2(y) \rightarrow A(x)\} \\
\hline
C & \text{dlg}_{\mathbb{P}}^A(\neg A \sqsubseteq C) \\
D \in D_B & \text{dlg}_{\mathbb{P}}^A(\neg X \sqsubseteq D) \cup \{A(x) \land X(x) \rightarrow \bot\} \\
B & \{A(x) \rightarrow B(x)\} \\
\forall R.D & \text{dlg}_{\mathbb{P}}^A(X \sqsubseteq D) \cup \{A(x) \land R(x,y) \rightarrow X(y)\} \\
D_1 \sqcap D_2 & \text{dlg}_{\mathbb{P}}^A(A \sqsubseteq D_1) \cup \text{dlg}_{\mathbb{P}}^A(A \sqsubseteq D_2) \\
D_1 \sqcup D_2 & \text{dlg}_{\mathbb{P}}^A(X_2 \sqsubseteq D_1) \cup \text{dlg}_{\mathbb{P}}^A(\neg X_1 \sqsubseteq D_2) \cup \{A(x) \land X_1(x) \rightarrow X_2(y)\} \\
\hline
\end{array}
\]

Fig. 2. Transforming axioms \( \neg A \sqsubseteq L^A_\tau \) to datalog

\[
\begin{array}{|l|}
\hline
C & \text{dlg}_{\mathbb{P}}^A(C(a), E) \\
D \in L^A_{\mathbb{P}} & \text{dlg}_{\mathbb{P}}^A(X \sqsubseteq D, E) \cup \{X(a)\} \\
D_1 \sqcap D_2 & \text{dlg}_{\mathbb{P}}^A(D_1(a), E) \cup \text{dlg}_{\mathbb{P}}^A(D_2(a), E) \\
D_1 \sqcup D_2 & \text{dlg}_{\mathbb{P}}^A(\neg X \sqsubseteq D_2) \cup \text{dlg}_{\mathbb{P}}^A(D_1(a), E \sqcup \neg X) \\
\exists R.D & \text{dlg}_{\mathbb{P}}^A(\neg X \sqsubseteq E) \cup \{X(a) \rightarrow R(a,b), X(a) \rightarrow Y(b)\} \cup \text{dlg}_{\mathbb{P}}^A(D(b), \neg Y) \\
E \in L^A_{\mathbb{P}}, X,Y \text{ fresh concept names, } b \text{ a fresh constant} \\
\hline
\end{array}
\]

Fig. 3. Transforming axioms \( C(a) \) with \( C \in L^A_\tau \) to datalog

We summarize these results in the emulation theorem for \( DLP_{\mathcal{ALC}} \).

**Theorem 1.** For every \( DLP_{\mathcal{ALC}} \) axiom \( \alpha \) as in Definition 5, one can construct a datalog program \( \text{dlg}(\alpha) \) that emulates \( \alpha \).

**Proof.** If \( \alpha = C \sqsubseteq D \) is a TBox axiom, define \( \text{dlg}(\alpha) := \text{dlg}_{\mathbb{P}}^A(A \sqsubseteq \text{NNF}(\neg C \sqcup D)) \cup \{A(x)\} \). If \( \alpha = C(a) \) is an ABox axiom, define \( \text{dlg}(\alpha) := \text{dlg}_{\mathbb{P}}^A(C(a), \bot) \). The result follows by Lemma 2 and 3. \( \square \)

It remains to show that \( DLP_{\mathcal{ALC}} \) is indeed the largest DLP fragment of \( \mathcal{ALC} \). We first define auxiliary datalog programs to entail that a concept’s extension is empty for arbitrary concepts that are not in \( L^A_\tau \).

**Definition 6.** Given a structural concept \( C \notin L^A_\tau \), a datalog program \( \mathbb{P} \models C \sqsubseteq \bot \) is recursively defined as follows:
If $C = \perp$ set $[C \subseteq \bot]_{\mathcal{A}} := \emptyset$.
- If $C \in A$ set $[C \subseteq \bot]_{\mathcal{A}} := \{C(x) \rightarrow \bot\}$.
- If $C = \neg B \in \neg A$ set $[C \subseteq \bot]_{\mathcal{A}} := \{B(x)\}$.
- If $C = \forall R.D$ with $D \notin L^\mathcal{A}_C$ set $[C \subseteq \bot]_{\mathcal{A}} := \{R(x, y) \rightarrow \bot\}$.
- If $C = \exists R.D$ set $[C \subseteq \bot]_{\mathcal{A}} := \{R(x, y) \rightarrow \bot\}$.
- If $C = D_1 \cap D_2$ with $D_1 \notin L^\mathcal{A}_C$, $D_1, D_2 \notin L^\mathcal{A}_C$ set $[C \subseteq \bot]_{\mathcal{A}} := \{D_1 \subseteq \bot\} \cup \{D_2 \subseteq \bot\}$.

Given a structural concept $C \notin L^\mathcal{A}_B$, a datalog program $[\top \subseteq C]_{\mathcal{A}} := \{\text{NNF}(\neg C) \subseteq \bot\}_{\mathcal{A}}$.

Note that this definition is well. In particular, observe that $C \notin L^\mathcal{A}_B$ implies $\text{NNF}(\neg C) \notin L^\mathcal{A}_B$. Moreover, it is easy to see that $[C \subseteq \bot]_{\mathcal{A}} (\top \subseteq C)_{\mathcal{A}}$ is satisfiable and entails $C \subseteq \bot (\top \subseteq C)$.

The next lemma shows that concepts that are not in $L^\mathcal{A}_B$ can be forced to require positive information to hold in any model in which they have a non-empty extension.

**Lemma 4.** If $C \notin L^\mathcal{A}_B$ is structural then there is a datalog program $[C \subseteq A]_{\mathcal{A}}$ for a fresh concept name $A$ such that

- $[C \subseteq A]_{\mathcal{A}} \cup \{C(a)\}$ is satisfiable for any individual name $a$, and
- $[C \subseteq A]_{\mathcal{A}} \models C \subseteq A$.

**Proof.** The result is shown by induction over the structure of $C$. If $C \in A$ is a concept name, then $[C \subseteq A]_{\mathcal{A}} := \{C(x) \rightarrow A(x)\}$ clearly satisfies the claim. If $C = \forall R.D$ with $D \notin L^\mathcal{A}_C$ set $[C \subseteq A]_{\mathcal{A}} := \{D \subseteq A\} \cup \{R(x, y) \rightarrow \bot\}$. The claim follows by induction.

If $C = \exists R.D$ with $D \neq \perp$ then $[C \subseteq A]_{\mathcal{A}} := \{R(x, y) \rightarrow A(x)\}$ clearly satisfies the claim.

If $C = D_1 \cap D_2$ with $D_1 \notin L^\mathcal{A}_B$, $D_1, D_2 \notin L^\mathcal{A}_C$ then $[C \subseteq A]_{\mathcal{A}} := \{D_1 \subseteq A\} \cup \{D_2 \subseteq \bot\}$ satisfies the claim by the induction hypothesis. For the case $C = D_1 \cup D_2$ with $D_1 \notin L^\mathcal{A}_B$ and $D_1, D_2 \notin L^\mathcal{A}_C$, we can define $[C \subseteq A]_{\mathcal{A}} := \{D_1 \subseteq A\} \cup \{D_2 \subseteq \bot\}$. The claim follows by induction.

Note that the program $[C \subseteq A]_{\mathcal{A}}$ does not emulate $C \subseteq A$ since the subprogram $[D_2 \subseteq \bot]_{\mathcal{A}}$ that is used for the $\cup$ case excludes a number of interpretations that satisfy $C$. But the previous result suffices for our subsequent arguments.

**Theorem 2.** Consider a structural concept $C$, an individual name $a$, and a concept name $A$ not occurring in $C$.

1. If $C \notin L^\mathcal{A}_B$ then $C(a)$ cannot be emulated by any datalog program.
2. If $C \notin L^\mathcal{A}_B$ then $A \subseteq C$ and $\top \subseteq C$ cannot be emulated by any datalog program.

In particular, no fragment of $\mathcal{ALC}$ that is larger than $\mathcal{DLP}_{\mathcal{ALC}}$ can be emulated in datalog.

**Proof.** The proof for both claims proceeds by an interleaved induction over the structure of $C$. Note that $C$ cannot be atomic in either case. We begin with the induction steps for claim (1), assuming that the claims hold for all subformulae of $C$. Suppose for a contradiction that there is a datalog program $P_{C(a)}$ that emulates $C(a)$.
If $C = \exists R \cdot D$ with $D \notin L_{a}^{A}$ then $P_{C(a)} \cup \{ R(a, y) \rightarrow y \approx b \}$ emulates $D(b)$ for a fresh individual $b$, contradicting the induction hypothesis (1) for $D$. If $C = \forall R \cdot D$ with $D \notin L_{a}^{A}$ then $P_{C(a)} \cup \{ A(x) \rightarrow R(a, x) \}$ emulates $A \subseteq D$, contradicting the induction hypothesis (2) for $D$. If $C = C_1 \cap C_2$ with $C_1 \notin L_{a}^{A}$ and $C_1 \notin L_{a}^{A}$ then $P_{C(a)} \cup \{ \top \subseteq C_2 \} \not\subseteq$ emulates $C_1(a)$, contradicting the induction hypothesis (1) for $C_1$.

Consider the case $C = C_1 \cup C_2$ where $C_1 \notin L_{a}^{A}$. If $C \notin L_{a}^{A}$ then $P_{C(a)} \cup \{ \top \subseteq \}$ emulates $C_1(a)$, again contradicting the induction hypothesis (1) for $C_1$. Otherwise, if $C_1, C_2 \in L_{a}^{A}$ then $C_1, C_2 \notin L_{a}^{A}$. Using fresh concept names $A_1$ and $A_2$, consider datalog programs $P_i := \{ A_i(x) \rightarrow \top \} \cup \{ C_1 \subseteq A_1 \} \not\subseteq \cup \{ C_2 \subseteq A_2 \} \not\subseteq (i = 1, 2)$. It is not hard to see that $\not\subseteq)$ is satisfiable, so the same is true for $P_{C(a)} \cup P_i$ by emulation. Thus, $P_{C(a)} \cup \{ \not\subseteq \} \cup \{ \not\subseteq \} \not\subseteq$ must have a model $I_i$ such that $A_i^{I_i} = \not\subseteq$ for $i = 1, 2$. By the least model property of datalog (see, e.g., [8]), this implies that $P_{C(a)} \cup \{ C_1 \subseteq A_1 \} \not\subseteq \cup \{ C_2 \subseteq A_2 \} \not\subseteq$ has a model $I$ such that $A_f^{I} = A_2^{I} = \not\subseteq$. Thus $P_{C(a)} \cup P_1 \cup P_2$ is satisfiable. But clearly $P_i \models C_i \subseteq (i = 1, 2)$ so $\not\subseteq)$ is unsatisfiable, contradicting the supposed emulation.

This finishes the induction steps for claim (1). For claim (2), suppose for a contradiction that $\not\subseteq \subseteq C \not\subseteq$ is emulated by some datalog program $P_{A \subseteq C}$. First consider the case that $C \notin L_{a}^{A}$. Then $P_{A \subseteq C} \cup \{ A(a) \}$ emulates $C(a)$ for some fresh individual $a$, contradicting the induction hypothesis (1) for $C$. Thus, the remaining induction steps only need to cover the cases of $C \in L_{a}^{A} \setminus L_{a}^{A}$. The case for $C = C_1 \cap C_2$ is similar to step (1). Likewise, the only remaining case of $C = C_1 \cup C_2$ is the case where, w.l.o.g., $C_1 \subseteq L_{a}^{A} \setminus L_{a}^{A}$, which can also be treated as before. There are no remaining cases for $C = \exists R \cdot D$.

Consider the case $C = \exists R \cdot D$ with $D \notin L_{a}^{A}$. Then $P_{A \subseteq C} \cup \{ \top \subseteq D \} \not\subseteq$ emulates $A \subseteq \exists R \cdot \top$. The logic obtained by extending $\mathcal{DL}_{\not\subseteq P_{A \subseteq C}}$ with axioms of the form $A \subseteq \exists R \cdot \top$ is known as Horn-$\mathcal{FL}^\not\subseteq$ [9]. Reasoning in Horn-$\mathcal{FL}^\not\subseteq$ was shown to be $\text{PSPACE}$ hard (op. cit., and this proof can easily be adopted to use only axioms of bounded size. Since $P \neq \text{PSPACE}$ the supposed emulation contradicts Proposition 1.

5 Defining DLP

In this section, we provide a direct definition of DLP. We first summarise the characterisation in Section 3.

Definition 7. A language scheme $L$ is a DLP language scheme if the set of its knowledge bases adheres to the principles DLP 1–DLP 6 of Section 3. A DLP language is a language of the form $L(\mathcal{F})$ for some signature $\mathcal{F}$ and some DLP language scheme $L$. Without loss of generality, it is assumed that the transformation datalog is defined globally over the set of all possible knowledge bases of a DLP language scheme $L$, i.e. $\text{datalog}(KB)$ is independent of the DLP language that KB is taken from.

With the exception of DLP 3, we did not distinguish between language schemes and languages in the above discussion, but both notions are needed to obtain the results in the following sections. On the one hand, a concrete finite signature must be fixed for complexity considerations, and individual DLP languages must be considered in this.
case. On the other hand, an essential technique of the below proofs is to show that certain DLP languages can be extended to include further types of axioms if the language already contains sufficiently expressive knowledge bases to emulate the semantics of the new axioms. In many cases, this method requires some additional auxiliary symbols that cannot be part of the given DLP language’s signature, and hence it must be possible to use a DLP language with a slightly larger signature. This requires the concept of a DLP language scheme from which compatible languages over arbitrary signatures can be derived.

Our goal in this section thus is to define the maximal DLP language scheme. Some practical considerations are needed for this to become practically feasible. Namely, it turns out that the characterisation as given in the previous section leads to a prohibitively complex syntactic description of the language. Our first goal in this section therefore is to identify ways of simplifying its presentation. Note that it is not desirable to simply eliminate “syntactic sugar” in general, since the very goal of this work is to characterise which SROIQ knowledge bases can be considered as syntactic sugar for datalog.

A natural approach is to restrict attention to axioms in some normal form. DLP 4 requires closure under negation normal form, which seems to free us from the burden of explicitly considering negative occurrences of non-atomic concepts. But NNF does not allow for this simplification, since concepts of the form $\leq n \, R \, D$ still contain $D$ in negative polarity. A modified NNF is more adequate.

A SROIQ\textsuperscript{free} concept expression $C$ is in positive negation normal form (pNNF) if
- if $\leq n \, D$ is a subexpression of $C$, then $D$ has the form $\neg D'$, and
- every other occurrence of $\neg$ in $C$ is part of a subconcept $\neg D$ with $D \in A$ or $D = \{a\}$ with $a \in I$.

It is easy to see that any SROIQ\textsuperscript{free} concept expression $C$ can be transformed into a semantically equivalent concept expression pNNF($C$) in linear time. A DLP language thus can be defined by providing its pNNF only.

While pNNF effectively reduces the size of a DLP definition by half, the definition is still exceedingly complex. The construction of disjunctive normal forms is compatible with pNNF, so we can additionally require this form of normalisation. Another source of complexity is the fact that SROIQ features many concept expressions for which all possible renamings are necessarily equivalent to $\top$ or $\bot$. Simple examples such as $\top \cup C$ were already encountered in the definitions of $L_\top$ and $L_\bot$ in Section 4, but SROIQ also includes expressions like $\geq 0 \, R \, C$ or $\leq 3 \, R \, \{a\} \sqcup \{b\}$.

**Definition 8.** Let $C$ be a SROIQ concept expression.
- $C$ is structurally valid if $\top \sqsubseteq C'$ is valid for every renaming $C'$ of $C$.
- $C$ is structurally unsatisfiable if $\bot \sqsubseteq C'$ is valid for every renaming $C'$ of $C$.
- $C$ is structurally refutable if it is not structurally valid, i.e. if there is a renaming $C'$ of $C$ such that $\top \sqsubseteq C'$ is refutable.
- $C$ is structurally satisfiable if it is not structurally unsatisfiable, i.e. if there is a renaming $C'$ of $C$ such that $\bot \sqsubseteq C'$ is refutable.

The renamings $C'$ considered here refer to renamings over arbitrary signatures, and are not restricted to the signature of $C$. 
Lemma 5. The grammars given in Fig. 4 characterise sets of SROIQ concept expressions as follows:

- $C \in L_{\leq n}$ iff $C^I$ contains at most $n$ elements for any interpretation $I$,
- $C \in L_{\leq n}$ iff $C^I$ contains more than $n$ elements for some interpretation $I$,
- $C \in L_{\geq n}$ iff $\Delta C^I \setminus C^I$ contains at most $n$ elements for any interpretation $I$,
- $C \in L_{\geq n}$ iff $\Delta C^I \setminus C^I$ contains more than $n$ elements for some interpretation $I$.

In particular, $L_I$, $L_{\leq n}$, $\overline{L}_I$, and $\overline{L}_{\leq n}$ characterise the sets of structurally valid, unsatisfiable, refutable, or satisfiable concept expressions.

Proof. We first show the “only if” direction of $L_{\leq n}$ by induction over the structure of the grammars. The base cases $\bot$ and $\{I\}$ (where $n \geq 1$ is required) are obvious. The case $L_{\leq n-1}$ (where $n \geq 1$) is immediate from the induction hypothesis. Note that the cases of $\sqcap$ and $\sqcup$ for $n = 0$ are simply special instances of the respective cases for $n \geq 1$. The cases for $L_{\geq n} \cap C$ and $L_{\leq n} \cap L_{\leq m}$ are again obvious from the induction hypothesis.

Considering the grammar for each operator, it can be seen that $\overline{L}_{\leq n}$ is indeed the set complement of $L_{\leq n}$ for each $n$. An easy induction over $n$ is used to show this formally, where it suffices to compare the cases for each constructor to see that they are exhaustive.
show the “only if” direction of the claim for $\mathcal{L}_{\leq n}$, it suffices to show the “only if” direction of the claim for $\mathcal{L}_{\leq n}$.

The “only if” direction of the claim for $\mathcal{L}_{\leq n}$ if again established induction over the structure of concepts in $\mathcal{L}_{\leq n}$. Most cases are obvious. For the case of $C \cap D$, it is necessary to note that the extensions of $C$ and $D$, in addition to containing more than $n$ elements, can always be selected freely to ensure that the intersection of both extensions contains enough elements.

The proofs for the claims about $\mathcal{L}_{\leq n}$ and $\mathcal{L}_{\geq n}$ are similar. □

The previous result shows that structural validity, satisfiability, unsatisfiability, and refutability of a concept expression can be recognised in polynomial time by using the given grammars. For another simplification of our characterisation, we may thus assume that almost all occurrences of such concepts have been eliminated in the concepts that we consider. This completes the ingredients we need for defining the normal form that is used below.

**Definition 9.** A concept expression $C$ is in DLP normal form if $C = \text{DNF}(\text{pNNF}(C))$ and

- if $C$ has a structurally valid subconcept $D$, then $D = \top$ and either $C = D$ or $D$ occurs in a subconcept of the form $\forall n R. D$.
- if $C$ has a structurally unsatisfiable subconcept $D$, then $D = \bot$ and either $C = D$ or $D$ occurs in a subconcept of the form $\exists n R. \neg D$.

The unique DLP normal form of a concept $D$ is denoted by DLPNF($C$).

It is easy to see that DLPNF($C$) can be computed in polynomial time. In particular, structurally valid and unsatisfiable subconcepts can be replaced by $\top$ and $\bot$, respectively, and expressions of the form $C \sqcup \bot$ and $C \sqcap \top$ can be reduced to $C$. Also note that the order of applying the single normalisation steps does not affect the DLP normal form. It therefore suffices to characterise concepts in DLP normal form that are a DLP language scheme. When convenient, we continue to use GCIs $C \subseteq D$ to represent the unique DLP normal form of $\neg C \sqcup D$. Exploiting associativity and commutativity of $\sqcap$ and of $\sqcup$, we furthermore disregard order and nesting of multiple conjunctions or disjunctions.

Whereas structurally valid and invalid subconcepts are ignored in DLP normal forms, we still have reason to consider concepts with restricted extensions. We thus use $\mathcal{D}_{\leq n}(\mathcal{D}_{\geq n}^{-})$ to denote the sublanguage of concepts of $\mathcal{L}_{\leq n}$ (L_{\geq n}) that are in DLP normal form.

Before giving the full definition of a large DLP language scheme, we provide some examples to sketch the complexities of this endeavour (datalog emulations are provided in parentheses). DLP expressions of the form $A \sqcap \exists R. B \sqsubseteq \forall S. C (A(x) \land R(x,y) \land B(y) \land S(x,z) \rightarrow C(z))$ are well-known. The same is true for $A \sqsubseteq \exists R.[c] (A(x) \rightarrow R(x,c))$ but hardly for $A \sqsubseteq \exists R.[c,\{d\}] (A(x) \rightarrow R(x,c), A(x) \rightarrow R(x,d))$. Another unusual form of DLP axioms arises when Skolem constants (not functions) can be used as in the case $\{c\} \sqsubseteq \exists R.A (R(c,s), R(c,s'), A(s), A(s'), s \approx s' \rightarrow \bot$ with fresh $s, s'$) and $A \sqsubseteq \exists R.(\{c\} \sqcup \exists S. \top) (A(x) \rightarrow R(x,c), A(x) \rightarrow S(c,s)$ with fresh $s$). Besides these simple
cases, there are various DLP axioms for which the emulation in datalog is significantly more complicated, typically requiring an exponential number of rules. Examples are \(|c| \supseteq \geq 2 R . (\neg [a] \sqcup A \sqcup B)\) and \(|c| \supseteq \geq 5 R . (A \sqcup [a] \sqcup ([b] \sqcap \leq 1 S . ([c] \sqcup [d])))\). These cases are based on the complex semantic interactions between nominals and atleast-restrictions.

