Chasing Sets: How to Use Existential Rules for Expressive Reasoning

David Carral, **Irina Dragoste**, **Markus Krötzsch** and **Christian Lewe**
Knowledge-Based Systems Group, TU Dresden, Dresden, Germany
{david.carral, irina.dragoste, markus.kroetzsch, christian.lewe}@tu-dresden.de

Abstract

We propose that modern existential rule reasoners can enable fully declarative implementations of rule-based inference methods in knowledge representation, in the sense that a particular calculus is captured by a fixed set of rules that can be evaluated on varying inputs (encoded as facts). We introduce Datalog(S) – Datalog with support for sets – as a surface language for such translations, and show that it can be captured in a decidable fragment of existential rules. We then implement several known inference methods in Datalog(S), and empirically show that an existing existential rule reasoner can thus be used to solve practical reasoning problems.

1 Introduction

Rules of inference are of fundamental importance in logical reasoning, and the central building block of many proof systems, including tableaux-based model constructions, resolution calculi, type-theoretic procedures [Ortiz *et al.*, 2010], and "consequence-driven" approaches in ontological reasoning [Kazakov, 2009]. Indeed, rule-based calculi of logical deduction are as old as formal logic itself.

More recently, rules have also seen much renewed interest as a declarative computing paradigm, and many rule-based reasoning systems have been presented [Benedikt *et al.*, 2014; Geerts *et al.*, 2014; Aref *et al.*, 2015; Baget *et al.*, 2015; Nenov *et al.*, 2015; Urbani *et al.*, 2016]. At the core of these implementations is the simple rule language *Datalog*, which is often extended with support for existential quantifiers or function terms in the consequences of rules. While this makes reasoning undecidable in general, there are many fast and scalable reasoners for cases where a finite model can be constructed using some variant of the *chase* procedure [Benedikt *et al.*, 2017; Urbani *et al.*, 2018]. It seems natural to exploit such systems to solve reasoning tasks in other areas of logic.

This can be achieved by translating theories of a relevant logic into sets of rules that entail equivalent consequences. This idea has been applied, e.g., to description logics [Ortiz *et al.*, 2010] and guarded logics [Ahmetaj *et al.*, 2018]. Many such approaches do not play to the strengths of modern rule engines, though, since they create exponentially many (i.e., millions of) rules [Carral *et al.*, 2018] or rules with linearly many (i.e., thousands of) variables [Ahmetaj *et al.*, 2018]. A more promising approach might be to "implement" reasoning algorithms in a fixed (small) set of logical rules that operates on instances of logical reasoning tasks encoded as (large) sets of input facts. This is closer to the typical presentation of rule-based deduction calculi, and was already proposed for rule-based implementations of some logics [Krötzsch, 2011].

Unfortunately, this approach is severely limited by the low *data complexity* of known rule languages, which is PTIME not just for Datalog, but for every previously proposed decidable existential rule language that guarantees the chase to be finite [Krötzsch *et al.*, 2019]. All of the aforementioned tools can only be used if the chase is finite;[1] otherwise reasoning will not terminate. We seem to be faced with a big dilemma: either to deal with large rule sets (in terms of rule number or size), or to be content with solving polynomial problems.

Surprisingly, however, the limits of decidable criteria that guarantee chase termination do not apply to the chase algorithm as such: Krötzsch et al. recently showed that, in theory, even current rule engines can perform complex computations based on a fixed "program" of rules while still being guaranteed to terminate with a finite chase [2019].

In this paper, we show that this theoretical insight can be turned into a practical approach for solving a wide range of logical reasoning tasks with existing rule engines. To unlock the necessary expressive power while guaranteeing termination, we need rather complex existential rule sets that would be difficult to adapt for implementing further reasoning tasks. We overcome this problem by proposing a more convenient intermediate language, Datalog(S), which extends Datalog with relationships over *sets of constants* (Section 3). This is inspired by the Datalog^S language of Ortiz *et al.* [2010], but differs in that the available sets are not fixed as part of the schema, but can grow with the size of the input: Datalog(S) achieves EXPTIME-complete data complexity. We show that every Datalog(S) program can be translated polynomially into a set of existential rules for which a previously proposed (and implemented) variant of the standard chase algorithm terminates in exponential time (Section 4). Notably, the translated rule set does not fall into any known frag-

[1] Even though weakly (frontier) guarded rules are a decidable existential rule fragment with EXPTIME-complete data complexity [Gottlob *et al.*, 2014], the chase does not terminate for this language, and no reasoners have been implemented for this fragment.

ment that guarantees chase termination, and we believe that it would be difficult to design an appropriate new fragment that is nearly as easy to use as Datalog(S).

We illustrate the capabilities of this new approach by implementing two previously proposed reasoning algorithms in Datalog(S): a consequence-based algorithm for description logics (Section 5), and a type-based algorithm for guarded Horn logic (Section 6). Moreover, we demonstrate practical applicability for the former by executing it on an existing rule engine to classify several real-world ontologies (Section 7). Details we had to omit are found online [Carral *et al.*, 2019].