**Definition 10.** We define the language scheme \( \mathcal{DLP} \) as follows. For every signature \( \mathcal{S} \), the language \( \mathcal{DLP}(\mathcal{S}) \) contains all knowledge bases consisting only of \( \mathsf{SROIQ} \) free axioms which are

- \( \mathsf{RBox} \) axioms over \( \mathcal{S} \), or
- \( \mathsf{GCIs} C \subseteq D \) over \( \mathcal{S} \) such that the DLP normal form of \( \neg C \sqcup D \) is a \( \mathcal{DLP} \) concept as defined in the following grammar:

\[
\mathcal{D}_{\mathcal{DLP}} := \top | \bot | C_H | \mathcal{D}^\mathsf{eq} (n \geq 1) | C_{=\top}
\]

where \( C_H \) is defined as in Fig. 5, and \( \mathcal{D}^\mathsf{eq} \) and \( C_{=\top} \) are defined as in Fig. 6, or

- \( \mathsf{ABox} \) axioms \( C(a) \) where the DLP normal form of \( C \) is \( \top, \bot \), or a \( \mathcal{D}_a \) concept as defined in Fig. 5.

In spite of the immense simplifications that DLP normal form provides, the definition of \( \mathcal{DLP} \) still turns out to be extremely complex. We have not succeeded in simplifying the presentation any further without losing substantial expressive features. Some intuitive explanations help to understand the underlying ideas. It is instructive to also compare these intuitions to the above examples.

The core language elements are in Fig. 5. Since all concepts are in DNF, each sublanguage consists of a conjunctive part \( C \) and a disjunctive part \( D \). Definitions of DLP typically distinguish between “head” and “body” concepts, and \( C_H \) and \( C_B \) play a similar role in our definition. \( C_H \) represents concepts that carry the full expressive power of a DLP GCI and that can serve as right hand sides (“heads”) of DLP GCIs. \( C_B \) concepts can be seen as negated generic left hand sides (“bodies”) of GCIs. However, these basic classes are not sufficient for defining a maximal DLP. \( C_a \) characterises concept expressions which can be asserted for named individuals – these are even more expressive than \( C_H \) in that existential restrictions are allowed (intuitively, this is possible as in the context of known individuals the existentially asserted role neighbours can be expressed by Skolem constants). \( D_{\mathsf{mt}} \) concepts then can be viewed as collections of individual assertions (e.g. \([a] \cap B\)). Another way of stating such assertions is to use \( C_z \) in a disjunction (e.g. \( \neg [a] \sqcup B\)).

By far the most complex semantic interactions occur for atleast-restrictions in ABox assertions: \( \mathcal{D}^\mathsf{eq} \) and all subsequent definitions address this single case. For example, the \( \mathcal{DLP} \) axiom \( [a] \supseteq \geq 2 R . (\neg [b] \sqcup A \sqcup B) \) can be emulated by the following set of datalog rules, where \( c_i \) are auxiliary constants:

\[
R(a, c_1), \quad R(a, c_2), \quad b \approx c_1 \rightarrow A(b), \quad b \approx c_2 \rightarrow B(b).
\]

This emulation uses internal symbols to resolve apparently disjunctive cases in a deterministic way. The datalog program does not represent disjunctive information: its least model simply contains two successors that are not equal to \( b \). The nested disjunction only becomes relevant in the context of some disjunctive \( \mathsf{FOL}_{\sqcup} \) formula, such as
Body concepts: for $C$ in normal form, $C \in D_B$ iff $C \subseteq A$ (or $\neg C \subseteq A$) is in DLP

$$C_B := \neg A \mid \neg [I] \mid \neg R.\text{Self} \mid \leq 0 R.\neg(D_B \cup \{ \bot \}) \mid C_B \cap C_B$$

$$D_B := C_B \mid D_B \cup D_B$$

Head concepts: for $C$ in normal form, $C \in D_H$ iff $A \cap C$ is in DLP

$$C_H := C_B \mid A \mid [I] \mid \exists R.\text{Self} \mid \geq 0 R.\neg D_H \mid \leq 1 R.\neg(D_H \cup \{ \bot \}) \mid C_H \cap C_H \mid D_H$$

$$D_H := C_H \mid D_H \cup D_H \mid D_H \cup D_H$$

Assertional concepts: for $C$ in normal form, $C \in D_a$ iff $[a] \subseteq C$ is in DLP

$$C_a := C_H \mid \geq n R.D^{}\omega \mid C_a \cap C_a$$

$$D_a := C_a \mid D_a \cup D_a$$

Disjunctions of nominal assertions of the form $[I] \cap C_a$

$$D_H := [I] \mid [I] \cap C_a$$

$$D_{H+1} := D_a \mid D_H$$

Conjunction of negated nominals, i.e. complements of some nominal disjunction

$$C_{\neg 1} := \neg [I]$$

$$C_{\neg(m+1)} := C_{\neg m} \cap \neg [I]$$

$$C_{\neg m} := \neg [I] \mid C_{\neg m} \cap C_{\neg m}$$

Filler concepts for $\geq n$ in $D_a$

$$D^{\omega} := \top | C_{\omega} \cup D_{\omega}^{\omega} (1 \leq m \leq n^2 - n) \mid D_{\omega} \cup D_{\omega}^{\omega} (m < n) \mid$$

$$D_a \cup D_{\omega}^{\omega} \cup D_H \text{ (for } r \equiv n - (m + 1) \text{ we have } r > 0 \text{ and } r(r - 1) \geq m)$$

where no disjuncts are added for expressions $D_a$ and $D_{\omega}^{\omega}$

Extended concepts with restricted forms of ("local") disjunctions, used in $D^{\omega}$ only

$$C_B^+ := C_B \mid \leq 0 R.\neg D_B^+ \mid \leq n R.\neg(D_B^+ \cup D_{2\omega+m}) \mid C_B^+ \cap C_B^+$$

$$D_B^+ := C_B^+ \mid D_B^+ \cup D_B^+ \mid D_B^+ \cup C_B^+$$

$$C_H^+ := C_H \mid \geq n R.D_H^+ \mid \leq 1 R.\neg D_H^+ \mid \leq n R.\neg(D_H^+ \cap D_{2\omega+m}) \mid C_H^+ \cap C_H^+ \mid D_H^+$$

$$D_H^+ := C_H^+ \mid D_H^+ \cup D_H^+ \mid D_H^+ \cup C_H^+$$

$$C_a^+ := C_H^+ \mid \geq n R.(D_H^+ \cup \{ \top \}) \mid C_a^+ \cap C_a^+$$

$$D_a^+ := C_a^+ \mid D_a^+ \cup D_a^+$$

$$D_{H+1}^+ := D_a^+ \mid D_H^+$$

---

Fig. 5. Grammars for defining DLP concepts in DLP normal form
∀x.x ≅ a ∨ x ≅ b. The considered theory is no longer datalog in this case, and the program simply “re-uses” the disjunctive expressive power provided by the external theory. The fact that the actual program is far from being semantically equivalent to the original axiom illustrates the motive and utility of our definition of emulation.

Many uses of nominals and atleast-restrictions lead to more complex interactions, some of which require completely different encodings. This is witnessed by the more complex arithmetic side condition used in $D^{\text{na}}$. Concepts in $D^{\text{sm}} \cap D^{+}_b$ correspond to disjunctions of $m$ nominal classes, each of which is required to satisfy further disjunctive conditions, as e.g. $[b] \uparrow \geq 1 R.(A \sqcup B)$. Now, as an example, a disjunction of an atomic class and four such “disjunctive nominals” is allowed as a filler for $\geq 7$ (since $3 \times 2 \geq 4$) but not for $\geq 6$ (since $2 \times 1 < 4$). Also note that the disjunctive concepts like $D^{+}_H$ and $D^{+}_m$ are allowed in fillers do not allow all types of disjunctive information but only a finite amount of “local” disjunctions. For example, $[a] \sqcup B \sqcup C$ requires one “local” decision about $a$, whereas concepts like $[a] \sqcap \leq 0 R.\neg(B \sqcup C)$ or $[a] \sqcap < 2 R.\neg \bot$ require arbitrarily many decisions for all $R$ successors.

The remaining grammars in Fig. 6 take care of less interesting special cases. Most importantly, $C^{\text{na}}_H$ covers all concepts that can be emulated if the interpretation domain is restricted to contain just one individual. $C^{\text{na}}_H$ contains axioms which make the knowledge base inconsistent as they deny the existence of a nominal. The auxiliary classes...
C^m describe concepts that cannot be satisfied by an interpretation with at most m elements in their domain, as described in the following lemma.

**Lemma 6.** A structural concept C ≠ ⊥ in DLP normal form is in C^m as defined in Fig. 5 for some m ≥ 1 iff, for all interpretations I with domain size #(A^I) ≤ m, we find I |= C ⊑ ⊥.

**Proof.** The “only if” direction can be shown by an easy induction, where the base cases are given by concepts ∃n R.D with n > m, and – in the case n = 1 – negated nominals $\neg\{a\}$. The proof is straightforward and we omit further details.

For the “if” direction, assume that $C \notin C^m \cup \{\bot\}$, and let $A$ be a domain of size m, i.e. #(A) = m. Then, for any $\delta \in \Delta$, we can find an interpretation $I(\delta, C)$ such that $A^{I(\delta, C)} = A$ and $\delta \in C^{I(\delta, C)}$. The base cases with C of the form C, $\exists R.$Self, $\{I\}$, $\neg C$, $\neg \exists R.$Self, and – if n = 1 – $\neg\{I\}$ are obvious. If $C = D_1 \cup D_2$, then, without loss of generality, $D_1 \notin C^m$ and $I(\delta, C) := I(\delta, D_1)$ satisfies the claim.

Now assume that $C$ is of the form $D_1 \cap D_2$. Then $D_1, D_2 \notin C^m \cup \{\bot\}$, and we find interpretations $I(\delta, D_1)$ and $I(\delta, D_2)$ as in the hypothesis. Since $C$ is structural, we can assume without loss of generality that $I(\delta, D_1) = I(\delta, D_2)$ and set $I(\delta, C) := I(\delta, D_1)$.

If $C = \exists n R.D$, then any interpretation $I(\delta, C)$ with $R^I(\delta, C) = \emptyset$ satisfies the claim.

If $C = \exists n R.D$ with n ≤ m, then consider distinct elements $\delta_1, \ldots, \delta_n \in \Delta$. Using structurality and the induction hypothesis again, we find a model $I(\delta, C) = I(\delta, D_1) = \ldots = I(\delta_n, D)$ such that $R^I(\delta, C) = \{\{\delta, \delta_i\} \mid 1 \leq i \leq n\}$. □

### 6 Emulating DLP in Datalog

In this section, we show that knowledge bases of DLP as given in Definition 10 can indeed be emulated in datalog.

Emulations are generally established by recursively defined transformations functions. Relevant (auxiliary) transformations are required for each of the languages defined in Fig. 5 and 6. In all cases, the built-in semantics of inverse roles is explicitly needed in datalog. For this purpose, an auxiliary datalog program $P_{inv}$ is defined as $P_{inv} := \{R(x, y) \rightarrow Inv(R)(y, x) \mid R \in R\}$, where R is the set of roles of the given DL. We begin with the rather simple case of $D_B$.

**Lemma 7.** Every DLP axiom $\neg A \sqsubseteq C$ with A a concept name and $C \in D_B \cup \{\bot\}$ is strongly emulated by the datalog program $\text{dlg}_B(\neg A \sqsubseteq C)$ as defined in Fig. 7.

**Proof.** Note that the definition in Fig. 7 is well – especially all recursive uses of $\text{dlg}_B$ refer to arguments in the domain of this function. The proof proceeds by induction over the structure of C, showing that the conditions of Definition 4 are satisfied. We show a single induction step to illustrate the easy argumentation.

Consider the case $C = D_1 \cup D_2$. For one direction of the claim, consider any model $I$ of $\neg A \sqsubseteq C$. An interpretation $I'$ over the extended signature is defined by setting $X_i^I := A^I \setminus D_i$ for $i = 1, 2$. It is easy to see that $I' \models \{\neg X_i \sqsubseteq D_i \mid i = 1, 2\} \cup \{X_i(x) \land X_j(x) \rightarrow A(x)\}$. By the induction hypothesis, we can find an interpretation $I_1$ that extends $I'$ and such that $I_1 \models \text{dlg}_B(\neg X_1 \sqsubseteq D_1)$. Another application of the hypothesis
are interdependent, so the correspondence yields a single extended interpretation $I'$ such that $I' \models \text{dlg}_I(A \subseteq \{c\})$ and $I' \models \text{dlg}_I(A \subseteq \{c\})$, as required. The other direction is shown in a similar fashion by applying the induction hypothesis and assumptions of the lemma.\qed
The induction steps for defining $\text{dlg}_H(C) \subseteq C$ are rather more complex, and some preparation is needed first. Concepts of the forms $D_H^+$, $D_H^\ast$, and $D_H^+$ allow for restricted forms of “local” disjunction. To make this notion explicit, we first elaborate how such concepts can be expressed as disjunctions of finitely many $\mathcal{DLP}$ knowledge bases.

**Definition 11.** Consider concept expressions $C$ and $D$ such that:

- $C \subseteq \neg C$ and $D \subseteq D_H^+$, or
- $C \subseteq C$ and $D \subseteq D_H^\ast$, or
- $C \subseteq I$ and $D \subseteq D_H^+$.

A set of knowledge bases $\mathcal{K}_{C \subseteq D}$ is defined recursively as follows:

1. If $D \subseteq D_H$, then $\mathcal{K}_{C \subseteq D} := \{C \subseteq D\}$.
2. Assume $D \subseteq D_H$ for the remaining cases.
3. If $D = D_1 \sqcup D_2$, then:
   1. If $D_1 \subseteq C_\lessgtr$, define auxiliary sets of knowledge bases $\mathcal{K}_M$ for $M \subseteq \text{ind}(D_1)$ as follows: $\mathcal{K}_M := \{C \subseteq \bigcap_{d \in M} \neg d \} \sqcup \bigcup_{d \in \text{ind}(D_1) \setminus M} \mathcal{K}_d$. Then set $\mathcal{K}_{C \subseteq D} := \bigcup_{d \in \text{ind}(D_1)} \mathcal{K}_M$.
   2. If $D_1 \subseteq D_H^+ \setminus C_\lessgtr$, then consider fresh concept names $B_1$ and $B_2$, and define $\mathcal{K}_{C \subseteq D} := \{C \subseteq \neg B_1 \sqcup B_2 \} \sqcup \mathcal{K}_{B_1 \subseteq D_1}, \mathcal{K}_{B_2 \subseteq D_2}$.
   3. If $D_1, D_2 \notin D_H$, then $\mathcal{K}_{C \subseteq D} := \mathcal{K}_{C \subseteq D_1} \cup \mathcal{K}_{C \subseteq D_2}$.
4. If $D = \not\exists n \text{R}.D'$ then:

<table>
<thead>
<tr>
<th>$D \subseteq D_H$</th>
<th>$\text{dlg}_H(A \subseteq C)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D \subseteq D_H$</td>
<td>$\text{dlg}_H(\neg X \subseteq D) \cup {A(x) \land X(x) \rightarrow \bot}$</td>
</tr>
<tr>
<td>$B$</td>
<td>${A(x) \rightarrow B(x)} \cup P_{\text{inv}}$</td>
</tr>
<tr>
<td>$\exists R$.Self</td>
<td>${A(x) \rightarrow R(x, y)} \cup P_{\text{inv}}$</td>
</tr>
</tbody>
</table>

$\exists n \text{R.D} \in \not\exists n \text{R.D}_i$.

$\cup_{c \in \text{ind}(D)} \{A(x) \rightarrow R(x, c)\} \cup \bigcup_{d \in \text{ind}(D_1)} \{A(x) \land c \land d \rightarrow \bot\}$.

$I = \text{ind}(D)$, and $D_i$ such that $D = D_i \sqcup (D_i \sqcap \{c\})$ for some $D_i \subseteq D_H$.

$A, B$ concept names, $c, d$ individual names, $R$ a role name, $X_i$ fresh concept names, $\text{dlg}_H(C \subseteq C)$ as defined in Fig. 9 below.
If \( D' \in D_0^+ \) then w.l.o.g. \( D' = D_1 \cup \ldots \cup D_n \) with \( D_i = \{d_i\} \cap D' \) and \( D' \in C_+^d \). Define \( K_{\text{CBD}} := \{\{c \subseteq \exists R. \bigcup_{i=1}^n \{d_i\}\} \cup \bigcup_{i=1}^n KB_i \mid KB_i \in K_{[d],[D']}\} \).

If \( D' \notin D_0^+ \) then consider a fresh individual name \( d \) and assume that \( K_{[d],[D']} = \{KB_1, \ldots, KB_s\} \). Let \( d_i = \{i\} \), \( i = 1, \ldots, n \) be fresh individuals, and let \( KB_i \) denote the knowledge base \( KB_i \) with all occurrences of \( d \) replaced by \( d_i \). Then define \( K_{\text{CBD}} := \{\{\{d_i\} \cap \{d\} \subseteq \perp \mid 1 \leq i \leq n\} \cup \{\{c \subseteq \exists R.\{d\} \mid 1 \leq i \leq n\} \cup \bigcup_{1 \leq i \leq n} KB_i \mid k_1, \ldots, k_n \in \{1, \ldots, s\}\} \).

If \( D = \leq n R. \neg D' \) then:

(5a) If \( D' \in C_\exists \) then a \( \geq n \)-partitioning \( M \) of \( \text{ind}(D') \) is a set \( M = \{M_1, \ldots, M_m\} \) of \( m \geq n \) mutually disjoint non-empty sets \( M_i \subseteq \text{ind}(D') \). Given such a \( \geq n \)-partitioning, define \( KB_M := \{|c| \subseteq |d| \mid c, d \in M_i \text{ for some } i \in \{1, \ldots, m\} \cup |C \cap \bigcap_{S \subseteq I, \#M_i \cap M \cap S \neq \emptyset \} > n\} \). Then define \( K_{\text{CBD}} := \{KB_M \mid M \text{ a } \geq n \text{-partitioning of } \text{ind}(D')\} \).

(5b) If \( D' = D_1 \cup D_2 \) where \( D_1 \in C_\exists \) and \( D_2 \in D_+^d \), then define a set of knowledge bases \( K_M \) for a set \( M \subseteq \text{ind}(D_1) \) as follows: \( K_M := \{KB \cup \bigcup_{d \in M} KB_d \mid KB \in K_{\text{CBD}} \text{ with } D' = \leq n R. \neg \bigcup_{d \in \text{ind}(D_1), M} \neg \{d\}, KB_d \in K_{[d],[D_2]}\} \). Then define \( K_{\text{CBD}} := \bigcup_{M \subseteq \text{ind}(D_1)} K_M \).

(5c) If \( n \leq 1 \) and \( D' \in D_+^d \) then consider a fresh concept name \( B \), and set \( C' := \neg B \) if \( D' \in D_B^+ \) and \( C' := B \) otherwise. Define \( K_{\text{CBD}} := \{\{c \subseteq \leq n R. \neg C'\} \cup KB \mid KB \in K_{\text{CBD}}\} \).

As usual, empty conjunctions are treated as \( \top \). In cases (3a) and (5b), the construction may lead to axioms in \( L_\top \); these axioms are omitted from \( K_{\text{CBD}} \).

Observe that, without loss of generality, the cases in the previous definition are indeed exhaustive and mutually exclusive for \( D \notin D_0^+ \). In particular, cases (5a) and (5b) cover all situations where \( D \in (D_{2^{n-m}} \cap D_+^d) \), where we find \( \#\text{ind}(D') \) > \( n \) and \( \#\text{ind}(D_1) \) > \( n \), respectively, since we assume that \( D \notin D_d \). It is easy to verify that all recursive uses of \( K_{\text{CBD}} \) satisfy the definition’s conditions on \( C \) and \( D \), and that all axioms in knowledge bases of \( K_{\text{CBD}} \) are in DLP normal form. Note that case (4b) can only occur if \( D \in D_0^+ \setminus D_+^d \), so \( C \) must be a nominal in these cases. Similar observations for the other cases allow us to state the following lemma.

**Lemma 9.** Consider concept expressions \( C \) and \( D \) as in Definition 11. If \( D \in D_0^+ \) then all axioms of the form \( C \subseteq E \) in knowledge bases of \( K_{\text{CBD}} \) are such that \( E \) is in \( D_0^+ \).

In particular, the knowledge bases in \( K_{\text{CBD}} \) are in DLP.

**Proof.** The claim can be verified by considering all axioms that are created in the cases of Definition 11. The claims for \( D_a^+ \), \( D_b^+ \), and \( D_B^+ \) are interdependent and must be proven together.

The claim clearly holds for the base case (1). Case (2) immediately follows from the induction hypothesis. Case (3a) is trivial since additional axioms of the form \( C \subseteq E \) do not occur in knowledge bases of \( K_{[d],[D_2]} \). Case (3b) and (3c) are again immediate from the induction hypothesis, where we note for (3b) that \( D_1 \cup D_2 \) is in \( D_a^+ \) and (3c) for \( D_1 \in D_B^+ \setminus C_\exists \) whenever \( D_2 \) is in \( D_0^+ \).
Case (4a) can only occur if \( D \in \mathbf{D}_H^+ \setminus \mathbf{D}_B^+ \) so it suffices to note that the concept \( \forall n R \sqcap \bigcup_{i=1}^n \lbrack d_i \rbrack \) is in \( \mathbf{D}_H \). Case (4b) in turn requires that \( D \in \mathbf{D}_B^+ \setminus \mathbf{D}_H^+ \), and clearly \( \forall n R \sqcap \bigcup_{i=1}^n \lbrack d_i \rbrack \in \mathbf{D}_e \).

Cases (5a) is immediate, since \( C \sqsubseteq \bigcup_{c \in S} \forall 1 R \sqcap \lbrack c \rbrack \sqsubseteq \bot \) is equivalently expressed as \( C \sqsubseteq \bigcup_{c \in S} \forall 0 R \sqcap \lnot \lbrack c \rbrack \), the conclusion of which is in \( \mathbf{D}_B \). Case (5b) follows directly by induction. Case (5c) comprises three relevant cases: \( n = 0 \) and \( D' \in \mathbf{D}_B^+ (D \in \mathbf{D}_B^+) \), \( n = 1 \) and \( D' \in \mathbf{D}_B^+(D \in \mathbf{D}_B^+) \), We find that \( C' \in \mathbf{D}_B^+(D \in \mathbf{D}_B^+) \) whenever \( D' \in \mathbf{D}_B^+(D \in \mathbf{D}_B^+) \), so that the claim holds in each case.

It remains to show the second part of the claim. Using the first part of the claim, the preconditions on \( C \) and \( D \) imply that all axioms \( C \sqsubseteq E \) that are constructed for \( \mathcal{K}_{C \sqsubseteq D} \) are in \( \mathcal{DLP} \). Axioms \( C' \sqsubseteq E \) in \( \mathcal{K}_{C \sqsubseteq D} \) with \( C' \neq C \) must be obtained from some \( \mathcal{K}_{C' \sqsubseteq D} \) that was used in the construction of \( \mathcal{K}_{C \sqsubseteq D} \). But such recursive constructions only occur in cases where the preconditions of the definition are satisfied, so the claim follows by induction.

The next proposition shows that \( C \sqsubseteq D \) is emulated by the disjunction of the knowledge bases in \( \mathcal{K}_{C \sqsubseteq D} \), thus establishing the correctness of the decomposition. DL does not provide a syntax for knowledge base disjunctions, and we do not want to move to first-order logic here, so we use a slightly different formulation that follows Definition 4.

**Proposition 3.** Consider concept expressions \( C \) and \( D \) as in Definition 11, both based on some signature \( \mathcal{S} \). Let \( \mathcal{S}' \) be the extended signature of \( \mathcal{K}_{C \sqsubseteq D} \).

- Every interpretation \( I \) over \( \mathcal{S} \) with \( I \models C \sqsubseteq D \) can be extended to an interpretation \( I' \) over \( \mathcal{S}' \) such that \( I' \models KB \) for some KB \( \in \mathcal{K}_{C \sqsubseteq D} \).
- For every interpretation \( I' \) over \( \mathcal{S}' \) such that \( I' \models KB \) for some KB \( \in \mathcal{K}_{C \sqsubseteq D} \), we find that \( I' \models C \sqsubseteq D \).

**Proof.** We proceed by induction. Case (1) is obvious. Cases (2) is immediate from the induction hypothesis. For case (3a), let \( M \) be the largest set of individuals such that \( I \models C \sqsubseteq \bigcup_{d \in M} \lnot \lbrack d \rbrack \). Using the induction hypothesis, it is easy to see that \( I \models C \sqsubseteq D \) implies that there is an extension \( I' \) of \( I \) such that \( I' \models KB \) for some KB \( \in \mathcal{K}_{C \sqsubseteq D} \). The converse is similar.

For case (3b), consider an interpretation \( I \) over \( \mathcal{S} \) with \( I \models C \sqsubseteq D \). Consider the extended signature \( \mathcal{S}' \) with the fresh concept names \( B_1 \) and \( B_2 \), and define an extension \( I'' \) of \( I \) over \( \mathcal{S}' \) by setting \( B_1^I \overset{\text{def}}{=} \lnot D_1^I \) and \( B_2^I \overset{\text{def}}{=} D_2^I \). Then \( I'' \models \lnot B_1 \sqsubseteq D_1 \) and \( I'' \models B_2 \sqsubseteq D_2 \), and we can apply the induction hypothesis for \( \mathcal{K}_{C \sqsubseteq D} \) and \( \mathcal{K}_{B \sqsubseteq D} \) to obtain models \( I'' \) (over some extended signature \( \mathcal{S}'' \)) such that \( I'' \models KB_1 \) for some KB \( 1 \in \mathcal{K}_{B \sqsubseteq D} \) and \( I'' \models KB_2 \) for some KB \( 2 \in \mathcal{K}_{B \sqsubseteq D} \). Since \( I'' \) are \( I'' \) agree on \( B_1 \), \( B_2 \), and all symbols of \( C \sqsubseteq D \), there is an interpretation \( I' \) such that \( I' \models KB_1 \sqcup KB_2 \). Since \( C' = \lnot B_1^{I''} \sqcup B_2^{I''} \), it is easy to see that \( I' \) satisfies the conditions of the claim. The other direction of the claim for (3b) is an easy consequence of the induction hypothesis.

Case (3c) can only occur if \( C \in \{I\} \), and it is easy to see that the claim holds in this case.

Case (4a) is again not hard to see when using the induction hypothesis. For case (4b), first note that \( C \) must be a nominal since \( D \) is not be in \( \mathbf{D}_H \). The required emulation
then is an easy consequence of standard Skolemization, where each successor \(d_i\) may satisfy any of the sufficient subconditions that are captured by \(\mathsf{KB}^1,\ldots,\mathsf{KB}^t\).

The reasoning for case (5a) is similar to case (3a): given an interpretation \(\mathcal{I}\), we find a \(\geq n\)-partitioning \(\mathcal{M}\) such that \(c, d \in M_i\) if \(c^f = d^f\). It is easy to see that \(\mathcal{I} \models C \subseteq D\) implies \(\mathcal{I} \models \mathsf{KB}_M\); no induction is required. The other direction is again obvious.

Case (5b) is a simple extension of case (5a) where a subset \(M\) of individuals is selected in each knowledge base to ensure that all individuals of \(M\) are instances of \(D_2\), thus reducing the requirement to a maximal number of \(R\)-successors that do not belong to \(M\). To express this more formally, we use expressions \(\geq 1 U.(d) \cap E\) where \(U\) is the universal role that can be emulated in \(\mathcal{D\mathcal{L}}\mathcal{P}\)– this allows us to embed ABox assertions into GCIs. With this notation, we observe that \(C \subseteq \leq n R.\neg(\mathcal{D}_1 \cup \mathcal{D}_2)\) is strongly emulated by the disjunction of the all the axioms \(C \subseteq \leq n R.\neg\big[(\exists \mathcal{d}_k \mathcal{d} \in \mathcal{M}) \mathcal{M} \mathcal{d} -\mathcal{d}_k \mathcal{d} \mathcal{M} + 1 U.(\mathcal{d} \cap \mathcal{C}_2)\) for all \(M \subseteq \text{ind}(\mathcal{D}_1)\). It is easy to see that the construction in (5b) corresponds to this disjunction, where conjunction is modelled as in case (2), and individual assertions are encoded using the recursive constructions \(\mathcal{K}_{\mathcal{d}_k \mathcal{d} \mathcal{M}}\), that are valid by the induction hypothesis. The converse is easily obtained by similar considerations.

Case (5c) uses a similar argument as case (3b). Consider an interpretation \(\mathcal{I}\) over \(\mathcal{S}\) with \(\mathcal{I} \models C \subseteq D\). For the extended signature \(\mathcal{S}'\) with fresh concept name \(\mathcal{B}\), an extension \(\mathcal{I}'\) of \(\mathcal{I}\) is defined by setting \(C'\mathcal{I}' := D\mathcal{I}'\). By the induction hypothesis for \(\mathcal{K}_{\mathcal{C}\mathcal{D}}\), we find a model \(\mathcal{I}'\) (over some extended signature \(\mathcal{S}'\)) such that \(\mathcal{I}' \models \mathsf{KB}\) for some \(\mathsf{KB} \in \mathcal{K}_{\mathcal{C}\mathcal{D}}\). But then there is a corresponding knowledge base \(\mathsf{KB}' = \{C \subseteq \leq n R.\neg\mathcal{C}\} \cup \mathsf{KB}\) in \(\mathcal{K}_{\mathcal{C}\mathcal{M}}\) such that \(\mathcal{I}' \models \mathsf{KB}'\). Thus \(\mathcal{I}'\) satisfies the conditions of the claim when restricted to \(\mathcal{S}\). The other direction is again easy. \(\square\)

We can now define datalog programs for strongly emulating axioms of the form \([c] \subseteq \geq n R.D^\mathcal{M}\). We consider all three main cases – \(C_{\mathcal{S}_m} \subseteq \mathcal{D}_{\mathcal{B}_1}, \mathcal{D}_{\mathcal{B}_2} \subseteq \mathcal{D}_{\mathcal{B}_1}, \mathcal{D}_{\mathcal{B}_3} \subseteq \mathcal{D}_{\mathcal{B}_1}\) individually, before combining these cases with the remaining forms of \(\mathcal{D}_{\mathcal{B}}\) to complete the induction.

**Lemma 10.** Consider a constant \(c\), and a concept \(C = \geq n R.D_1 \cup D_2\) such that \(D_1 \in \mathcal{C}_{\mathcal{S}_m}, D_2 \in \mathcal{D}_{\mathcal{B}_1}\), and \((1 \leq m \leq n^2 - n)\).

Assume that, for every individual symbol \(d\) and every knowledge base \(\mathsf{KB} \in \mathcal{K}_{\mathcal{d}_k \mathcal{d} \mathcal{M}}\), there is a datalog program \(\mathsf{dl}_q(\mathsf{KB})\) that strongly emulates \(\mathsf{KB}\).

Then we can effectively construct a datalog program \(\mathsf{dl}_q([c] \subseteq C)\) that strongly emulates \([c] \subseteq C\).

**Proof.** Let \(h\) be the smallest number such that \(2^h \geq (\#\mathcal{K}_{\mathcal{d}_k \mathcal{d} \mathcal{M}})^n\), where \(d\) is an arbitrary constant (clearly, the cardinality \(\#\mathcal{K}_{\mathcal{d}_k \mathcal{d} \mathcal{M}}\) does not depend on the choice of \(d\)). Now let \(S := \{c_{ijk} \mid i, j \in \{1,\ldots,n\}, k \in \{1,\ldots,h\}\}\) be a set of \(n \times n \times h\) fresh constants. It is convenient to consider the indices of constants in \(S\) to be coordinates, so that \(S\) consists of the elements of a three dimensional matrix with \(n\) rows, \(n\) columns, and \(h\) layers. Now given any \(k = 1,\ldots,h\), we define sets \(A_i^k, B_i^k \subseteq S\) for all \(i = 1,\ldots,n\) by setting:

\[
A_i^k := \{c_{i1k}, c_{i2k}, \ldots, c_{ink}\} \quad \text{and} \quad B_i^k := \{c_{1ik}, c_{2ik}, \ldots, c_{nik}\}.
\]

In other words, \(A_i^k (B_i^k)\) is the \(i\)th row (column) in layer \(h\) of \(S\). Now given a set \(O \subseteq S\), define \(O(k) := \{c_{ijk} \in O \mid i, j \in \{1,\ldots,n\}\}\) – the intersection of \(O\) with layer \(h\) in \(S\). Now for every \(h\)-tuple \(v = (X_1,\ldots,X_h)\) with \(X_k \in [A, B]\) for all \(k = 1,\ldots,h\), there is
a unique partitioning $P_v = \{O_1, \ldots, O_n\}$ of $S$ into $n$ disjoint subsets $O_i \subseteq S$ ($1 \leq i \leq n$) for which the following holds: for every $i \in \{1, \ldots, n\}$ and $k \in \{1, \ldots, h\}$, we find that $O_j(k) = (X_i)^k$. Observe that the $2^h$ partitions $P_v$ that can be constructed in this way are indeed mutually distinct. Intuitively, the partitions $P_v$ thus encode binary numbers of $h$ digits.

Given partitionings $P = \{O_1, \ldots, O_p\}$ and $P' = \{O_1', \ldots, O_p'\}$ of $S$, we say that $P$ is finer than $P'$ if, for every $i \in \{1, \ldots, p\}$, we find that $O_i'$ is a union of parts $O_j \in P$. Note that every part $O_j$ can be contained in at most one part $O_j'$, and thus $p' \leq p$. Partitions of the form $P_v$ have the following important property: for every partition $P = \{O_1, \ldots, O_p\}$ of $S$ with $p \in \{n, \ldots, n + m - 1\}$, there is at most one partition of the form $P_v$ such that $P$ is finer than $P_v$. To show this, consider two $h$-tuples $v, w \subseteq \{A, B\}^h$ that differ in (at least) the $k$th component ($k \in \{1, \ldots, h\}$), i.e. (w.l.o.g.) the $k$th component of $v$ is $A$, and the $k$th component of $w$ is $B$. Now for any partition $P$ that is finer than $P_v$ and $P_w$, for every $i \in \{1, \ldots, n\}$ there are parts $O_{1i}, \ldots, O_{ni} \in P$ such that $A^k = O_{1i}(k) \cup \ldots \cup O_{ni}(k)$, and parts $O_{1i}', \ldots, O_{ni}' \in P'$ such that $B^k = O_{1i}'(k) \cup \ldots \cup O_{ni}'(k)$. This implies that $P$ cannot contain a part $O$ such that $\# O(k) > 1$ since no two sets $A^k$ and $B^k$ share more than one constant. Hence $P$ must have at least $n^2$ parts to cover all elements in layer $k$. Now the precondition $m \leq n^2 - n$ implies that $n + m - 1 < n^2$, which establishes the claim.

To establish the required datalog program, partitions of constants are considered as equality classes, and rules are created to check for particular equalities. To this end, define a conjunction $[[O]] := c_1 \land \ldots \land c_l$ for every set $O = \{c_1, \ldots, c_l\} \subseteq S$. This notation is extended to partitions $P = \{O_1, \ldots, O_p\}$ of $S$ by setting $[[P]] := [[O_1]] \land \ldots \land [[O_p]]$.

Consider a fresh constant $d$. For every $h$-tuple $v \in \{A, B\}^h$, let $\phi_v : P_v \rightarrow \mathcal{K}_{d_1|d_2}$ be a mapping of parts of $P_v$ to knowledge bases in $\mathcal{K}_{d_1|d_2}$ such that, for every $n$-tuple $K = (KB_1, \ldots, KB_n) \in \mathcal{K}_{d_1|d_2}^n$ of knowledge bases, there is an $h$-tuple $w \in \{A, B\}^h$ with partition $P_v = \{O_1, \ldots, O_n\}$ as defined above, and $\phi_v(O_i) = KB_i$ for all $i = 1, \ldots, n$. This choice of the functions $\phi_v$ is possible due to our initial choice of $h$, since there are $2^h$ such functions but only $\# \mathcal{K}_{d_1|d_2}^n$ different $n$-tuples of knowledge bases from $\mathcal{K}_{d_1|d_2}^n$.

For every partition $P$ of $S$ into $i \in \{1, \ldots, n + m - 1\}$ parts, datalog rules are constructed as follows. If $P$ is not finer than any partition of the form $P_v$, then only the rule $[[P]] \rightarrow \bot$ is added (this includes the case of $P$ having less than $n$ parts). Otherwise, let $P_i$ be the unique partition of this form that is finer than $P$. For every part $O$ of $P_i$, select one part $\pi(O)$ of $P$ such that $\pi(O) \subseteq O$, so that there are $n$ distinct selected parts in $P$. Now let $d_1, \ldots, d_m$ denote the $m$ constants of $D_1$. For every $e = d_1, \ldots, d_m$ and for every part $O \in P_v$, let $A$ be a fresh concept name and construct the following datalog:

(i) $[[P]] \land e \approx f \rightarrow A(e)$, where $f \in \pi(O)$ is arbitrary,

(ii) $\text{dlog}(KB^i)|_{d(e)}$ where $KB^i$ is obtained from $\phi_i(O)$ by replacing all occurrences of $d$ with $e$.

Now $\text{dlog}_i([c] \subseteq C)$ is defined to be the union of $P_{1n}$ and all datalog rules constructed above, and the datalog facts $R(c, c_{ijk})$ for all $i, j \in \{1, \ldots, n\}$ and $k \in \{1, \ldots, h\}$.

It remains to show that $\text{dlog}_i([c] \subseteq C)$ strongly emulates $[c] \subseteq C$. For the one direction, consider a model $I$ of $[c] \subseteq C$. We need to show that it can be extended to a model of $\text{dlog}_i([c] \subseteq C)$. Select $n$ distinct $R$-successors $\delta_1, \ldots, \delta_n$ of $c^I$ such that $\delta_i \in (D_1 \sqcup D_2)^I$ for all $i = 1, \ldots, n$. By Proposition 3, for all $e \in [d_1, \ldots, d_m]$, if $e^I \in D_2^I$ then there is
an extended interpretation $I_e$ such that $I_e \models KB_e$ for some $KB_e \in \mathcal{K}_{\mid\leq D_2}$. Since $I_e$ extends $I$ only over fresh symbols that occur in one $\mathcal{K}_{\mid\leq D_2}$, all interpretations $I_e$ can be combined into a single extension $I'$ of $I$.

Now let $KB'_e \in \mathcal{K}_{\mid\leq D_2}$ denote the knowledge base from which $KB_e$ is obtained by replacing all axioms of the form $\{d\} \subseteq F$ by $\{e\} \subseteq F$, where $d$ is the constant used when constructing $dlg_a(c) \subseteq C$. By the construction of $dlg_a(c) \subseteq C$, there is a tuple $v \in \{A, B\}^h$ and a partition $P_v = \{O_1, \ldots, O_n\}$ such that $\phi_i(O_i) = KB'_d$ for all $i = 1, \ldots, n$ for which $d_i = \delta_1$ and $d_i^{f} \neq \delta_i$ for all $l < j$.

Consider any $e \in \{d_1, \ldots, d_m\}$ with $e^{f} \in D_2^{f}$. The model $I'$ above was constructed such that $I' \models KB_e$, and thus, by the assumption of the lemma, there is an extension $\mathcal{J}'$ of $I'$ such that $\mathcal{J}' \models \text{dlg}(KB_e)$. We define a model $\mathcal{J}$ of $dlg_a(c) \subseteq C$ by further extending $\mathcal{J}'$. For all constants $f \in S$, define $f^{\mathcal{J}} := \delta_i$ for the unique $i \in \{1, \ldots, n\}$ such that $f \in O_i$. Moreover, for each of the fresh concept name $A$ introduced in (i) above, let $A^{\mathcal{J}}$ be the smallest extension for which all rules of (i) are satisfied by $\mathcal{J}$.

Now it is easy to see that $\mathcal{J}$ satisfies the facts $R(c, e_{jk})$ for all $i, j \in \{1, \ldots, n\}$ and $k \in \{1, \ldots, h\}$. To see that it also satisfies the rules constructed in (ii) above, note that the rules (ii) for some particular $e \in \{c_1, \ldots, c_m\}$ are always satisfied if $\mathcal{J} \models A(e)$. Assume $\mathcal{J} \models A(e)$. By minimality of $A^{\mathcal{J}}$, this implies that $\mathcal{J} \models e = f$ for some $f \in S$ that belongs to a part $O_i$ of $P_v$, and thus $e^{\mathcal{J}} = \delta_i$ for some $i \in \{1, \ldots, n\}$. By construction, $\phi_i(O_i)$ is of the form $KB'_d$ (where $e$ might be unequal to $d_i$, but with $e^{f} = d_i^{\mathcal{J}}$). Since $\delta_i \in (D_1 \sqcup D_2)^f$, we find $\delta_i \in D_2^{f}$ and thus $\mathcal{J} \models \text{dlg}_a(b^{f}) \subseteq F$ for all $b \subseteq F \in KB'_d$, where $b' = e$ if $b = d$ and $b' = b$ otherwise. This shows that the rules (ii) are indeed satisfied by $\mathcal{J}$.

For the other direction, consider a model $I$ of $\text{dlg}_a(c) \subseteq C$. We need to show that it is also a model of $\{c\} \subseteq C$. Let $P$ be the partition of $S$ that corresponds to the equivalence classes on $S$ induced by $I$. By the construction of $\text{dlg}_a(c) \subseteq C$, the partition $P$ is finer than some partition of the form $P_v$, and thus has at least $n$ parts. Moreover, $n$ of the parts of $P$ are selected parts of the form $\pi(O)$ for some $O \in P_v$. It is not hard to see that the $n$ domain elements of $I$ that correspond to the selected parts are $R$-successors of $c$ that belong to $(D_1 \sqcup D_2)^f$, which is an easy consequence of rules (i) and (ii) together with the assumed model-theoretic correspondences for axioms in $\mathcal{K}_{\mid\leq D_2}$. \hfill \Box

**Lemma 11.** Consider a constant $c$, and a concept $C = R \cup D_1 \cup D_2$ such that $D_1 \in D_B$, $D_2 \in D_{ml}^n$ with $m < n$ of the form $D_2 = (\{c_1\} \cap C_1) \cup \ldots \cup (\{c_m\} \cap C_m)$.

Assume that for every $i \in \{1, \ldots, m\}$ and every knowledge base $KB \in \mathcal{K}_{\mid\leq C}$, there is a datalog program $\text{dlg}(KB)$ that strongly emulates $KB$.

Then we can effectively construct a datalog program $\text{dlg}_a(c) \subseteq C$ that strongly emulates $\{c\} \subseteq C$.

**Proof.** For each $i = 1, \ldots, m$, let $l_i \geq 1$ be the least number such that $2^{l_i} \geq \#\mathcal{K}_{\mid\leq C}$, and consider a set of fresh constants $S_i := \{a_{1i}, b_{1i}, \ldots, a_{li}, b_{hi}\}$. Let $V_i$ denote the set of all sets of the form $\{x_1, x_2, \ldots, x_l\}$ with $x_h \in \{a_{h}, b_{h}\}$ for all $h \in \{1, \ldots, l_i\}$. Let $\phi_i : V_i \rightarrow \mathcal{K}_{\mid\leq C}$ be an arbitrary surjective function (which exists due to the choice of a sufficiently large $l_i$).
Consider fresh constants $d_1, \ldots, d_{n-m}$ (note that $n-m \geq 1$) and a fresh concept name $B$. We construct the following datalog rules and programs:

(i) $\text{dlg}_B(\neg B \sqsubseteq D_1)$

(ii) for every $i \in \{1, \ldots, n-m\}$:

\[
R(c, d_i),
\]

$B(d_i) \rightarrow \bot$ for a fresh concept name $A$,

(iii) for every $i, j \in \{1, \ldots, n-m\}$, $i \neq j$:

\[
d_i \approx d_j \rightarrow \bot,
\]

(iv) for every $i \in \{1, \ldots, n-m\}$ and $j \in \{1, \ldots, m\}$:

\[
d_i \approx c_j \rightarrow \bot,
\]

(v) for every $i \in \{1, \ldots, m\}$ and $h \in \{1, \ldots, l_i\}$:

\[
R(c, a_{ih}),
\]

\[
R(c, b_{ih}),
\]

\[
a_{ih} \approx b_{ih} \rightarrow \bot,
\]

(vi) for every $i \in \{1, \ldots, m\}$ and $v = \{x_{i1}, x_{i2}, \ldots, x_{id_i}\} \subseteq V_i$:

\[
B(x_{i1}) \wedge \ldots \wedge B(x_{id_i}) \rightarrow A(c_i)\] for a fresh concept name $A$,

\[
A(c_i) \rightarrow c_i \approx x_{i1},
\]

\[
\text{dlg}(\phi(v))|_{A(c_i)}.
\]

(vii) for every $i, j \in \{1, \ldots, m\}$, $i \neq j$, for every $e \in S_i$ and $f \in S_j \cup \{d_1, \ldots, d_{n-m}\}$:

\[
e \approx f \rightarrow e \approx d_1.
\]

Now $\text{dlg}_B([c] \subseteq C)$ is defined as the union of $P_{\text{inv}}$ and all rules and programs constructed above.

It remains to show that $\text{dlg}_B([c] \subseteq C)$ that strongly emulates $[c] \subseteq C$. For the one direction, consider any model $I$ of $[c] \subseteq C$. Select $n$ distinct $R$-successors $\delta_1, \ldots, \delta_n$ of $c$ such that $\delta_i \in (D_1 \cup D_2)^I$ for all $i = 1, \ldots, n$. By Proposition 3, for all $i \in \{1, \ldots, m\}$, if $c_i^I \in C_i^I$ then there is an extended interpretation $I_i$ such that $I_i \models KB_i$ for some $KB_i \in \mathcal{K}([c] \subseteq C)$. Since $I_i$ extends $I$ only over fresh symbols that occur in one $\mathcal{K}([c] \subseteq C)$, all interpretations $I_i$ can be combined into a single extension $I'$ of $I$. By the assumption of the lemma, we find an extension $J'$ of $I'$ such that $J' \models \text{dlg}(KB_i)$.

A model $J$ of $\text{dlg}_B([c] \subseteq C)$ is defined by further extending $J'$. For the auxiliary concept $B$ of (i), define $B^J := (\neg D_1)^J$ and let $J$ be such that $J \models \text{dlg}_B(\neg B \sqsubseteq D_1)$ (which is possible by Lemma 7). For each $i \in \{1, \ldots, n-m\}$, select $d_i^J \in \{\delta_1, \ldots, \delta_n\}$ such that rules (ii)–(iv) above are satisfied. This is always possible since at most $m$ elements of $\{\delta_1, \ldots, \delta_n\}$ can be in $(\neg D_1)^J$. Without loss of generality, we assume that $d_i^J = \delta_i$.

Now select an injective function $\psi : \{1, \ldots, m\} \rightarrow \{2, \ldots, n\}$ such that $\psi(i) = j$ if $c_i^J = \delta_j$ for some $j \in \{1, \ldots, n\}$ and there is no $i' < i$ such that $c_i^J = \delta_j$; and $\psi(i) \in D_1^I$ otherwise. Again, it is not hard to see that this is always possible. Now for each $i \in \{1, \ldots, m\}$, interpretations for constants in $S_i$ are defined as follows. If $c_i^J \in C_i^I$, then let $v \in V_i$ be such that $KB_i = \phi(v)$. Otherwise, let $v \in V_i$ be arbitrary. For all $h \in \{1, \ldots, l_i\}$ and $x \in \{a_{ih}, b_{ih}\}$, define $x^J := \delta_{\psi(h)}$ if $x \in v$, and $x^J := \delta_1$ otherwise. It is not hard to see that $J$ satisfies rules (v) and (vii). For the auxiliary concepts $A$ introduced in (vi) for some set $w \in V_i$, set $A^J := [c_i^J]$ if $w = v$ and $\delta_{\psi(h)} \in (\neg D_1)^J$ (which also implies $c_i^J = \delta_{\psi(h)}$), and set $A^J := \emptyset$ otherwise. Thus, there is at most one such auxiliary concept.
for $i$ that is non-empty, corresponding to the set $v \in V_i$ for which $\text{KB}_i = \phi_i(v)$. The construction of $\mathcal{J}'$ ensures that the remaining rules of (vi) are satisfied as required. It should be observed that this construction also works in the case that $c_i^f = c_j^f$ for some $i \neq j$.

For the other direction, consider any model $I$ of $\text{dlg}_e([c] \subseteq C)$. The rules of (i)–(iv) obviously establish $n - m$ distinct $R$-successors $d_1, \ldots, d_{n-m}$ of $c$ that are in $D_1$. According to rules (vii), for every $i \in \{1, \ldots, m\}$ and every $k \in \{1, \ldots, l_i\}$, some $x_{ik} \in \{a_{ik}, b_{ik}\}$ is unequal to all constants in $S_j \cup \{d_1, \ldots, d_{n-m}\}$ for all $j \neq i$ with $j \in \{1, \ldots, m\}$. Hence, if the premise of the first rule of (vi) is false for all $v \in V_i$, then there must be some $k \in \{1, \ldots, l_i\}$ such that $x_{ik} \neq B'c$ and hence, by (i), $x_{ik}' \in D_1$, yielding the required distinct $R$-successor for $i$. Otherwise, if the premise of the first rule of (vi) is true for some $v \in V_i$, then $c_i^f \approx x_{i1}$ is the required successor, since $c_i^f \in D_2$ is ensured by the rules of (vi) together with the assumptions of the lemma.

**Lemma 12.** Consider a constant $c$, and a concept $C = \mathbb{R} R D_1 \sqcup D_2 \sqcup D_3$ such that $D_1 \subseteq D_0$, $D_2 \subseteq D_0^c$ of the form $D_2 = (\{c_1\} \cap C_1) \cup \ldots \cup (\{c_m\} \cap C_m)$. $D_3 \subseteq D_0^c$ of the form $D_3 = (\{c_{m+1}\} \cap C_{m+1}) \cup \ldots \cup (\{c_{m+l}\} \cap C_{m+l})$, and for $r := n - (m + l)$ we have $r > 0$ and $r(r - 1) \geq m$.

Assume that, for every constant $e$, $\text{dlg}_e([e] \subseteq D_1)$ strongly emulates $[e] \subseteq D_1$, and that, for every $\text{KB} \in \mathcal{K}[c][\subseteq C]$ ($i \in \{1, \ldots, m + l\}$), $\text{dlg}(\text{KB})$ strongly emulates $\text{KB}$.

Then we can effectively construct a datalog program $\text{dlg}_e([c] \subseteq C)$ that strongly emulates $[c] \subseteq C$.

**Proof.** Let $s \geq 1$ be such that $2^s \geq \prod_{i=1}^{m} \# \mathcal{K}[c][\subseteq C]$. Consider the following sets of fresh constants:

- $\{d_i \mid i = 1, \ldots, r\}$,
- $\{e_{ij} \mid i = 1, \ldots, m, j = 1, \ldots, s\}$,
- $\{f_i \mid i = 1, \ldots, l\}$.

Now, for each $i = 1, \ldots, m$, let $\phi_i : \{1, 2\}^s \rightarrow \mathcal{K}[c][\subseteq C]$ be a surjective function from $s$-ary binary numbers to $\mathcal{K}[c][\subseteq C]$, which exists due to our choice of $s$. Moreover, for each $i = 1, \ldots, m$, let $\psi_i = \langle h, k \rangle$ be a pair of distinct numbers $h, k \in \{1, \ldots, r\}, h \neq k$ such that $\psi_i \neq \psi_j$ whenever $i \neq j$. This choice is possible since there are $r(r - 1)$ such pairs and $r(r - 1) \geq m$ was assumed. Given any $j$-ary tuple $\theta$, we use $\theta(k)$ to denote the $k$th component of $\theta$ for $k = 1, \ldots, j$. In particular, the notation $\psi_i(v(j))$ ($i = 1, \ldots, m$, $j = 1, \ldots, s$) with tuples $v \in \{1, 2\}^s$ below.

Let $B$ be a fresh concept name – we will use it to mark certain distinct $R$-successors that the datalog program must ensure to exist. We construct the following datalog rules and programs:

(i) for all $e, f \in \{d_1, \ldots, d_r, c_1, \ldots, c_{m+l}\}$ with $e \neq f$:

\begin{align*}
B(e) \land B(f) \land e \approx f &\rightarrow \bot, \\
B(e) &\rightarrow R(c, e),
\end{align*}

(ii) for all $i \in \{1, \ldots, r\}$:

\begin{align*}
B(d_i), \\
\text{dlg}_e([d_i] \subseteq D_1).
\end{align*}
(iii) for all $i \in \{1, \ldots, m\}$, $v \in \{1, 2\}^*$, $h \in \{1, \ldots, s\}$:

\[ R(c, e_{ih}), \]

\[ \text{dlg}_a(\{e_{ih}\} \subseteq D_1), \]

for all $j \in \{1, \ldots, r\}$, $j \neq \psi_i(1)$, $j \neq \psi_i(2)$: $e_{ih} \approx d_j \rightarrow \bot$,

\[ e_{i1} \approx d_{\psi_i(1)} \land \ldots \land e_{is} \approx d_{\psi_i(2)} \rightarrow A(c_i) \text{ for a fresh concept name } A, \]

\[ A(c_i) \rightarrow B(c_i), \]

\[ \text{dlg}(\phi_i(v))(A(c_i)), \]

(iv) for all $i, j \in \{1, \ldots, m\}$ with $i \neq j$, for all $h \in \{1, \ldots, s\}$:

if there is $k \in \{1, 2\}$ such that $\psi_i(k) = \psi_j(k)$: $e_{ih} \approx e_{jh} \rightarrow e_{ih} \approx d_{\psi_i(k)}$,

otherwise: $e_{ih} \approx e_{jh} \rightarrow \bot$.

(v) for all $i \in \{1, \ldots, l\}$, $j \in \{1, \ldots, r\}$:

\[ R(c, f_i), \]

\[ \text{dlg}_a(\{f_i\} \subseteq D_1), \]

\[ f_i \approx d_j \rightarrow A(c_{mi+1}) \text{ for a fresh concept name } A, \]

\[ A(c_{mi+1}) \rightarrow B(c_{mi+1}), \]

\[ \text{dlg}_a(\{c_{mi+1}\} \subseteq C_{mi+1}), \]

\[ \text{dlg}_a(\{c_{mi+1}\} \subseteq A(c_{mi+1})), \]

(vi) for all $e \in \{f_1, \ldots, f_l, e_{11}, \ldots, e_{1s}, \ldots, e_{ml}, e_{m1}\}$:

for all $f \in \{f_1, \ldots, f_l\}$ with $e \neq f$: $f \approx e \rightarrow f \approx d_1$,

for all $f \in \{c_1, \ldots, c_{mi+1}\}$: $B(f) \land f \approx e \rightarrow \bot$.

Now $\text{dlg}_a([c] \subseteq C)$ is defined as the union of $P_{inv}$ and all rules and programs constructed above.

It remains to show that $\text{dlg}_a([c] \subseteq C)$ that strongly emulates $[c] \subseteq C$. For the one direction, consider any model $I$ of $[c] \subseteq C$. Select $n$ distinct $R$-successors $\delta_1, \ldots, \delta_n$ of $c'$ such that $\delta_i \in (D_1 \cup D_2 \cup D_3)$ for all $i = 1, \ldots, n$. By Proposition 3, for all $i \in \{1, \ldots, m\}$, if $c_i^f \subseteq C_i^f$ then there is an extended interpretation $I'$ such that $I' \models KB_i$ for some $KB_i \in \mathcal{K}_{[c] \subseteq C}$. As in the proof of Lemma 11 above, we can find an extended interpretation $J'$ such that $J' \models KB_i$. Using a similar argument, we can choose $J'$ such that $J' \models \text{dlg}_a([c_j] \subseteq C_j)$ for each $j \in \{m+1, \ldots, m+l\}$ for which $c_j^f \subseteq C_j^f$.

A model $J$ of $\text{dlg}_a([c] \subseteq C)$ is defined by further extending $J'$. At least $r$ elements $\delta \in \{\delta_1, \ldots, \delta_n\}$ must satisfy $\delta \in D_1$ w.l.o.g. assume that this is the case for $\delta_1, \ldots, \delta_r$. Then set $d_j^f := \delta_i$ for all $i \in \{1, \ldots, r\}$.

Now select an injective function $\sigma : \{1, \ldots, m+l\} \rightarrow \{1, \ldots, n\}$ such that $\sigma(i) = j$ if $c_i^f \subseteq C_j^f$, $c_j^f = \delta_j$ for some $j \in \{1, \ldots, n\}$ and there is no $i > j$ such that $c_i^f = \delta_j$; and $\sigma(i) \in D_1$ otherwise. Such a function clearly exists. Consider some $i \in \{1, \ldots, m\}$. If $\delta_{\sigma(i)} \in D_1^f$, then set $c_{\sigma(h)}^f := \sigma(h)$ for each $h \in \{1, \ldots, s\}$. Otherwise, $\delta_{\sigma(h)} = c_{\sigma(h)}$. In this case, let $v \in \{1, 2\}$ be such that $KB_i = \phi_i(v)$, and define $c_{\sigma(h)}^f := d_{\phi_i(v)(\sigma(h))}$ for each $h \in \{1, \ldots, s\}$. Finally, for $i \in \{1, \ldots, l\}$, define $f_i^f := \delta_{\sigma(m+i)}$.

By the assumption of the lemma, for each program of the form $\text{dlg}_a([e] \subseteq D)$ that is constructed in rules (ii), (iii), and (v), we can extend $J$ to symbols of $\text{dlg}_a([e] \subseteq D)$ so that the respective programs are satisfied. For $B$ we select the smallest extensions $B_J$ for which the rules of (ii), (iii), and (v) that use $B$ are satisfied. It is easy to check that the rules of (i) are satisfied. Similarly, we assign minimal extensions to all auxiliary concept names $A$ introduced in (iii) and (v). Now it is not hard to check that $I'$ satisfies all rules of (i)–(vi) as required.
which strongly emulates \( \mathcal{H} \). The remaining induction steps are very similar to the proof of (iii) and (iv) ensure that \( d \) is not equal to any other constant of the form \( d_j \). Rules (iv) ensure that \( e \) is also not equal to any constant of the form \( e_j \), and thus \( e \) constitutes an additional \( R \)-successor of \( c \) that is in \( D_1 \). If no such \( e \) exists, then a rule of (iii) applies for some \( v \in \{1,2\} \), implying that \( c^I \) is the respective fresh concept name \( A \). But then the rules of (iii) together with the assumptions of the lemma imply that \( I \models \phi_3(v) \in \mathcal{K}_{[c]} \subseteq C \). By Proposition 3, we find that \( c^I \in C^I \). Rules (i) and (iv) ensure that \( c \) is distinct from the remaining \( R \)-successors. Overall, we thus obtain \( r + m + l = n \) distinct \( R \)-successors of \( c \) that belong to \( D_1 \) or \( D_2 \) or \( D_3 \).

**Lemma 13.** Consider a concept \( C \subseteq D_a \) and constant \( c \) such that every datalog program \( \text{dlg}_d(c) \subseteq D \) (\( \text{dlg}_B(X \subseteq D) \)) on the right hand side of Fig. 9 strongly emulates \( c \subseteq D \). Then the datalog program \( \text{dlg}_d(c) \subseteq C \) as defined in Fig. 9 strongly emulates \( c \subseteq C \).

**Proof.** The proof proceeds by induction. The complex cases have already been established in Lemma 10, 11, and 12. The remaining induction steps are very similar to the steps in Lemma 7 and 8.

We can now complete our induction by summarizing the previous lemmata.

**Proposition 4.** Consider concepts \( C \subseteq D_a \), \( D \subseteq D_a \), a concept name \( A \), and a constant symbol \( c \). Lemma 7, 8, 10, 11, 12, and 13 together define a recursive construction procedure for datalog programs \( \text{dlg}_d(A \subseteq C) \) and \( \text{dlg}_d(c) \subseteq D \) that strongly emulate \( A \subseteq C \) and \( c \subseteq D \), respectively.

**Proof.** The mentioned results are the basis for establishing an inductive argument to prove the claim. Lemma 10, 11 which 12 require the existence of certain datalog programs \( \text{dlg}(\mathcal{H}) \). For this proof, we define \( \text{dlg}(\mathcal{H}) \) as \( \text{dlg}(\mathcal{H}) \) for this proof, we define \( \text{dlg}(\mathcal{H}) \) := \{\text{dlg}_d(\neg A \subseteq E) \mid \neg A \subseteq E \in \mathcal{H} \} \).
KB) ∪ {dlg_A(A ⊑ E) | A ⊑ E ∈ KB} ∪ {dlg_a([f] ⊑ E) | [f] ⊑ E ∈ KB} (we provide a more general definition of dlg(KB) for other forms knowledge bases at the end of this section). According to Lemma 9 this definition is well and covers all axioms that can occur in KB.

It remains to show that the preconditions of each induction step are indeed satisfied by applying the induction hypothesis that the claim hold for proper subconcepts of the considered concepts. This is obvious whenever preconditions require the claim to hold for programs of the form dlg_A(A ⊑ E) or dlg_a([f] ⊑ E) where C' and D' are proper subconcepts of C and D, respectively.

The induction steps for dlg_a([f] ⊑ D), however, need to use Lemma 10, 11 which 12 additionally require that, for a proper subconcept D' of D and some KB ∈ K[c]∈D', the claim holds for all programs dlg_A(A ⊑ E) with A ⊑ E ∈ KB and for all programs dlg_a([f] ⊑ E) with [f] ⊑ E ∈ KB (the translations dlg_a(¬A ⊑ E) are always given by Lemma 7). Inspecting Definition 11, we find that most axioms in knowledge bases of K[c]∈D' are of the form C ⊑ D' with D' a proper subconcept of D', so that the induction hypothesis applies. However, all cases other than (1), (2), and (3c) also introduce additional axioms that are not referring to subconcepts. By checking the recursive definitions of these axioms, it is easy to see that the claim holds for all axioms of this form.

We still need to show that the “propositional” concepts in D^n can also be emulated in datalog.

**Lemma 14.** For every concept C ∈ D^n for some n ≥ 1, one can construct a datalog program dlg(C) that strongly emulates C.

**Proof.** C is of the form ((c_1) ∩ C_1) ∪ ... ∪ ((c_n) ∩ C_n) with C_1 ∈ C^P_H and C_i ∈ C^P_i for i = 2, ..., n. It is not hard to see that C is semantically equivalent to (c_1) ∩ C_1. This is shown by induction over n. Clearly, all models of C have domains with at most n elements. By Lemma 6, for all n > 1, (((c_1) ∩ C_1) ∪ ... ∪ ((c_n) ∩ C_n) is semantically equivalent to ((c_1) ∩ C_1) ∪ ... ∪ ((c_{n-1}) ∩ C_{n-1}), as required.

All models of (c_1) ∩ C_1 have a unary domain, so that further simplifications are possible. Given any concept D in DLP normal form, let φ(D) be the concept that is obtained by exhaustively applying the following rules:

- If D has a subconcept ≥ 1 R.E, replace this subconcept by E ⊑ R.Self.
- If D has a subconcept ≥ m R.E with m > 1, replace this subconcept by ⊥.
- If D has a subconcept ∼ m R.¬E with m > 1, replace this subconcept by ⊤.

It is easy to check that D ∈ C^P_B(D ∈ C^P_H) implies DLPNF(φ(D)) ∈ D_B (DLPNF(φ(D)) ∈ D_H). Clearly, (c_1) ∩ C_1 is semantically equivalent to (c_1) ∩ φ(C_1), which is in turn equivalent to the knowledge base {⊤ ⊑ {c_1}, {c_1} ⊑ φ(C_1)}. Thus, by Proposition 4, C is strongly emulated by dlg(C) := {x ⊑ c_1} ∪ dlg_a({c_1} ⊑ DLPNF(φ(C_1))) as long as DLPNF(φ(C_1)) ⊆ {⊤, ⊥}. If DLPNF(φ(C_1)) = ⊤ then set dlg(C) := ⊥. If DLPNF(φ(C_1)) = ⊥ then set dlg(C) := {⊤ → ⊥} (the unsatisfiable rule with empty body and head).

To obtain the main result of this section, it remains to show that RBox and ABox axioms in DLP can also be emulated in datalog.
<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>$\text{dlg}(\alpha) \setminus P_{\text{inv}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Ref}(R)$</td>
<td>${R(x, y)}$</td>
</tr>
<tr>
<td>$\text{Tr}(R)$</td>
<td>${R(x, y) \rightarrow \perp}$</td>
</tr>
<tr>
<td>$\text{Sym}(R)$</td>
<td>${R(x, y) \rightarrow R(y, x)}$</td>
</tr>
<tr>
<td>$\text{Asy}(R)$</td>
<td>${R(x, y) \land R(y, x) \rightarrow \perp}$</td>
</tr>
<tr>
<td>$\text{Dis}(R_1, R_2)$</td>
<td>${R_1(x, y) \land R_2(x, y) \rightarrow \perp}$</td>
</tr>
<tr>
<td>$\text{Tra}(R)$</td>
<td>${R(x, y) \land R(y, z) \rightarrow R(x, z)}$</td>
</tr>
<tr>
<td>$R_1 \circ R_2 \circ \ldots \circ R_n \subseteq R$</td>
<td>${R_1(x_0, x_1) \land \ldots \land R_n(x_{n-1}, x_n) \rightarrow R(x_0, x_n)}$</td>
</tr>
</tbody>
</table>

Fig. 10. Transforming $\text{SROIQ}$ RBox axioms to datalog

**Theorem 3.** For every $\mathsf{DLP}$ axiom $\alpha$ as in Definition 10, one can construct a datalog program $\text{dlg}(\alpha)$ that strongly emulates $\alpha$.

**Proof.** If $\alpha$ is a TBox axiom of the form $C \subseteq D$, then set $E := \mathsf{DLPNF}(\neg C \cup D)$. If $E = \top$ then $\text{dlg}(\alpha) := \{\}$. If $E = \bot$ of $E \in C_{\mathcal{T}}$ then $\text{dlg}(\alpha) := \{\top \rightarrow \bot\}$ (the unsatisfiable rule with empty body and head). It is easy to see, that concepts of the form $\alpha$ are indeed unsatisfiable when used as axioms. If $E \in D^m$ for some $n \geq 1$ then set $\text{dlg}(\alpha) := \text{dlg}(E)$ as defined in Lemma 14. Finally, if $E \in D_\alpha$ then set $\text{dlg}(\alpha) := \text{dlg}(A \subseteq E) \cup \{A(\alpha)\}$ as defined in Proposition 4, where $A$ is a fresh concept name.

If $\alpha$ is an ABox axiom of the form $C(\alpha)$ with $\mathsf{DLPNF}(C) \in D_a$ then set $\text{dlg}(\alpha) := \text{dlg}_a(\alpha) \subseteq \mathsf{DLPNF}(C)$ as defined in Proposition 4.

If $\alpha$ is an RBox axiom then $\text{dlg}_R(\alpha)$ is obtained as the union of $P_{\text{inv}}$ and the rules given in Fig. 10. Set $\text{dlg}(\alpha) := \text{dlg}_R(\alpha)$. It is easy to see that this datalog program satisfies the claim. \qed

## 7 Model Constructions

In this section, we introduce constructions on first-order logic interpretations which will be essential for showing that certain formulae cannot be in DLP. The general approach is to find operations that preserve models for datalog programs, i.e. operations under which the set of models of any datalog program must be closed. A well-known model construction in logic programming is the intersection of two Herbrand models, and it is well-known that Horn logic is closed under such intersections. The next definition generalises intersections in two ways: on the one hand, it uses functions to allow for interpretations with different (non-Herbrand) domains; on the other hand, it allows us to construct additional domain elements as feature combinations of existing elements.

**Definition 12.** Consider a first-order logic signature $\mathcal{S}$ and two interpretations $\mathcal{I}_1$ and $\mathcal{I}_2$ over that signature. Consider a set $\mathcal{A}$ and functions $\mu : \mathcal{A} \rightarrow \mathcal{A}^{\mathcal{I}_1}$ and $\nu : \mathcal{A} \rightarrow \mathcal{A}^{\mathcal{I}_2}$ such that, for each constant $c$ in $\mathcal{S}$, there is exactly one element $\delta_c \in \mathcal{A}$ for which $\mu(\delta_c) = c^{\mathcal{I}_1}$ and $\nu(\delta_c) = c^{\mathcal{I}_2}$. The product interpretation $\mathcal{J} = \mathcal{I}_1 \times_{\mu, \nu} \mathcal{I}_2$ is defined as follows:
\[ Δ^2 := Δ, \]

- for each constant c in \( S \), set \( c^2 := δ_c \),
- for each n-ary predicate symbol \( p \) and n-tuple \( \delta \in Δ^n \), set \( \delta ∈ p^I \) if \( μ(\delta) ∈ p^1 \) and \( ν(\delta) ∈ p^2 \), where \( μ(\delta) \) and \( ν(\delta) \) denote the tuples obtained by applying \( μ \) and \( ν \) to each component of \( \delta \).

The previous definition does not imply that constants have distinct interpretations: \( δ_c = δ_d \) if and only if \( c^1 = d^1 \) and \( c^2 = d^2 \). As the definition of equality in product models is similar to the definition of predicate extensions, it is convenient to formulate Definition 12 for first-order logic without equality, assuming that \( \approx \) is introduced by the well-known axiomatization of its properties. A direct definition for \( \text{FOL}_e \) is straightforward.

The essential property of product interpretations is the following:

**Proposition 5.** Consider a signature \( S \), interpretations \( I_1 \) and \( I_2 \), and functions \( μ : Δ → Δ^1 \) and \( ν : Δ → Δ^2 \) as in Definition 12. Then, for every datalog program \( P \) over \( S \), we find that \( I_1 \models P \) and \( I_2 \models P \) implies \( I_1 × μ,ν I_2 \models P \).

**Proof.** Let \( J := I_1 × μ,ν I_2 \). Consider any rule \( B → H \) in \( P \), and a variable assignment \( Z \) for \( J \) such that \( J, Z \models B \). Define a variable assignment \( Z_1 \) for \( I_1 \) by setting \( Z_1(x) := μ(Z(x)) \). By Definition 12, it is easy to see that \( I_1, Z_1 \models B \), and thus \( I_1, Z_1 \models H \). Analogously, we construct a variable assignment \( Z_2 \) such that \( I_2, Z_2 \models B \) and \( I_2, Z_2 \models H \). It is easy to see that this implies \( J, Z \models H \) as required. \( \square \)

A well-known special case of the above product construction is obtained for \( Δ = Δ^1 × Δ^2 \) with \( μ \) and \( ν \) being the projections to the first and second component of each pair in \( Δ \). It turns out that this canonical product construction is not sufficient to detect all cases of knowledge bases that cannot be emulated in datalog. For example, the set of models of the non-DLP axiom \( \{ a \} \succeq 2 R.(\lnot\{ b \} \sqsubseteq \leq 1 S.\lnot A ) \) is closed under canonical products. The more general construction above is needed to address such cases.

When using Proposition 5 to show that a knowledge base cannot be emulated in datalog, it must be taken into account that emulation is not as strong as semantic equivalence. It is not sufficient to show that the models of a knowledge base are not closed under products. For example, the DLP axiom \( \{ a \} \succeq 1 R.\top \) has a model \( I \) with domain \( Δ^1 := \{ a, x \} \), \( Δ^2 := a \), and \( R^I := \{ (a, x) \} \). Yet, the function \( μ : \{ a \} → \{ a, x \} \) with \( μ(a) = a \) can be used to construct an interpretation \( I × μ,ν I \) that is not a model of the axiom. Note that all preconditions of Definition 12 are satisfied. Proposition 5 allows us to conclude that there is no datalog program that is semantically equivalent to \( \{ a \} \succeq 1 R.\top \), but not that there is no such program emulating the axiom. To show that a knowledge base cannot even be emulated in datalog, we therefore use the following observation.

**Lemma 15.** Consider a knowledge base \( KB \) over some signature \( S \). If there are first-order logic theories \( T_1 \) and \( T_2 \) over \( S \) such that:

- \( KB ∪ T_1 \) and \( KB ∪ T_2 \) are satisfiable, and
- for every pair of models \( I_1 \models KB ∪ T_1 \) and \( I_2 \models KB ∪ T_2 \), possibly based on an extended signature \( S' \), there are functions \( μ \) and \( ν \) such that \( I_1 × μ,ν I_2 \not\models KB \),
then KB cannot be emulated in datalog.

If \( T_1 = T_2 \) then this conclusion can also be obtained if the precondition only holds for pairs of equal models \( I_1 = I_2 \).

**Proof.** For a contradiction, suppose that these preconditions of the lemma hold and there is a datalog program \( P \) that emulates KB. Then \( P \cup KB \cup T_i \) is satisfied by some model \( I_i \) of \( P \) for each \( i = 1, 2 \), where the relevant signature of \( P \) may be larger than the signature of KB. Let \( J = I_1 \times_{\mu,\nu} I_2 \) denote the product interpretation from the second condition. Applying Proposition 5, we find that \( J \) is a model of \( P \) that is not a model of KB, contradicting the supposed emulation. The last part of the claim is obvious. \( \square \)

The optional extension of the signature in the previous lemma can be important since the preconditions of Definition 12 require that the domain of the constructed model contains elements for all constant symbols.

As a simple example for this approach, we show that KB = \{\( \top \subseteq A \cup B \}\) cannot be emulated in datalog. Define auxiliary knowledge bases \( KB_1 = \{A \subseteq \bot\} \) and \( KB_2 = \{A \subseteq \bot\} \). Clearly, KB \( \cup KB_1 \) and KB \( \cup KB_2 \) are satisfied by some models \( I_1 \) and \( I_2 \), respectively. However, it is easy to see that no product of \( I_1 \) and \( I_2 \) can be a model of KB – independent of the choice of \( \mu \) and \( \nu \) – since the extensions of \( A \) and \( B \) must always be empty in such a product.

Of course there are other examples for which \( \mu \) and \( \nu \) must be chosen more carefully. In particular, it is sometimes necessary to restrict the amount of new elements that are introduced by the product. The following definition provides a useful notation for such a restricted form of products that will be sufficient for most applications:

**Definition 13.** Consider interpretations \( I_1 \) and \( I_2 \) over a signature \( \mathcal{S} \), and let \( \mathbf{I} \) be the set of constants in \( \mathcal{S} \). Given a set \( S \subseteq \mathbf{I} \times \mathbf{I} \), functions \( \mu : A \rightarrow A^{I_1} \) and \( \nu : A \rightarrow A^{I_2} \) are defined as follows:

- \( A := S \cup \{(c, c) \mid c \in \mathbf{I}\} \),
- \( \mu((c, d)) := c^{I_1} \),
- \( \nu((c, d)) := d^{I_2} \).

\( I_1 \times_S I_2 \) denotes the product interpretation \( I_1 \times_{\mu,\nu} I_2 \) for these functions.

A special aspect of the previous definition is that it restricts attention to named elements – elements that are represented by some individual name – in the original models. It is an easy corollary of Proposition 5 that all other elements are indeed irrelevant for satisfying a datalog program.

### 8 Structural Maximality of DLP

In this section, we show that the earlier definition of \( DLP \) is indeed maximal for the underlying principles. The proof mainly uses the principle of structurality (DLP 6) due to which it suffices to show that structural concept expressions that are not in \( DLP \) cannot be emulated in datalog. To this end, we generally use the strategy suggested by Lemma 15. The below discussions often use datalog rules or DL axioms in the
context of first-order logic to conveniently denote an arbitrary $\text{FOL}_{eq}$ theory of the same semantics, as obtained by any of the standard translations. Especially, this abbreviated form never refers to the more complex datalog transformation of $\text{DLP}$ concepts, and it is only used when syntactic details are not relevant. Moreover, we assume that $\approx$ always denotes the equality predicate, and do not explicitly provide axiomatizations for it.

The outline of the proof is as follows. We start by specifying some useful kinds of auxiliary datalog programs in Definition 14 and 15. The first major class of concept expressions is excluded by Proposition 6 which shows that concepts that are not in $\text{DLP}_2$ can usually not be emulated in datalog. This result is prepared by Lemma 16, Lemma 17, and Lemma 18. These lemmata also are of some utility later on, since they can be used to exclude most forms of existential statements from $\text{DLP}$.

The second main ingredient of the maximality proof is Corollary 1. It extends Proposition 6 by establishing that concepts can typically not be emulated in datalog if they are not in $\text{DLH}$. The chief insight that leads to this result is formulated in Lemma 19 which sports the most complex proof of this section. After this, it is comparatively easy to establish Lemma 20 to treat some pathological cases that had been excluded from the earlier considerations. In particular, it includes the “propositional” case where a DL concept enforces a unary interpretation domain.

The outcomes of Proposition 6, Corollary 1, and Lemma 20 are finally summarized in the main Theorem 4.

To pursue the proof strategy outlined by Lemma 15, our main work consists in specifying suitable auxiliary theories $T_1$ and $T_2$. To simplify this task, we first define some auxiliary theories that will be used frequently. Many of these constructions have the additional advantage of being in datalog – with the important consequence that they are still satisfied by product interpretations (Proposition 5). Often this is relevant for showing that said product interpretations cannot satisfy a given non-$\text{DLP}$ concept.

Whereas many concept expressions $C$ cannot be emulated in datalog, it is usually possible to specify a datalog program that entails $\{c\} \subseteq C$ for a given constant $c$ by specifying sufficient properties that $c$ must satisfy for this to be true. This only fails if $C$ is structurally unsatisfiable. The below construction generalises this idea to any number of constants, and to the dual case where $\{c\} \nsubseteq \neg C$ is entailed. The constructions in Definition 14 and 15 should be compared to the simpler cases discussed in Definition 6 which serve essentially the same purpose for $\text{ALC}$.

**Definition 14.** Consider a structural concept $C$ in positive normal form, and individual names $c_0, \ldots, c_n$ for $n \geq 0$. If $C \not\in \text{L}_{eq}$, the datalog program $[c_0, \ldots, c_n \in C]$ is defined recursively as follows:

- If $C = \top$ or $C = \forall 0 R D$ then $[c_0, \ldots, c_n \in C] := \emptyset$.
- If $C = \{d\}$ then $n = 0$ and $[c_0 \in C] := \{c_0 \approx d\}$.
- If $C$ is of the form $A$, $\neg A$, or $\exists R. \text{Self}$, or $\neg \exists R. \text{Self}$, then $[c_0, \ldots, c_n \in C] := \bigcup_{\text{datalog}(\{c_i\} \subseteq C)}$.
- If $C = D_1 \cap D_2$, then $D_i \not\in \text{L}_{eq}$ for $i = 1, 2$, and $[c_0, \ldots, c_n \in C] := [c_0, \ldots, c_n \in D_1] \cup [c_0, \ldots, c_n \in D_2]$.
- If $C = D_1 \cup D_2$ with $D_i \not\in \text{L}_{eq}$, then $[c_0, \ldots, c_n \in C] := [c_0, \ldots, c_n \in D_1]$.
If $C = D_1 \sqcup D_2$ with $D_1 \in \mathbf{L}_{\omega \eta'}$ and $D_1 \in \mathbf{L}_{\omega \eta''}$ such that $m' + m'' = n - 1$, then
\[
[c_0, \ldots, c_n \in C] := [c_0, \ldots, c_m \in D_1] \sqcup \{c_{m+1}, \ldots, c_{m'} \in D_2\}.
\]
- If $C = \exists m R.D$ with $m \geq 1$, consider fresh constants $d_0, \ldots, d_m$, and set $[c_0, \ldots, c_n \in C] := [d_0, \ldots, d_m \in D] \cup \{R(c_i, d_j) | 0 \leq i \leq n, 0 \leq j \leq m\} \cup \{d_i \approx d_j \rightarrow \bot | 0 \leq i < j \leq m\}.

If $C \not\in \mathbf{L}_{\omega - n}$ then define a datalog program $[c_0, \ldots, c_n \in C] := [c_0, \ldots, c_n \in \text{pNNF}(\neg C)]$.

Note that the given cases directly follow the definition of $\mathbf{L}_{\omega n}$ in Fig. 4. Also note that $[c_0, \ldots, c_n \in C]$ and $[c_0, \ldots, c_n \not\in C]$ are satisfiable, even if we additionally require that all constants $c_i$ are mutually unequal (which is not implied by the datalog programs).

Definition 14 can be viewed as a way to entail statements of the form $[c_0] \sqcup \ldots \sqcup [c_n] \subseteq C$ if $C \not\in \mathbf{L}_{\omega n}$. For cases where $C$ is not in $\mathbf{L}_{\omega n}$ for any $n \geq 0$ this approach can be generalized to entail statements of the form $D \subseteq C$ for a more general class of concepts $D$. The necessary construction is provided by the following definition which is very similar to Definition 14. We provide an alternative perspective and specify the dual case – entailing $C \subseteq D$ in cases where $C \not\in \mathbf{L}_{\omega - n}$ for all $m \geq 0$ – which is the only case that is needed in our subsequent arguments.

**Definition 15.** Consider a structural concept $C$ in positive normal form, and a concept $D \in \mathbf{D}_H$.

If $C \not\in \mathbf{L}_{\omega - m}$ for any $m \geq 0$, the datalog program $[C \subseteq D]_\omega$ is defined recursively as follows:

- If $C = \bot$ then $[C \subseteq D]_\omega := \emptyset$.
- If $C$ is of the form $A. \{1\}$, then $[C \subseteq D]_\omega := \text{datalog}(C \subseteq D)$.
- If $C$ is of the form $\neg A. \text{Self}$, then $[C \subseteq D]_\omega := \text{datalog}(C \subseteq \bot)$.
- If $C = D_1 \sqcup D_2$, then $D_i \not\in \mathbf{L}_{\omega - m}$ for any $m \geq 0$, and $[C \subseteq D]_\omega := [D_1 \subseteq D]_\omega \cup [D_2 \subseteq D]_\omega$.
- If $C = D_1 \sqcap D_2$, then $D_i \not\in \mathbf{L}_{\omega - m}$ for any $m \geq 0$, then $[C \subseteq D]_\omega := [D_1 \subseteq D]_\omega$.
- If $C = \exists m R.\neg D$, consider fresh constants $d_0, \ldots, d_m$, and set $[C \subseteq D]_\omega := \{d_0, \ldots, d_m \not\in D\} \cup \{d_i \approx d_j \rightarrow \bot | 0 \leq i < j \leq m\} \cup \{\text{datalog}(\exists 1 R.[d_i])\}$.
- If $C = \exists m R.\neg D$, then $[C \subseteq D]_\omega := \text{datalog}(\exists 1 R.\top \subseteq D)$.

It should be noted that the cases of the definition are indeed exhaustive. Also observe that $[C \subseteq D]_\omega$ is always satisfiable, where $D \not\subseteq \bot$ is important to ensure that this is actually true for cases like $[\{c\} \subseteq D]_\omega$. This also shows that $[C \subseteq A]_\omega \cup \{A \subseteq \bot\}$ cannot be assumed to be satisfiable in general.

Some further observations should be made in order to understand how Definitions 14 and 15 can be used when discussing datalog emulation. The constructions in both cases do certainly not emulate the statement that they entail. For example, $[c \in C]$ enforces one particular case for which $[c] \subseteq C$; it does in general not describe all such cases. Moreover, the program $[C \subseteq D]_\omega$ may enforce a much stronger condition such as $C \subseteq \bot$. This illustrates that the extension of $C$ can be constrained by $[C \subseteq D]_\omega$. Conversely, a knowledge base $[A \sqcup B \subseteq D]_\omega$ might entail the stronger statement $A \subseteq D$. 
Luckily, as long as structurality is assumed, the knowledge bases of Definition 14 and 15 hardly semantically interact with concept expressions other than those that they are constructed from. Yet, it must be noted that \( \llbracket c_0, \ldots, c_n \in C \rrbracket \) may introduce mutually unequal individuals \( d_i \) for the case \( C = \exists m R.D \), and that two distinct individuals are already required if \( C = \neg [d] \). This effect can occur for all of the above constructions. Logical theories in \( \text{FOL}_{\text{ac}} \) can restrict the maximum size of the domain, and the same is accomplished by DL axioms that correspond to concept expressions in \( \text{L}_{\text{sm}} \) for some \( m \geq 0 \). We need to exclude this possibility when using the above definitions.

The previous discussion shows that it is important to carefully check all states of Definitions 14 and 15 to avoid undesired semantic ramifications. A useful intuition is that the constructed theories enforce a simplification upon \( C \) that allows us to disregard the concept’s internal structure. As an example of a typical usage of these constructions, consider the axiom \( \alpha = \{a\} \subseteq C_1 \cup C_2 \) with \( C_2 \notin \text{L}_\tau \). Then \( \alpha \cup \{a \notin C_2\} \) implies \( \{a\} \subseteq C_1 \). So \( \{a \notin C_2\} \) allowed us to dismiss an “uninteresting” \( C_2 \) to focus on the impact of \( C_1 \).

The following lemmata use the product construction to create elements that are not in a given concept’s extension, where we usually use the abbreviated product construction of Definition 13. In the weakest case, elements outside the extension must be provided to achieve this (Lemma 16). With stronger side conditions, some or even all of the elements can be part of the concept extension (Lemma 17 and 18). The lemmata are essential ingredients for showing that subconcepts that are not in \( \text{D}_{\text{ec}}^+ \) cannot occur in any DLP concept that is in normal form, and the assumptions of the lemma are therefore motivated by the definition of \( \text{D}_{\text{ec}}^+ \).

**Lemma 16.** Consider a structural concept \( C \) in DLP normal form such that \( C \neq \bot \) and \( C \not\in \text{D}_{\text{ec}-}\) for all \( n \geq 0 \) (in particular \( C \neq \top \)). Let \( c_0, \ldots, c_n \) be fresh constants. There is a consistent datalog program \( \llbracket c_0, \ldots, c_n \notin C \rrbracket \) such that

- \( \llbracket c_0, \ldots, c_n \notin C \rrbracket \models \neg (c_i \approx c_j) \) for all \( i, j \in \{0, \ldots, n\} \) with \( i \neq j \),
- \( \llbracket c_0, \ldots, c_n \notin C \rrbracket \models \{c_i\} \subseteq \neg C \) for all \( i = 0, \ldots, n \),
- for all models \( I_1, I_2 \) of \( \llbracket c_0, \ldots, c_n \notin C \rrbracket \), and any set of constants \( N \subseteq I \) with \( \{c_0, \ldots, c_n\} \subseteq N \), the product \( \mathcal{J} = I_1 \times_{(N \times N)} I_2 \) is such that \( \langle c_i, c_j \rangle \notin C^\mathcal{J} \) for all \( i, j \in \{0, \ldots, n\} \).

**Proof.** Using a fresh concept name \( A \), we define \( \llbracket c_0, \ldots, c_n \notin C \rrbracket \models \llbracket C \subseteq \neg A \rrbracket \) for \( \{c_i\} \subseteq \{0 \leq i \leq n\} \cup \{c_i \approx c_j \rightarrow \bot \mid 0 \leq i < j \leq n\} \). Given models \( I_1 \) and \( I_2 \) of \( \llbracket c_0, \ldots, c_n \notin C \rrbracket \), and \( \mathcal{J} = I_1 \times_{(N \times N)} I_2 \), we find that \( \langle c_i, c_j \rangle \notin A^\mathcal{J} \) for all \( i, j \in \{0, \ldots, n\} \). Since \( \llbracket c_0, \ldots, c_n \notin C \rrbracket \) is in datalog, it is satisfied by \( \mathcal{J} \), and thus we conclude \( \langle c_i, c_j \rangle \notin C^\mathcal{J} \) for all \( i, j \in \{0, \ldots, n\} \) as required.

The next lemma considers concepts \( C \not\in \text{D}_{\text{ec}}^+ \). The lemma is also stated for sets of individuals, and additional care is now needed to ensure that it is possible for \( C \) to have a set of (distinct) instances. It is not enough to assume \( C \not\in \text{D}_{\text{ec}}^+ \) for some or all \( n \geq 0 \) since this pre-condition cannot be preserved by all recursive constructions. Namely, the

\[2\] This implication is not quite an emulation since \( \{a \notin C_2\} \) can require a minimal domain cardinality, as discussed above.
Lemma 17. Consider a structural concept C in DLP normal form such that C \in D^+_B, and C does not have a subconcept D \notin D^+_B. Let n \geq 0 be such that n = 0 if C is a disjunction or C \in D^+_B for some k \geq 0, and consider fresh constants c_0, \ldots, c_n, d_0, \ldots, d_m.

Define M := \{c_0, \ldots, c_n, d_0, \ldots, d_m\} \cup \{\langle c \in I \mid c \text{ occurs in } C \rangle\}. There is a consistent datalog program \llbracket c_0, \ldots, c_n \in C, d_0, \ldots, d_m \notin C \rrbracket_k such that

- \llbracket c_0, \ldots, c_n \in C, d_0, \ldots, d_m \notin C \rrbracket_k \models \neg(e \equiv f) for all e, f \in \{c_0, \ldots, c_n, d_0, \ldots, d_m\} with e \neq f.
- \llbracket c_0, \ldots, c_n \in C, d_0, \ldots, d_m \notin C \rrbracket_k \models \{e\} \subseteq C for all i = 0, \ldots, n.
- \llbracket c_0, \ldots, c_n \in C, d_0, \ldots, d_m \notin C \rrbracket_k \models \{d\} \subseteq \neg C for all i = 0, \ldots, m.
- for all models I_1, I_2 of \llbracket c_0, \ldots, c_n \in C, d_0, \ldots, d_m \notin C \rrbracket_k, and any set of constants N \subseteq 1 with M \subseteq N, the product J = I_1 \times_N I_2 is such that \{c_i, d_j\} \notin C^J for all i \in \{0, \ldots, n\} and j \in \{0, \ldots, m\}.

Proof. Note that the conditions imply that C \in D^+_B, and hence C \notin \{\top, \bot\}. Set P := \{e \equiv f \rightarrow \bot \mid e, f \in \{c_0, \ldots, c_n, d_0, \ldots, d_m\}, e \neq f\}. We define \llbracket c_0, \ldots, c_n \in C, d_0, \ldots, d_m \notin C \rrbracket_k recursively based on the structure of C, and we inductively show that it has the required properties. Both parts can conveniently be interleaved. Thus, in each of the following cases, let I_1 and I_2 be models of the \llbracket c_0, \ldots, c_n \in C, d_0, \ldots, d_m \notin C \rrbracket_k just defined, and let J be the product interpretation as in the claim:

- If C has the form A, \{I\} or \exists R. Self, then \llbracket c_0, \ldots, c_n \in C, d_0, \ldots, d_m \notin C \rrbracket_k := P \cup \llbracket c_0, \ldots, c_n \in C \rrbracket \cup \llbracket d_0, \ldots, d_m \notin C \rrbracket_k.

  It is easy to see that J satisfies the claim. Note that the pre-conditions of the lemma imply n = 0 whenever C \in \{I\}.

- If C = D_1 \sqcup D_2 with D_1 \notin D^+_B, then \llbracket c_0, \ldots, c_n \in C, d_0, \ldots, d_m \notin C \rrbracket_k := \llbracket c_0, \ldots, c_n \in D_1, d_0, \ldots, d_m \notin D_1 \rrbracket_k.

  Since I_1 and I_2 are models of \llbracket c_0, \ldots, c_n \in D_1, d_0, \ldots, d_m \notin D_1 \rrbracket_k, the claim follows immediately by induction.

- If C = D_1 \sqcap D_2 with D_1 \notin D^+_B and D_2 \notin D^+_B \forall k for all k \geq 0, then n = 0 is required. Define \llbracket c_0 \in C, d_0, \ldots, d_m \notin C \rrbracket_k := \llbracket c_0 \in D_1, d_0, \ldots, d_m \notin D_1 \rrbracket_k \cup \llbracket D_2 \subseteq \{c_0\}\rrbracket_k.

  I_1 and I_2 are models of \llbracket c_0, \ldots, c_n \in D_1, d_0, \ldots, d_m \notin D_1 \rrbracket_k and we can apply the induction hypothesis. The desired result follows since the product J also satisfies the datalog program \llbracket D_2 \subseteq \{c_0\}\rrbracket_k.

- If C = \exists R. D with k \geq 1, then \llbracket c_0, \ldots, c_n \in C, d_0, \ldots, d_m \notin C \rrbracket_k := P \cup \{R(c_i, e_j) \mid 0 \leq i \leq n, 1 \leq e \leq k\} \cup \{e_1, \ldots, e_n \in D\} \cup \{R(d_i, x) \rightarrow \perp \mid 0 \leq j \leq m\} for fresh individual names e_1, \ldots, e_k.

  It is again easy to see that J satisfies the claim.

- If C = \forall R. \neg D with D \notin D^+_B, then, for a fresh constant e, define \llbracket c_0, \ldots, c_n \in C, d_0, \ldots, d_m \notin C \rrbracket_k := P \cup \{R(c_i, x) \rightarrow x \equiv e, R(c_i, e) \mid 0 \leq i \leq n\} \cup \{R(d_i, f) \mid 0 \leq i \leq m\} \cup \{e \in D, f \notin D \rrbracket_k.

  We find that \langle (c_i, d_j), (e, f) \rangle \in R^J for all i \in \{0, \ldots, n\} and j \in \{0, \ldots, m\}. The claim follows from the induction hypothesis.
- If \( C = \subseteq 1 R \neg D \) with \( D \notin D_{2\omega-k} \) for all \( k \geq 0 \), then consider fresh individuals \( e, f, g \). Define \( \llbracket c_0, \ldots, c_n \in C, d_0, \ldots, d_m \notin C \rrbracket_x := P \cup (R(c_i, x) \rightarrow x = e, R(c_i, e) \mid 0 \leq i \leq n) \cup (R(d_i, f), R(d_i, g) \mid 0 \leq i \leq m) \cup \llbracket e, f, g \notin D \rrbracket_x \).

We find that \( \langle c_i, d_j \rangle, \langle e, f \rangle \rangle \in R^J \) and \( \langle c_i, d_j \rangle, \langle e, g \rangle \rangle \in R^J \) for all \( i \in \{0, \ldots, n\} \) and \( j \in \{0, \ldots, m\} \). The claim follows from Lemma 16.

It should be verified that the given cases are exhaustive. In particular, \( C = \subseteq 1 R \neg D \) with \( D \notin D_{2\omega-k} \) for all \( k \geq 0 \) is the only case where \( C = \subseteq k R \neg D \) for some \( k \geq 1 \) – all other forms are either in \( D^*_H \) or not in \( D^*_H \). Moreover, all recursive applications of the construction satisfy the necessary pre-conditions, especially the requirements for \( n \geq 1 \) are preserved.

The third and final lemma in this series is only needed for two individuals so that we can simplify our presentation slightly. However, the construction now becomes more complex since we can no longer use an auxiliary datalog theory, and since more care is needed in selecting a suitable product interpretation.

**Lemma 18.** Consider a structural concept \( C \) in DLP normal form such that \( C \notin D^*_H \), and \( C \) does not have a subconcept \( D \notin D^*_H \). Let \( c_0, c_1 \) be fresh constants. There is a consistent first-order theory \( \llbracket c_0, c_1 \in C \rrbracket_x \) and a set of constants \( N \subseteq I \) such that

- \( \llbracket c_0, c_1 \in C \rrbracket_x \models \lnot(c_0 \equiv c_1) \),
- \( \llbracket c_0, c_1 \in C \rrbracket_x \models \{c_i \} \subseteq C \) for \( i = 0, 1 \),
- for all models \( I \) of \( \llbracket c_0, c_1 \in C \rrbracket_x \), the product \( J = I \times_{(N \subseteq I)} I \) is such that \( \langle c_0, c_1 \rangle \notin C^J \).

**Proof.** The conditions again imply that \( C \in D^*_H \), and hence \( C \notin \{T, \bot\} \). Moreover, \( C \notin D^*_H \) and \( C \in D^*_H \) implies that \( C \notin D_{\leq 1} \). Indeed, \( C \notin D_{2\omega} \) since \( C \) is in DLP normal form, and thus \( C \in D_{\leq 1} \) would imply that \( C \) is of the form \( \llbracket I \rrbracket \cap C^* \subseteq D^*_H \). This property is inherited by subconcepts \( D \) of \( C \) as long as \( D \notin D^*_H \).

We define \( \llbracket c_0, c_1 \in C \rrbracket_x \) recursively based on the structure of \( C \), and we inductively show that it has the required properties. Both parts can conveniently be interleaved. In addition, we also specify a suitable set \( N \) of constant symbols to use in the product construction in the recursion. Thus, in each of the following cases, let \( I \) be a model of the \( \llbracket c_0, \ldots, c_n \in C \rrbracket_x \) just defined, and let \( J \) be the product interpretation as in the claim.

- If \( C = D_1 \cup D_2 \) with \( D_1, D_2 \notin D^*_H \) then \( \llbracket c_0, c_1 \in C \rrbracket_x := \llbracket c_0 \in D_1, c_1 \notin D_1 \rrbracket_x \cup \llbracket c_1 \in D_2, c_0 \notin D_2 \rrbracket_x \) and the set \( N \) is defined as in Lemma 17.

Using Lemma 17, it is easy to see that \( J \) satisfies the claim.

- If \( C = D_1 \cup D_2 \) with \( D_1 \notin D^*_H \) and \( D_2 \in D^*_H \) then consider a fresh concept name \( A \). Since \( C \notin D_{2\omega-n} \) for all \( n \geq 0 \), the same holds for \( D_1 \) and \( D_2 \). Moreover, \( D_1 \notin D_{\leq 1} \) as discussed initially. We thus can define \( \llbracket c_0, c_1 \in C \rrbracket_x := \llbracket c_0, c_1 \in D_1 \rrbracket_x \cup \llbracket D_2 \subseteq \lnot A \rangle \subseteq \{A(c_0), A(c_1)\} \). The set \( N \) is defined to be the same as for \( \llbracket c_0, c_1 \in D_1 \rrbracket_x \).

\( I \) is a model of \( \llbracket c_0, c_1 \in D_1 \rrbracket_x \) and we can apply the induction hypothesis. The desired result follows since the product \( J \) also satisfies the datalog program \( \llbracket D_2 \subseteq \lnot A \rangle \subseteq \{A(c_0), A(c_1)\} \) (Proposition 5).
– If \( C = D_1 \cap D_2 \) then we can assume \( D_1 \notin D^+_H \). Clearly, \( C \notin D_{21} \) implies \( D_1, D_2 \notin D_{21} \). Thus we can set \( [c_0, c_1] \in C \) := \( [c_0, c_1] \in D_1 \times \) \( [c_0, c_1] \in D_2 \), where \( N \) is again taken to be the set of constants as defined for \( [c_0, c_1] \in D_1 \).

We can again apply the induction hypothesis since \( I \models [c_0, c_1] \in D_1 \), and use the fact that \( J \models [c_0, c_1] \in D_2 \).

– If \( C = \forall n R.D \) then \( D \notin D^+_H \cup D_{mn-1} \cup \{\perp\} \). Since all subconcepts of \( C \) are assumed to be in \( D^+_H \), we conclude that \( D \notin D_{mn} \). Thus we can introduce fresh individual symbols \( d_0, \ldots, d_n \) and set \( [c_0, c_1] \in C \) := \( [d_0], \ldots, d_n \in D \) \( \equiv \{e \approx f \mid e, f \in \{c_0, c_1, d_0, \ldots, d_n\}, e \neq f\} \cup \{X.R(c_0, x) \leftrightarrow \vee_{0 \leq i < n} x \approx d_i\} \cup \{X.R(c_1, x) \leftrightarrow \vee_{0 \leq i < n} x \approx d_i\} \cup \{d_0 \in D\} \cup \{d_n \in D\} \cup \{\perp\} \). Define \( N := \{c_0, c_1\} \).

We claim that \( \langle c_0, c_1 \rangle \in A^f \) is such that \( \langle c_0, c_1 \rangle \notin C^J \). Consider any element \( \langle e, f \rangle \in A^f \) such that \( \langle\langle c_0, c_1\rangle, \langle e, f\rangle\rangle \in R^J \). By the construction of \( J \), we have that \( \langle c_0, c_1 \rangle, \langle e, f \rangle \in R^J \), and thus \( e^J = d_i^J \) and \( f^J = d_j^J \) for some \( i \in \{0, \ldots, n-1\}, j \in \{1, \ldots, n\} \). Since the constants \( d_i \) are unequal to \( c_0, c_1 \), this implies that \( e, f \notin N \), and thus \( e = f = d_i = d_j \). Therefore, \( \langle e, f \rangle \) is equal to \( d_i^J \) for some \( i \in \{1, \ldots, n-1\} \) whenever \( \langle\langle c_0, c_1\rangle, \langle e, f\rangle\rangle \in R^J \), as required for \( \langle c_0, c_1 \rangle \notin C^J \).

– If \( C = \forall 0 R.\neg D \) with \( D \notin D^+_H \) then define \( [c_0, c_1] \in C \) := \( [c_0, c_1] \in D \) \( \cup [R(c_0, c_0), R(c_1, c_1), R(c_1, c_0), R(c_1, c_1)] \), where \( N \) is defined as for \( [c_0, c_1] \in D \).

The claim follows by induction as before.

– If \( C = \forall 1 R.\neg D \) with \( D \notin D^+_H \cup \{\perp\} \) then \( [c_1, c_0] \in C \) := \( [c_0, D, c_1 \notin D] \) \( \cup [R(c_0, c_0), R(c_1, c_1)] \), where \( N \) is defined to be the set \( M \) as given in Lemma 17.

The claim is a consequence of Lemma 17.

– If \( C = \forall n R.\neg D \) with \( n \geq 2 \) then consider fresh individual symbols \( c_2, \ldots, c_n \) and define \( [c_0, c_1] \in C \) := \( [c_0, c_1] \notin D \) \( \cup [c_2, \ldots, c_n \notin D] \cup [R(c_i, c_j) \cup \{R(c_1, c_i) \cup \{R(c_1, c_0), R(c_1, c_0), R(c_1, c_1), R(c_1, c_0), R(c_1, c_1)]\} \cup \{c_0, c_1, c_2, \ldots, c_n \notin D\} \). The latter are not in \( D \) by Lemma 16.

Atomic concepts, nominals, Self restrictions, and their negations do not occur since \( C \notin D^+_H \).

The previous result is used in the following proposition to show that certain kinds of atmost-concepts are generally excluded from DLP, even if they occur as subconcepts only.

**Proposition 6.** Given a structural concept \( C \notin \{\top, \perp\} \) in DLP normal form, the following three statements are equivalent:

– \( C \notin D^+_H \)

– \( C \) has a subconcept \( D \notin D^+_H \).

– \( C \) contains a subconcept \( \forall k. S.\neg F \) such that \( F \in D^+_H \) and \( F \notin D_{mn-1} \) for all \( l \geq 0 \).

and:

(a) \( k = 0 \) and \( F \notin D^+_H \cup \{\perp\} \), or

(b) \( k = 1 \) and \( F \notin D^+_H \cup \{\perp\} \), or
(c) \( k \geq 2 \).

If these statements hold and, in addition, \( C \not\in D_{\mathcal{S},n} \) for all \( n \geq 0 \), and \( C \not\in C_{\mathcal{F},+} \), then \( C \) cannot be emulated in datalog.

**Proof.** Note that the preconditions on \( C \) imply that \( \{C\} \) is satisfiable. The claimed equivalence is easily verified by considering the grammar for \( D_{T,n}^+ \) as given in Fig. 5, where it should be noted that some cases are inherited from \( D_{T_H}^+ \) and \( D_{T_E}^+ \). Also observe that \( F \in D_{T,n}^+ \) is thus equivalent to saying that \( F \) has no subconcept \( E \not\in D_{T,n}^+ \).

First, we define an auxiliary theory that requires \( \preceq k S \neg F \) to be non-empty in order for \( C \) to be satisfied. As before, we sometimes mix first-order logic and DL to denote an arbitrary \( \mathcal{FOL}_{\mathcal{L}} \) theory that represents the first-order semantics of this combination. Given a constant symbol \( c \), and a subconcept \( D \) of \( C \) such that \( \preceq k S \neg F \) is a subconcept of \( D \), we recursively construct a \( \mathcal{FOL}_{\mathcal{L}} \) theory \( T(c, D) \):

- If \( D = \preceq k S \neg F \), then \( T(c, D) := 0 \).
- If \( D = D_1 \cap D_2 \) with \( \preceq k S \neg F \) a subconcept of \( D_1 \), then \( T(c, D) := T(c, D_1) \).
- If \( D = D_1 \cup D_2 \) with \( \preceq k S \neg F \) a subconcept of \( D_1 \), then \( T(c, D) := T(c, D_1) \cup \{c \not\in D_2\} \).
- If \( D = \exists n R D' \), then consider fresh constants \( c_1, \ldots, c_n \) and define \( T(c, D) := \{\forall x R(c, x) \rightarrow \bigvee_{1 \leq i \leq n} c_i = x\} \cup T(c_0, D') \).
- If \( D = \preceq n R \neg D' \) (with \( R \neq S \)), then consider fresh constants \( c_0, \ldots, c_n \) and set \( T(c, D) := \{\bigwedge_{0 \leq i < n} R(c_i) \land \bigwedge_{0 \leq i < j \leq n} \neg(c_i = c_j)\} \cup \{c_0, \ldots, c_n \not\in D\} \cup T(c_0, D') \).

Note that \( T(c, D) \) is satisfiable, due to the subconcept \( \preceq k S \neg F \) of \( D \), since \( C \) is in DLP normal form. Now the theory \( T \) is defined as \( T := T(c, C) \) for some fresh constant \( c \). It is easy to see that \( T \cup \{C\} \) is satisfiable, and that \( T \cup \{C\} \cup \{\preceq k S \neg F \subseteq \bot\} \) is unsatisfiable.

Consider the case \( k = 0 \). Let \( a \) and \( b \) be fresh constants. We use the construction of Lemma 18 to ensure that every element in the respective product interpretations has an \( S \)-successor \( (a, b) \) in \( \neg F \), and \( N \) denotes the according set of constant symbols as in the definition of \( \{a, b \in F\}_x \). Some care is needed to ensure that the auxiliary theory \( T \) remains true in any such product interpretation. Thus define \( T' := T \cup \{ \neg\phi \upharpoonright a \} \mid a \in N \), \( d \) occurs in \( T \} \cup \{\forall x R(c, a) \wedge S(x, b)\} \cup \{a, b \in F\}_x \). It is not hard to see that \( T' \cup \{C\} \) is satisfiable. For an arbitrary model \( I \) of \( T' \cup \{C\} \), consider the product interpretation \( J' := I \times \mathcal{L} C \). Since \( J \) satisfies \( \forall x S(x, a) \land S(x, b) \) (by Proposition 5), we find \( \langle \delta, \langle a, b \rangle \rangle \in S_J \) for all \( \delta \in A^J \). Thus Lemma 18 entails \( J \models \preceq 0 S \neg F \subseteq \bot \).

Moreover, \( J \) satisfies \( T \). This is a consequence of Proposition 5 for all axioms of \( T \) that are in datalog. The only axioms for which this is not the case are of the form \( \forall x R(c, x) \rightarrow \bigvee_{1 \leq i \leq n} c_i = x \). Consider any element \( \langle e, f \rangle \in A^J \) such that \( \langle c^J, \langle e, f \rangle \rangle \in R^J \). By the construction of \( J \), we have that \( \langle c^J, e^J \rangle, \langle c^J, f^J \rangle \in R^J \), and thus \( e^J = c_i^J \) and \( f^J = c_j^J \) for some \( i, j \in \{1, \ldots, n\} \). Since all constants in \( N \) must be unequal to constants \( c_i \), this implies that \( e, f \not\in N \), and thus \( e = f = c_i = c_j \). Therefore, \( \langle e, f \rangle \) is equal to \( c_i^J \) for some \( i \in \{1, \ldots, n\} \) whenever \( \langle c^J, \langle e, f \rangle \rangle \in R^J \), so that the considered axiom of \( T \) is indeed satisfied.

Since \( T \cup \{C\} \cup \{\preceq k S \neg F \subseteq \bot\} \) is unsatisfiable, this implies \( J \not\models \{C\} \). This establishes the preconditions for Lemma 15 (for the case \( T_1 = T_2 \)) and thus shows the claim.
The other cases \( k = 1 \) and \( k \geq 2 \) are very similar, using constructions \( \| a \in F, b \notin F \| \) \(_k \) and \( \| c_1, \ldots, c_k \notin F \| \) of Lemma 17 and 16. For \( k = 1 \), it is admissible that \( a^T \notin F^T \) is an \( S \)-successor of all elements. For \( k \geq 2 \), \( k \) such \( S \)-successors \( c_1^T, \ldots, c_k^T \notin F^T \) are allowed. In either case, the product construction generates further \( S \)-successors that require \( \leq k.S.\neg F \) to be empty.  

Observe how the previous proof depends on using the second pre-condition of Lemma 15 where a single model is multiplied with itself. This is essential to ensure that the auxiliary theory \( T \) is satisfied in the product, even though it contains non-datalog axioms. The above result also marks a case where we really need product constructions that are different from the canonical product that uses all pairs of (named) individuals as the new interpretation domain. The auxiliary theory \( T \) in the above case would not generally be satisfied in a canonical product: the non-datalog axioms introduced for at least-restrictions require a fixed set of successor individuals, whereas a canonical product contains additional successors that correspond to pairs of the original individuals.

For the remaining steps of the proof, we use some additional auxiliary constructions. The datalog programs of Definitions 14 and 15 are not suitable to isolate properties that exclude a concept from DLP: to the contrary, they simply enforce certain entailments to override any complex semantic effects. The following definition therefore provides us with knowledge bases that can be used to “measure” information about the extension of a concept \( C \) without enforcing \( C \subseteq \bot \). The underlying intuition is that non-emptiness of some concepts can be ensured to entail positive information. The construction thus can be viewed as a generalization of the construction in Lemma 4 to the more complex case of \( SROIQ \).

We provide two cases: \( \| c \in C \leadsto A \|_B \) is used to detect whether a constant \( c \) is in \( C \), while \( \| C \leadsto A \|_{B \subseteq} \) is used to detect if \( C \) is generally non-empty. Both constructions can only work (in \( DLP \)) if \( C \) “contains” positive information, i.e. if it is not in \( D_B \). Note that the constructions can be considered as specializations of \( \| a \notin C \| \) and \( \| C \subseteq A \|_{\subseteq} \).

**Definition 16.** Consider a structural concept \( C \) in DLP normal form such that \( C \notin D_B \cup \{ \bot, \top \} \cup D_{\geq n \rightarrow k} \) for some \( k \geq 0 \). For individual names \( c_0, \ldots, c_k \) and concepts \( A_0, \ldots, A_k \in D_B \), a datalog program \( \| c_0, \ldots, c_k \in C \leadsto A_0, \ldots, A_k \|_B \) is defined recursively as follows:

- If \( C \) has the form \( A_0 \) \( \{ I \} \) or \( \exists R. \text{Self} \), then \( \| c_0, \ldots, c_k \in C \leadsto A_0, \ldots, A_k \|_B := \bigcup_{0 \leq i \leq k} \text{datalog}(\{ c_i \} \cap C \subseteq A_i) \).
- If \( C = D_1 \cap D_2 \) with \( D_1 \notin D_B \), then w.l.o.g. \( D_1 \) is not a conjunction and thus \( D_1 \notin D_{\geq m \rightarrow k} \) for all \( m \geq 0 \). Define \( \| c_0, \ldots, c_k \in C \leadsto A_0, \ldots, A_k \|_B := \{ c_0, \ldots, c_k \in D_1 \leadsto A_0, \ldots, A_k \} \).
- If \( C = D_1 \cup D_2 \) with \( D_1 \notin D_B \), then \( D_1, D_2 \notin D_{\geq m \rightarrow k} \). Set \( \| c_0, \ldots, c_k \in C \leadsto A_0, \ldots, A_k \|_B := \{ c_0, \ldots, c_k \in D_1 \leadsto A_0, \ldots, A_k \} \cup \{ c_0, \ldots, c_k \in D_2 \} \).
- If \( C = \exists n R. D \) with \( n \geq 1 \), then \( \| c_0, \ldots, c_k \in C \leadsto A_0, \ldots, A_k \|_B := \{ R(c_i, x) \leadsto A(c_i) \mid 0 \leq i \leq k \} \).
- If \( C = \leq n R. \neg D \), then, for a fresh constant \( d \) and fresh concept name \( B \), define \( \| c_0, \ldots, c_k \in C \leadsto A_0, \ldots, A_k \|_B := \{ d \in D \leadsto B \} \cup \{ R(c_i, d), B(d) \leadsto A(c_i) \mid 0 \leq i \leq k \} \).
- If $C = \leq n R. \neg D$ with $n \geq 1$, then consider fresh constants $d_i$ ($i = 0, \ldots, n$). Define $[c_0, \ldots, c_k \in C \leadsto A_0, \ldots, A_k]_B := \{ R(c_i, d_j) \mid 0 \leq i \leq k, 0 \leq j \leq n \} \cup \{ d_j \approx d_i \rightarrow A(c_i) \mid 0 \leq j < l \leq n, 0 \leq i \leq k \} \cup \{ d_0, \ldots, d_n \notin D \}$. Moreover, if $C \notin D_{\text{aux}}$ for all $k \geq 0$, then a datalog program $\{ C \leadsto A \}_B$ is defined recursively as follows:

- If $C$ has the form $A$, then $\{ C \leadsto A \}_B := \text{datalog}(C \subseteq A)$.
- If $C = D_1 \cap D_2$ with $D_1 \notin D_B$ and $D_1 \notin D_{\text{aux}}$ for all $n \geq 0$, then $\{ C \leadsto A \}_B := \{ D_1 \leadsto A \}_B$.
- If $C = D_1 \cup D_2$ with $D_1 \notin D_B$, then $\{ C \leadsto A \}_B := \{ D_1 \leadsto A \}_B \cup \{ D_2 \subseteq A \}_B$.
- If $C = \exists R. D$ with $n \geq 1$, then $\{ C \leadsto A \}_B := \{ (R(x, y) \rightarrow A(x)) \}$.
- If $C = \leq 0 R. \neg D$, then, for a fresh constant $c$ and fresh concept name $B$, define $\{ C \leadsto A \}_B := \{ \{ c \in D \rightarrow B \} B \cup \{ (R(c, x), B(c) \rightarrow A(x)) \}$.
- If $C = \leq n R. \neg D$ with $n \geq 1$, then consider fresh constants $c_i$ ($i = 0, \ldots, n$). Define $\{ C \leadsto A \}_B := \{ (R(c_i, x) \mid 0 \leq i \leq n) \} \cup \{ c_i \approx c_j \rightarrow A(x) \mid 0 \leq i < j \leq n \} \cup \{ c_0, \ldots, c_n \notin D \}$.

It should be noted that the cases in the previous definition are indeed exhaustive: side conditions usually are only provided to specify a particular situation that can be assumed without loss of generality. Conditions that follow from the assumptions are omitted. Observe that the necessary conditions for recursion are satisfied in all cases of the definition. The choice of $D_1$ in the cases for $C = D_1 \cap D_2$ is possible since we disregard nesting order of $\cap$: if there is some $D_1 \notin D_B$, then there is some such $D_1$ that does not have a $C_i$ disjunct (which is in $D_B$) while still $D_1 \notin D_B$. But then this $D_1 \notin D_{\text{aux}}$ for all $m \geq 0$ as required.

It is not hard to see that, given the preconditions of Definition 16, we find that $[c_0, \ldots, c_k \in C \leadsto A_0, \ldots, A_k]_B \models \bigcup_{0 \leq i \leq k} C \cap \{ c_i \} \subseteq A_i$ and $\{ C \leadsto A \}_B \models C \subseteq A$. Notably, the case $C = \leq n R. \neg D$ uses different approach than the other cases: the positive information used to entail non-emptiness of $A$ is found in the equality relations that are implied between auxiliary constants $d_i$.

Observe that the datalog programs of Definition 16 again may significantly constrain the extension of $C$. For example, if $C = \leq 1 R. \neg \bot$ then $\{ C \leadsto A \}_B$ is only satisfied by interpretations that entail either $C \subseteq \bot$ or $\top \subseteq C$. This may entail $\top \subseteq A$, so we will only use $\{ C \leadsto A \}_B$ if $\top \subseteq A$ or $C \subseteq \bot$ is satisfiable. Non-emptiness of $C$ might also be unavoidable, so one cannot assume that $\{ C \leadsto A \}_B \models [A \subseteq \bot]$ is satisfiable. Yet, the remaining freedom will generally suffice for our purposes.

Another noteworthy fact is that $[c_0, c_1 \in C \leadsto A_0, A_1]_B$ is not the same as $[c_0 \in C \leadsto A_0]_B \cup [c_1 \in C \leadsto A_1]_B$, which is the reason why the definition must explicitly include cases with $k > 0$. To see this, consider $C = (\leq a \cap \neg \{ b \}) \cup B$. Then $[c_0, c_1 \in C \leadsto A_0, A_1]_B = \{ B(c_0) \rightarrow A_0(c_0), B(c_1) \rightarrow A_0(c_1), c_0 \approx a, c_1 \approx b \}$ but $[c_0 \in C \leadsto A_0]_B \cup [c_1 \in C \leadsto A_1]_B = \{ B(c_0) \rightarrow A_0(c_0), B(c_1) \rightarrow A_0(c_1), c_0 \approx a, c_1 \approx a \}$, the latter entails the unwanted consequence $c_0 \approx c_1$ since the auxiliary programs $[c_i \approx \neg \{ a \} \cap \neg \{ b \}]$ are constructed independently for $i = 0, 1$ instead of using $[c_0, c_1 \notin \{ a \} \cap \neg \{ b \}]$.

The following lemma provides some important ingredients for showing maximality of $DLP$, since it establishes the pre-conditions of Lemma 15 for broad classes of concepts.
Lemma 19. Let $C \in \mathcal{D}_n^+$ be a structural concept expressions in DLP normal form, let $I$ be the set of constants of the given signature, and let $a, b, c \in I$ be arbitrary constants not occurring in $C$.

1. If $C \notin D_H$, then one of the following is true:
   - There is a theory $T$ and a set of constants $N \subseteq I$ with $a, b \in I$ such that: given an arbitrary model $I$ of $\{\{a\} \cup \{b\} \subseteq C\} \cup T$, we find that $\mathcal{J} = I \times_{(N \times N)} I$ is such that $(a, b) \notin C^\mathcal{J}$.
   - There are theories $T_1, T_2$ such that: given arbitrary models $I_j$ of $\{\{a\} \cup \{b\} \subseteq C\} \cup T_i (i = 1, 2)$, we find that $\mathcal{J} = I_1 \times_{(N \times N)} I_2$ is such that $(a, b) \notin C^\mathcal{J}$.

2. If $C \in D_m$, then there are theories $T_1, T_2$ such that: given arbitrary models $I_1$ of $\{\{c\} \subseteq C\} \cup T_i (i = 1, 2)$, we find that $\mathcal{J} = I_1 \times_{(N \times N)} I_2$ is such that $c^\mathcal{J} = \langle c, c \rangle \notin C^\mathcal{J}$.

In all cases, models $I_1$ and $I_2$ as described in the claims exist.

Proof. By Proposition 6, $C \in \mathcal{D}_n^+$ implies that $D \in \mathcal{D}_n^+$ for all subconcepts $D$ of $C$.

We start by considering claim (1). Claim (2) is shown independently below, so if $C \notin D_n$, then we obtain theories $T_1$ and $T_2$ as in claim (2) for some fresh constant $c$. It is easy to see that the theories $T_i^* := T_i \cup \{a \sim c, b \sim c\} (i = 1, 2)$ suffice for establishing claim (1). It remains to show claim (1) for cases where $C \in D_n$. An easy induction can be used to show that $D_n^* \subseteq D_H$. Hence, using our assumption that $C \notin D_H$, we can also conclude $C \notin D_H$.

The only remaining cases for claim (1) therefore are such that $C \notin D_n^*$, so that Lemma 18 can be applied. Define $T_1 := T_2 := \llcorner a, b \in C \lrcorner$, and define $N$ as in the lemma. The claim follows from Lemma 18.

For claim (2), we construct theories $T_1 = T_1(c, C)$ and $T_2 = T_2(c, C)$ for a fresh constant $c$ as in the claim. The proof proceeds by induction over the structure of $C$. Note that $C$ cannot be an atomic class, nominal, Self restriction, or the negation thereof.

Consider the case $C = D_1 \cap D_2$. Without loss of generality, we find that $D_1 \notin D_n$. Applying the induction hypothesis, we obtain theories $T_i(c, C) := T_i(c, D_1) (i = 1, 2)$ that satisfy the claim.

Consider the case $C = D_1 \cup D_2$. As a first case, assume that $D_1 \notin D_n$. Then we can define theories $T_i(c, C) := T_i(c, D_1) \cup \llcorner c \notin D_2 \lrcorner (i = 1, 2)$. The claim then follows from the induction hypothesis together with the fact that every product interpretation constructed from models of $T_i(c, C) (i = 1, 2)$ must also satisfy $\llcorner c \notin D_2 \lrcorner$ by Proposition 5. The case $D_2 \notin D_n$ is similar.

Now assume that $C = D_1 \cap D_2$ with $D_1, D_2 \notin D_n$. Using fresh concept names $A_1, A_2$ and the construction of Definition 16, define $T_i(c, C) := \{A_i(c) \rightarrow \bot\} \cup \bigcup_{i = 1, 2} \llcorner c \in D_j \lrcorner \cup A_j \lrcorner$ for $i = 1, 2$. Then any product interpretation $\mathcal{J}$ of any two models of $T_i(c, C) (i = 1, 2)$ satisfies $\bigcup_{i = 1, 2} \llcorner c \in D_j \lrcorner \cup \{A_i(c) \rightarrow \bot\}$, and hence $\mathcal{J} \notin \llcorner c \notin D_1 \lrcorner (i = 1, 2)$ as required.

Consider the case $C = \llcorner 0 R \lrcorner \lrcorner$ with $D \notin D_H$. Since $C \in \mathcal{D}_n^+$, we find $D \in \mathcal{D}_n^+$.

Using $\mathcal{D}_n^* \cap D_n \subseteq D_H$ as above, we conclude that $D \notin D_n$, which allows us to apply the induction hypothesis. Consider a fresh individual name $d$ and define $T_i(c, C) := T_i(d, D) \cup \llcorner R(c, d) \lrcorner (i = 1, 2)$. Given models $I_i$ of $T_i(c, C) (i = 1, 2)$, the induction
hypothesis implies that $\mathcal{J} := I_1 \times_{\mathcal{D}_1} I_2$ does not satisfy $\{d\} \subseteq D$. Since $\mathcal{J} \models R(c, d)$, we conclude $\mathcal{J} \not\models \{c\} \subseteq C$.

Consider the case $C = \leq 1 R \not\models D$ with $D \in \mathcal{D}_B$ and $D \not\in \mathcal{D}_{2^{n-1}}$. Using fresh symbols $c_1, c_2, A_1, A_2$, we define $T_i(c, C) := \{A_i(c_i) \rightarrow \bot\} \cup \{c_1, c_2 \in D \rightarrow A_1, A_2\} \cup \{R(c, c_1), R(c, c_2)\}$ for $i = 1, 2$. Using similar arguments as in the last case of $C = D_1 \cup D_2$, we find that no product interpretation of models of $T_i(c, C)$ ($i = 1, 2$) can satisfy $\{c\} \subseteq C$.

Consider the case $C = \leq n R \not\models D$ with $n \geq 2$ and $D \not\in \mathcal{D}_{2^{n-1}}$. Using fresh individuals symbols $c_0, \ldots, c_n$, set $T := \{c_0, \ldots, c_n \not\in D\} \cup \{R(c, c_i) \mid 0 \leq i \leq n\}$. We define $T_1(c, C) := T \cup \{c_i \approx c_j \rightarrow \bot \mid 1 \leq i < j \leq n\}$ and $T_2(c, C) := T \cup \{c_i \approx c_j \rightarrow \bot \mid 0 < i < j \leq n-1\}$. Thus, any model of $\{c\} \subseteq C \cup T_1(c, C) \cup \{\{c\} \subseteq C \cup T_2(c, C)\}$ entails $c_0 \approx c_1 \approx c_n$, which is a contradiction.

Consider the case $C = \geq 1 R \not\models D$. Then $D \in \mathcal{D}_A$ and $D \not\in \mathcal{D}_B$. For a fresh constant $d$, define $T_i(c, C) := T(d(d), D) \cup \{R(c, x) \rightarrow d \approx x\}$ for $i = 1, 2$. The claim follows from the induction hypothesis and the fact that every considered product interpretation also satisfies $\{R(c, x) \rightarrow d \equiv x\}$.

Consider the case $C = \geq n R \not\models D$ with $n \geq 2$ and $D \not\in \mathcal{D}_{2^n}$. Without loss of generality, we can assume that $D$ is of the form $C_1 \cup \ldots \cup C_p \cup E$ ($p \geq 1$) where no $C_i$ is a disjunction, $C_i \not\models \mathcal{D}_B$ for $i = 1, \ldots, p$, and $E \in \mathcal{D}_B \cup \{\bot\}$. For the following argument, we use $E = \bot$ to cover the case where no such $E$ is given in the original DLP normal form. Note that $E$ might be a disjunction but cannot be $\top$.

First assume that there is some $\mathcal{F} \subseteq \{E, C_1, \ldots, C_p\}$ such that $F \in \mathcal{D}_{2^{n-k}}$ for some $k \geq 0$. Since $\mathcal{F}$ is in DLP normal form, it is a disjunction that contains some disjunct in $C_{n-m} (m \geq 1)$. All subconcepts of $D$ are assumed to be in $\mathcal{D}_A$, so if $m \leq n^2 - n$ then $D \not\in \mathcal{D}_{2^n}$, a contradiction. Thus $D$ is of the form $D_1 \cup D_2$ with $D_1 \in C_{n-m}$ and $m > n^2 - n$. Moreover, $D_2 \not\in \mathcal{D}_A$ since otherwise we would find $D \in \mathcal{D}_A \subseteq \mathcal{D}_{2^n}$.

The set of constants in $D_2$ is denoted as $\text{ind}(D_1) = \{c_1, \ldots, c_m\}$. Let $p_1, p_2, \ldots, p_{n^2-n}$ denote a sequence of all pairs $p_i = \langle d_1, d_2 \rangle$ of constants $d_1, d_2 \in \{c_1, \ldots, c_m\}$ with $d_1 \neq d_2$. The order is inessential, but some order is needed for notational purposes. Define auxiliary theories $T_i(c, C) := \{\forall x R(c, x) \rightarrow \bigvee_{1 \leq j \leq n} c_j \approx x\} \cup \bigcup_{1 \leq j \leq n} T_i(c, C)(c_j, D_2) \cup \{c_j \approx d_i \mid n < j < m, p_{j-n} = \langle d_1, d_2 \rangle\}$. Observe that the first component in this definition refers only to the first $n$ constants $c_1, \ldots, c_n$, the second part is specified for all $m$ constants, and the third component refers to the last $n-m$ constants $c_{n+1}, \ldots, c_m$ only.

To see that these theories satisfy the claim, consider models $I_i$ of $\{c\} \subseteq C \cup T(c, C)$ ($i = 1, 2$), and let $\mathcal{J} := I_1 \times_{\mathcal{D}_1} I_2$ denote their product. Observe that, by the construction of $T_i(c, C)$, the constants $c_j$ ($1 \leq j \leq m$) are mutually unequal in $\mathcal{J}$. Now consider an arbitrary element $\delta \in A^\mathcal{J}$ such that $(c^\mathcal{J}, \delta) \in R^\mathcal{J}$. By definition of the product, there must be a constant symbol $d$ — possibly an auxiliary constant that did not occur in $C$ — such that $\delta = \langle d, d \rangle$ and $(c^\mathcal{J}, d^\mathcal{J}) \in R^\mathcal{J}$ for $i = 1, 2$. Since the models $I_1$ satisfy $\forall x R(c, x) \rightarrow \bigvee_{1 \leq j \leq n} c_j \approx x$, we conclude that $I_1 \models d = c_j$ and $I_2 \models d = c_k$ for some $c_j$ and $c_k$ (possibly distinct!) $j, k \in \{1, \ldots, n\}$. Thus, there are at most $n^2$ elements $\delta \in A^\mathcal{J}$ such that $(c^\mathcal{J}, \delta) \in R^\mathcal{J}$, since there are at most $n^2$ distinct ways of selecting $j, k$. Now $m$ of those $n^2$ elements are of the form $(c_j^\mathcal{J})$ for some $j = 1, \ldots, m$, and by the induction hypothesis we find that $c_j^\mathcal{J} \not\in D_2^\mathcal{J}$. Since $c_j^\mathcal{J} \not\in D_1^\mathcal{J}$ is immediate, we thus find that
$c^{\vec{J}} \notin D^{\vec{J}}$ for all $j = 1, \ldots, m$. Summing up, we conclude that $\mathcal{J}$ can have most $n^2 - m$ distinct $R$-successors for $c$ which are in $D$. Since $n^2 - m < n^2 - (n^2 - n) = n$, we find that $\mathcal{J} \not\models \{c\} \models \forall n R.D$, as required.

For the remainder of the proof, assume that $F \notin D_{2n-k}$ for all $F \in \{E, C_1, \ldots, C_p\}$ and $k \geq 0$. In particular, we can use the constructions of Definition 15 and 16. Now if $\{\{c\} \models \forall n R. \neg \forall \{c\} \models \forall n R.x$ is unsatisfiable, then $C_1 \cup \ldots \cup C_p \in D_{2n-1}$. Since we assumed that $C_1 \cup \ldots \cup C_p \in D'_{2n}$, this again implies $D \in D^{2n}$. Hence, $\{\{c\} \models \forall n R.x$ must be satisfiable (note that this includes the case $E = \perp$). It is easy to see that $\{\{c\} \models \forall n R. \neg \forall \{c\} \models \forall n R.x$ emulates $\{\{c\} \models \forall n R.C_1 \cup \ldots \cup C_p\}$, and that the claim can thus be established by induction. So for the remaining considerations we can assume that $E$ is not present at all, i.e. that $C \models \forall n R.C_1 \cup \ldots \cup C_p$.

By the assumptions on $C_i$, we can apply Definition 16 and set $T' := \bigcup_{1 \leq i \leq p} (\{c_i \models \forall n R.C_i \models \forall n R.D_i \} \cup \{R(x, y) \wedge A_i(y) \rightarrow B_i(x)\})$ for fresh concept names $A_1, \ldots, A_p, B_1, \ldots, B_p$. It is easy to verify that $\{\{c_i \models \forall n R.C_i \models \forall n R.D_i \} \cup \{R(x, y) \wedge A_i(y) \rightarrow B_i(x)\}\}$ is defined analogously for the second components. Due to the constructions of Definition 12, and thus cannot be a model of $\{\{c\} \models \forall n R.C_1 \cup \ldots \cup C_p\}$, as required.

Now consider the case (B) where $T' \cup \{\{c\} \models \forall n R.C_i \models \forall n R.D_i \} \cup \{R(x, y) \wedge A_i(y) \rightarrow B_i(x)\}$ is consistent. Then there are two disjoint subsets $I_1, I_2 \subseteq \{1, \ldots, p\}$ for which $T_i(c, C) := T \cup \{B_i \in \perp \mid i \in I_1\}$ is such that $T_i(c, C) \cup \{\{c\} \models \forall n R.C_i \models \forall n R.D_i \} \cup \{R(x, y) \wedge A_i(y) \rightarrow B_i(x)\}$ is consistent for $k = 1, 2$, while $T_1(c, C) \cup \{\{c\} \models \forall n R.C_i \models \forall n R.D_i \} \cup \{R(x, y) \wedge A_i(y) \rightarrow B_i(x)\}$ is inconsistent. Every product interpretation of models of $T_i(c, C) (k = 1, 2)$ entails $T$ (by Proposition 5) and $B_i \subseteq \perp$ (by Definition 12), and thus cannot be a model of $\{\{c\} \models \forall n R.C_1 \cup \ldots \cup C_p\}$, as required.

As a second case (B.2), assume that $C_h \notin D_{2h}$, for all $k \geq 0$ since $C_h$ is not a disjunction. Since this implies that $T \cup \{\{c\} \models \forall n R.C_i \models \forall n R.D_i \} \cup \{R(x, y) \wedge A_i(y) \rightarrow B_i(x)\}$ and that $C_h \models \forall n R.D_h$, we find that $\{\{c\} \models \forall n R.C_h \models \forall n R.D_h\}$ emulates $\{\{c\} \models \forall n R.D_h\}$. The claim follows by induction.

Consider the case (B.2.1) where $C_h \notin D_h$. For fresh individuals $c_1, \ldots, c_n$ define $T'' := T' \cup \{R(c, x) \rightarrow \bigvee_{1 \leq i \leq n} c_i \models x\}$. Note that $T'' \cup \{\{c\} \models \forall n R.C_i \models \forall n R.D_i \} \cup \{R(x, y) \wedge A_i(y) \rightarrow B_i(x)\}$ is satisfiable by interpretations $T$ that have $c_i^{\vec{J}} \models c_i^{\vec{J}'}$ as the $n$ distinct $R$-successors of $c$. Define $T_i(c, C) := \bigcup_{1 \leq j \leq n} T_i(c_j, C)$, as each model of $C$ requires one $R$-successor of $c$ in each concept of the form $C_i$, we find that $\{\{c\} \models \forall n R.C_1 \models \forall n R.D_1\}$ emulates $\{\{c\} \models \forall n R.D_h\}$. The claim follows by induction.

To show that this satisfies the claim, consider models $T_i$ of $\{\{c\} \models \forall n R.C_i \models \forall n R.D_i \}$ ($i = 1, 2$). Since the induction hypothesis only applies to named individuals, we introduce $n^2$ fresh constants $\langle c_j, c_k \rangle$ for $j, k \in \{1, \ldots, n\}$. $T_i$ is extended to $T_i'$ over this extended signature by setting $\langle c_j, c_k \rangle^{T_i'} := c_i^{T_i}$, so that $T_i' \models \{c_j \models \forall n R.C_i \models \forall n R.D_i\}$. The extended interpretation $T_i'$ is defined analogously for the second components. Due to the constructions in this proof, for any constants $a, f$, we find that $T_i(e, C_h)$ is the same as $T_i(f, C_h)$.
with \( e \) uniformly replaced by \( f \) \((i = 1, 2)\). Thus, we find that \( I'_i \models T_i(\langle c_j, c_k \rangle, C_h) \) for \( i = 1, 2 \) and all \( j, k \in \{1, \ldots, n\} \). Moreover, \( I'_1 \models \{(c_j, c_k)\} \equiv C_h \) so the induction hypothesis can be applied to obtain \( I'_1 \times (\forall \times \forall) I'_2 \not\models \{(c_j, c_k)\} \equiv C_h \) where \( I' \) denotes the extended set of constants.

It is not hard to see that the interpretations \( J' = I'_1 \times (\forall \times \forall) I'_2 \) and \( J = I_1 \times (\forall \times \forall) I_2 \) are equal (possibly up to renaming of domain elements). In particular, \( J' \) entails \( \langle c_j, c_k \rangle \approx \langle c_j, c_l \rangle, \langle c_r, c_k \rangle \rangle \). Hence we find that \( J' \not\models \{(c_j, c_k)\} \equiv C_h \). Moreover, since \( I_1 \) and \( I_2 \) satisfy \( T'' \), we find that \( \langle c_T, \delta \rangle \in R^J \) implies \( \delta = \langle c_j, c_k \rangle \) for some \( j, k \in \{1, \ldots, n\} \). Thus we obtain \( J' \not\models \{\langle c \rangle \} \subseteq C \) as required.

As the final case (B.2.2), assume that \( C_h \in D_a \). Since \( D \not\in D^{2m} \), we find \( D \not\models C_h \), i.e. \( p > 1 \). We concluded \( \bigcup_{1 \leq p \leq h} C_i \in D^{m-1} \) above for all sub-cases of (B). Hence \( D \) is of the form \( D_1 \sqcup D_2 \sqcup D_3 \) – where we assume that \( m \) is the least natural number for which \( D \) has this form – and \( m \) and \( l \) do not satisfy the relevant conditions in the definition of \( D^{2m} \). Accordingly, we denote \( D \) as \( C_h \sqcup M_1 \sqcup \ldots \sqcup M_m \sqcup L_1 \sqcup \ldots \sqcup L_l \). Since \( M_1, \ldots, M_m, L_1, \ldots, L_l \in D_{\geq 1} \), they are each of the form \( \{d\} \cap C \) for some individual name \( d \) let \( e_1, \ldots, e_m, f_1, \ldots, f_l \) denote these individual names. Set \( r := n - (m + l) \), and consider fresh individual names \( c_1, \ldots, c_r \). Define a set \( X := \{c_1, \ldots, c_r, e_1, \ldots, e_m, f_1, \ldots, f_l\} \) of all constants considered as \( R \)-successors of \( c \).

Using the induction hypothesis, we define \( T_i(c, C) := \{\|e_1\|, \ldots, \|e_m\|, \|f_1\|, \ldots, \|f_l\| \not\in C_h\} \cup \{\langle c_1, \ldots, c_r, e_1, \ldots, e_m, f_1, \ldots, f_l \not\in C_h\} \cup \bigcup_{1 \leq j \leq m} T_i(\langle e_j, M_j \rangle) \cup \{\forall x.R(c, x) \rightarrow \bigvee_{d \in X} d \equiv x\} \) \( i = 1, 2 \). Note that the construction of Lemma 16 is possible: if \( C_h \) would be in \( D^{2m}_a \), then \( C \in D_a \) would imply \( C \in D_b \), which cannot be.

To show that this satisfies the claim, consider models \( I_i \models \{\{c\} \subseteq C \} \sqcup T_i(c, C) \) \((i = 1, 2)\), and let \( J = I_1 \times (\forall \times \forall) I_2 \) be the corresponding product interpretation. By the constructions of \( T_i(c, C) \), we obtain that \( \langle c_T, \delta \rangle \in R^J \) implies \( \delta = \langle a, b \rangle \) for some \( a, b \in X \). We distinguish various cases:

- If \( a, b \in \{e_1, \ldots, e_m, f_1, \ldots, f_l\} \) and \( a \neq b \), then \( \langle a, b \rangle \not\in E^J \) for all \( E = M_1, \ldots, M_m, L_1, \ldots, L_l \). Moreover, \( \langle a, b \rangle \not\in C_h \) by Lemma 16.
- If \( a = b = e_j \) for some \( j = 1, \ldots, m \), then \( \langle a, b \rangle \not\in C_h \) again by Lemma 16. As above, \( \langle a, b \rangle \not\in L_i^J \) for all \( i = 1, \ldots, l \). A similar argument shows \( \langle a, b \rangle \not\in M_i^J \) for all \( i = 1, \ldots, m \) with \( i \neq j \), whereas \( \langle a, b \rangle \not\in M_i^J \) follows by the induction hypothesis.
- If \( a \in \{e_1, \ldots, e_m, f_1, \ldots, f_l\} \) and \( b \in \{c_1, \ldots, c_r\} \), then \( \langle a, b \rangle \not\in C_h \) follows from Lemma 17. The conclusion \( \langle a, b \rangle \not\in E^J \) for all \( E = M_1, \ldots, M_m, L_1, \ldots, L_l \) follows as before.

In each of these cases, we thus find that \( \langle a, b \rangle \not\in D^J \). Therefore, the only elements \( \langle a, b \rangle \) of \( D^J \) that might be in \( D^J \) are such that either \( a = b \in \{f_i, \ldots, f_l\} \) or \( a, b \in \{c_1, \ldots, c_r\} \). This yields a maximum of \( l + r^2 \) \( R \)-successors for \( c^J \). Since \( D \not\in D^{2m} \), we find that \( r(r - 1) < m \) (the case \( r \leq 0 \) cannot occur for any case under (B)). Equivalently, \( r^2 - r < m \) which in turn is equivalent to \( r^2 - n + m + l < m \). But then \( r^2 + l < n \), and we find \( J \not\models \{c\} \subseteq C \), as required.

The previous lemma already suffices to exclude a significant amount of axioms from DLP:

\[ \]
Corollary 1. Let $C$ be a structural concept expression in DLP normal form, let $A$ be a fresh concept name, and let $c$ be a fresh constant symbol.

1. If $C \notin D_H \cup \{\top, \bot\}$, then $A \subseteq C$ cannot be emulated by any datalog program.
2. If $C \notin D_H \cup \{\top, \bot\}$, then $\{c\} \subseteq C$ cannot be emulated by any datalog program.
3. If $C \notin D_H \cup \{\top, \bot\}$, and $C \notin D_{2n}$ for all $n \geq 0$, and $C \notin C_{\neq \top}$, then $C$ cannot be emulated by any datalog program.

Proof. If $C \notin D_H^\ast$, then the result follows from Proposition 6 in all cases. Thus assume that $C \in D_H^\ast$ for the remainder of the proof.

For claim (1), consider fresh individual symbols $a$ and $b$, and construct $T_1$ and $T_2$ as in Lemma 19 (1). Define $T_i' := T_i \cup \{A(a), A(b)\}$ for $i = 1, 2$. Then $T_1$ and $T_2$ satisfy the preconditions of Lemma 15 for the knowledge base $KB = \{\{a\} \subseteq A, \{b\} \subseteq A, A \subseteq C\}$. In particular, $T_i \cup \{A \subseteq C\}$ is satisfiable since $C$ is in DLP normal form and $C \neq \bot$. This suffices to establish the claim.

For claim (2) and (3), we can directly use the theories $T_1$ and $T_2$ of Lemma 19 (2) and (1), respectively. To ensure that the preconditions of Lemma 15 hold for claim (3), we need to ensure that $\{C\} \cup T_i$ is satisfiable for $i = 1, 2$. To this end, $C \notin C_{\neq \top} \cup \{\bot\}$ ensures that $\{C\}$ is satisfiable. $C \notin D_{2n}$ for all $n \geq 0$ ensures that $C$ is satisfiable by interpretations of arbitrary domain sizes, and it is not hard to see that $\{C\} \cup T_i$ is consistent when considering the construction in Lemma 19.

The previous result already covers a significant amount of concept expressions that are not in $\{\top, \bot\} \cup D_H \cup D_{\neq} \cup C_{\neq}$. It remains to show that concepts in $D_{2n} \setminus (D_{\neq} \cup C_{\neq})$ for some $n \geq 1$ cannot belong to DLP.

Lemma 20. Let $C$ be a structural concept expression in DLP normal form such that $C \notin \{\top, \bot\}$, and $C \in D_{2n} \setminus (D_{\neq} \cup C_{\neq} \cup D_H)$ for some $n \geq 1$. Then $C$ cannot be emulated by any datalog program.

Proof. Observe that, for any $m \geq 1$, we find $C_{\leq m}^p \subseteq D_{\leq m}^p \subseteq C_{\leq m} \subseteq C_{\leq m+1}$. We define the degree $d(D)$ of a concept expression $D$ as follows. If $D \in C_{\leq m}$ for some $m \geq 1$, then let $d(D)$ be the largest such $m$. Otherwise, if $D \in D_H^\ast$, then define $d(D) := 1$. Otherwise set $d(D) := 0$. Now since $C \in D_{\neq}$ it is of the form $C = \left(\bigcup_{i=1}^{n-1} (c_i \cap C_i) \cup \ldots \cup (c_n \cap C_n)\right)$, and we can assume that $d(C_i) \leq d(C_{i+1})$ for all $i = 1, \ldots, n-1$. Using this notation, it is not hard to see that $C \notin D_{\neq}$ is equivalent to saying that $d(C_i) < i$ for some $i = 1, \ldots, n$.

First consider the case that $i > 1$. We find that $C$ is semantically equivalent to $(c_1 \cap C_1) \cup \ldots \cup (c_i \cap C_i)$. To see this, assume that $n \geq i$. Every model of $C$ has at most $n$ elements in its domain. Since $d(C_n) \geq n$ by construction, $C_n \in C_{\leq n}$. By Lemma 6, we thus obtain $C_n \subseteq C$ as a consequence of $C$, showing that $C$ is equivalent to $(c_1 \cap C_1) \cup \ldots \cup (c_{n-1} \cap C_{n-1})$. The claim thus follows by induction.

Now $C_j \notin C_{\leq j}$ holds for all $j \leq i$. Using Lemma 6, we thus find that $(c_i) \subseteq C_i$ is satisfiable by models of at most $i$ elements in their domain. By structurality of $C$, we find that $C$ is satisfiable, and clearly $C$ is only satisfied by models with exactly $i > 1$ domain elements. Finite domain sizes can be enforced by $\text{FOL}_m$ theories, and hence must be preserved by emulation. But domain sizes greater than 1 are not preserved by the product construction of Definition 12, so the fact that $C$ cannot be emulated in datalog is a consequence of Proposition 5.
Consider the case $i = 1$. Using the same argument as above, we find that $C$ is semantically equivalent to $\{c_1\} \cap C_1$. By construction, $C_1 \not= D_i^n$. The claim is now shown by a miniaturized version of the proof steps that we used to establish Corollary 1, where relevant constructions and arguments largely collapse due to the requirement that the domain of interpretation is unary. We first provide two auxiliary constructions for the “propositional” variants of Definition 14 and 16. Given a structural concept $D \not= D_i^n$ and a constant $d$, recursively construct a datalog program $\llbracket d \not= D \rrbracket^p$ as follows:

- If $D \in C_{\perp} = 1$ then $\llbracket d \not= D \rrbracket^p := \emptyset$.
- If $D$ is of the form $A$, $\neg A$, $\neg (\neg I)$, $\exists R. \text{Self}$, or $\neg \exists R. \text{Self}$, then $\llbracket d \not= D \rrbracket^p := \text{datalog}(\{d \in D\} \subseteq \neg D)$.
- If $D = D_1 \cap D_2$ with $D_1 \not= D_i^n$, then $\llbracket d \not= D \rrbracket^p := \llbracket d \not= D_1 \rrbracket^p$.
- If $D = D_1 \cup D_2$ with $D_1, D_2 \not= D_i^n$, then $\llbracket d \not= D \rrbracket^p := \llbracket d \not= D_1 \rrbracket^p \cup \llbracket d \not= D_2 \rrbracket^p$.
- If $D = \leq R. \neg D'$ with $D' \not= D_i^n$, then $\llbracket d \not= D \rrbracket^p := \llbracket d \not= \neg D' \rrbracket^p$.
- If $D = \geq R. D'$ with $n > 0$, then $\llbracket d \not= D \rrbracket^p := \{\neg R(d, d)\}$.

If $D \not= D_i^n$ then, for a concept name $A$, we recursively construct a datalog program $\llbracket d \cap D \subseteq A \rrbracket^p$ as follows:

- If $D$ has the form $A$ or $\exists R. \text{Self}$, then $\llbracket d \cap D \subseteq A \rrbracket^p := \text{datalog}(\{d \cap D \subseteq A\})$.
- If $D = D_1 \cap D_2$ with $D_1 \not= D_i^n$, then $\llbracket d \cap D \subseteq A \rrbracket^p := \llbracket d \cap D_1 \subseteq A \rrbracket^p$.
- If $D = D_1 \cup D_2$ with $D_1, D_2 \not= D_i^n$, then $\llbracket d \cap D \subseteq A \rrbracket^p := \llbracket d \cap D_1 \subseteq A \rrbracket^p \cup \llbracket d \not= D_2 \rrbracket^p$.
- If $D = \leq R. \neg D'$ with $D' \not= D_i^n$, then $\llbracket d \cap D \subseteq A \rrbracket^p := \llbracket d \cap D' \subseteq A \rrbracket^p \cup \{R(d, d)\}$.
- If $D = \geq R. D'$ with $D' \not= C_{\perp} = 1$, then $\llbracket d \cap D \subseteq A \rrbracket^p := \{R(d, x) \rightarrow A(x)\}$.

To establish the claim, we recursively construct theories $T_1 := T_1(c_1, C_1)$ and $T_2 := T_2(c_1, C_1)$ that satisfy the preconditions of Lemma 15. Note that $C$ cannot be an atomic class, nominal, Self restriction, or the negation thereof.

Consider the case $C = D_1 \cup D_2$ with $D_1, D_2 \not= D_i^n$. It is easy to see that $T_i(c_1, C) := T_i(c_1, D_1) \cup \{\neg A_i(x)\} \cup \bigcup_{j=1}^n \llbracket d \cap D_j \subseteq A_j \rrbracket^p (i = 1, 2)$ satisfy the claim for fresh concept names $A_1, A_2$. Furthermore, if $C = D_1 \cup D_2$ with $D_1 \not= D_i^n$ and $D_2 \not= D_i^n$ then the claim is satisfied by $T_i(c_1, C) := T_i(c_1, D_1) \cup \llbracket d \not= D_2 \rrbracket^p (i = 1, 2)$. Similarly, for the case $C = D_1 \cap D_2$ with $D_1 \not= D_i^n$, the theories $T_i(c_1, D_1) (i = 1, 2)$ satisfy the claim.

Now consider the case $C = \geq n R. D$. Then $n = 1$ and $D \not= D_i^n$. Since $C$ is semantically equivalent to $D$ on singleton domains, the claim follows again by induction. A similar reasoning is possible for the case $C = \leq n R. D$ with $n = 0$ and $D \not= D_i^n$. □

We are now, finally, in a position to state the main theorem of this section.

**Theorem 4.** If $C$ is a concept expression in DLP normal form such that $C \not= DLP$, then $C$ cannot be contained in any DLP language scheme in the sense of Definition 7.

**Proof.** By Definition 10, $C \not= \{\top, \bot\} \cup C_H \cup D_i^n \cup C_{\neg \top}$ for all $n \geq 1$. If $C \not= D_i^n$ for all $n \geq 0$ and $C \not= D_i^n$, then the result follows by Proposition 6. If $C \not= D_i^n$ for all $n \geq 0$ and $C \not= D_i^n$, then the result follows by Corollary 1. If $C \in D_i^n$ for some $n \geq 0$, then the result follows by Lemma 20. □
9 Conclusions and Outlook

DLP provides an interesting example for the general problem of characterising syntactic fragments of a logic that are motivated by semantic properties. We derived and motivated a number of design principles for achieving such a characterisation for DLP, most notably the principles of modularity (closure under unions of knowledge bases) and structurality (closure under non-uniform renaming of signature symbols), and show that the presented DLP language scheme is the largest one possible. Formalisms like our maximal DLP are unnecessarily large for practical applications, but understanding overall options and underlying design principles is indispensable for making an informed choice of DL for a concrete task.

Our work also clarifies the differences between DLP and the DLs $\mathcal{EL}$ and Horn-$\mathcal{SHIQ}$ which can both be expressed in terms of datalog as well. First of all, neither $\mathcal{EL}$ nor Horn-$\mathcal{SHIQ}$ can be completely emulated in datalog (DLP 2). The datalog obtained in these cases still preserves satisfiability even when arbitrary ABox facts (without complex concepts) are added. In other words, $\mathcal{EL}$ and Horn-$\mathcal{SHIQ}$ satisfy a weaker version of DLP 2 where Definition 3 is restricted to test formulae $\varphi$ that are conjunctions of such ABox facts. Under those weakened principles, a larger space of possible DL fragments is allowed, but it is not clear whether (finitely many) maximal languages exist in this case. There is clearly no largest such language, since both $\mathcal{EL}$ and DLP abide by the weakened principles whereas their (intractable) union does not.

Even when weakening the principles of DLP like this, Horn-$\mathcal{SHIQ}$ is still excluded which cannot be modular (DLP 5) by Proposition 1. In the presence of transitivity, Horn-$\mathcal{SHIQ}$ also is not strictly structural (DLP 6), but this problem could be overcome by using distinct signature sets for simple and non-simple roles. Again, it is open which results can be established for Horn-$\mathcal{SHIQ}$-like DLs based on the remaining weakened principles.

This work also explicitly introduces a notion of semantic emulation which appears to be novel, though loosely related to conservative extensions. In essence, it requires that a theory can take the place of another theory in all logical contexts, based on a given syntactic interface. Examples given in this paper illustrate that emulation can be very different from semantic equivalence. Yet, our criteria can be argued to define minimal requirements for preserving a theory’s semantics even in combination with additional information, so emulation appears to be a natural tool for enabling information exchange in distributed knowledge systems. We expect that the explicit articulation of this notion will be useful for studying the semantic interplay of heterogeneous logical formalisms in general.

Finally, the approach of this paper – seeking a structural logical fragment that is provably maximal under certain conditions – immediately leads to a number of further research questions. For example, what is the maximal fragment of SWRL (“datalog $\cup \mathcal{SROIQ}$”) that can be expressed in $\mathcal{SROIQ}$? Clearly, this fragment would contain DL Rules [12] and maybe some form of DL-safe rules [13]. But also the maximal $\mathcal{FOL}_{\omega}$ fragment that can be expressed in a well-known subset such as the guarded fragment or the two-variable fragment might be of general interest. We argue that ultimate answers to such questions can indeed be obtained by a careful articulation of basic design principles. At the same time, our study indicates that the required definitions and arguments...
can become surprisingly complex when dealing with a syntactically rich formalism like description logic. The main reasons for this is that constructs that are usually considered “syntactic sugar” have non-trivial semantic effects when considering logical fragments that are structurally closed.

Acknowledgements Research reported herein was supported by the EU in the IST project ACTIVE (IST-2007-215040), and by the German Research Foundation under the ReaSem project.
References

Changelog

Version 1.0 (Apr 24 2009) Initial release.

Version 1.1 (May 8 2009) Update in Section 3: The principle DLP 3’ (transformation tractability) is no longer required. Proposition 2 was strengthened by no longer requiring DLP 3’ or ExpTime≠NEExpTime. All uses of DLP 3’ have been removed from the paper.

Version 2.0 (Sept 11 2009) Major update of all sections. Updated grammar for DLP. Complete proofs for datalog emulation and maximality of DLP. Essentially, all of Section 5, 6, and 7 is new.

Version 2.1 (Sept 13 2009) New section presenting the DLP fragment of ALC with complete proofs, intended as a light-weight introduction to the whole approach.