2 Preliminaries

We consider a signature based on mutually disjoint, countably infinite sets of *constants* $\mathbf{C}$, *variables* $\mathbf{V}$, *nulls* $\mathbf{N}$, and *predicates* $\mathbf{P}$. We assign some *arity* $ar(p) \geq 0$ to all $p \in \mathbf{P}$. A *term* is an element in $\mathbf{T} = \mathbf{C} \cup \mathbf{V} \cup \mathbf{N}$. We abbreviate lists of terms $t_1, \ldots, t_n$ as $\vec{t}$, and treat such lists as sets. An *atom* is a formula $p(\vec{t})$ with $p \in \mathbf{P}$, $\vec{t} \subseteq \mathbf{T}$, and $ar(p) = |\vec{t}|$.

We write $\varphi[\vec{x}]$ to indicate that $\vec{x}$ is the set of all free variables in the formula φ. An *(existential) rule* is a null-free formula of the form $\forall \vec{x}, \vec{z}.\beta[\vec{x}, \vec{z}] \rightarrow \exists \vec{y}.\eta[\vec{x}, \vec{y}]$ where $\vec{x}$ and $\vec{y}$ are disjoint lists of variables, β (the *body*) and η (the *head*) are conjunctions of atoms, and η contains at least one atom. Often, we omit universal quantifiers from rules. A *fact* is a variable-free rule with an empty body and a single atom in the head. A rule is *generating* if it contains existential variables, and *non-generating* (or *Datalog*) otherwise.

For a formula φ and a *substitution* $\sigma : \mathbf{V} \rightarrow \mathbf{C} \cup \mathbf{N}$, let $\varphi\sigma$ be the expression obtained by replacing all unbound occurrences of all $x \in \mathbf{V}$ in φ by $\sigma(x)$ if σ is defined for x.

Definition 1. *Consider a rule $\rho = \beta[\vec{x}, \vec{z}] \rightarrow \exists \vec{y}.\eta$, an atom set $\mathcal{A}$, and a substitution σ. The rule and substitution $\langle \rho, \sigma \rangle$ is applicable to $\mathcal{A}$ if (i) the domain of σ is $\vec{x} \cup \vec{z}$, (ii) $\beta\sigma \subseteq \mathcal{A}$, and (iii) $\eta\sigma' \not\subseteq \mathcal{A}$ for all $\sigma' \supseteq \sigma$. Then, $\rho(\mathcal{A})$ is the superset of $\mathcal{A}$ which, for all tuples $\langle \rho, \sigma \rangle$ applicable to $\mathcal{A}$, contains the facts in $\eta\sigma' \subseteq \rho(\mathcal{A})$ with $\sigma' \supseteq \sigma$ a substitution mapping each $y \in \vec{y}$ to a fresh null. For a rule set $\mathcal{R}$ and an atom set $\mathcal{A}$, let $\mathcal{R}(\mathcal{A}) = \bigcup_{\rho \in \mathcal{R}} \rho(\mathcal{A})$. Let $\mathcal{R}_\forall$ and $\mathcal{R}_\exists$ be the sets of all non-generating and generating rules in $\mathcal{R}$, respectively; and let $\mathcal{R}^*_\forall(\mathcal{A})$ be the superset of $\mathcal{A}$ with $\mathcal{R}_\forall(\mathcal{R}^*_\forall(\mathcal{A})) = \mathcal{R}^*_\forall(\mathcal{A})$.*

Definition 2 (Datalog-First Chase)**.** *For a rule set $\mathcal{R}$, let $\mathcal{R}_0 = \emptyset$, $\mathcal{R}_1, \ldots$ be the sequence with $\mathcal{R}_i = \mathcal{R}_\exists(\mathcal{R}^*_\forall(\mathcal{R}_{i-1}))$ for all $i \geq 1$. The* chase *of $\mathcal{R}$ is the set $\mathsf{chase}(\mathcal{R}) = \bigcup_{i \geq 1} \mathcal{R}_i$. The chase of $\mathcal{R}$* terminates *if $\mathcal{R}_{k-1} = \mathcal{R}_k$ for some $k \geq 1$.*

Note that the Datalog-first chase is not a radically different chase method but simply a specific type of rule-application strategy for the *standard chase* (also referred to as *restricted*).

Fact 1. *A fact ϕ is entailed by a rule set $\mathcal{R}$ iff $\phi \in \mathsf{chase}(\mathcal{R})$.*

This follows from the fact that the standard chase produces a *universal model* under all strategies [Deutsch *et al.*, 2008].

3 Datalog with Sets

We now introduce the syntax and semantics of a language that extends Datalog with a set datatype that can represent collections of (non-set) elements. It is defined as a sorted logic with built-in set-related functions and predicates of the expected semantics. This approach is close in spirit to the extension of Datalog with *complex values* [Abiteboul *et al.*, 1994] for the special case with only two sorts (called **dom** and {**dom**} by Abiteboul et al.), no negation, and a restricted choice of complex value operators.

We consider two *sorts*: a sort of *objects* $\mathbf{obj}$ and a sort of *sets* $\mathbf{set}$. A signature of Datalog(S) is based on countably infinite sets of *object constants* $\mathbf{C_{obj}}$, *object variables* $\mathbf{V_{obj}}$, *set variables* $\mathbf{V_{set}}$, and *predicate names* $\mathbf{P}$ including *special predicates* $\{\in, \subseteq\} \subseteq \mathbf{P}$. The *signature* of a predicate symbol p is a tuple $sig(p) = \langle \mathbf{S_1}, \ldots, \mathbf{S_n} \rangle$ of sorts, with $sig(\in) = \langle \mathbf{obj}, \mathbf{set} \rangle$ and $sig(\subseteq) = \langle \mathbf{set}, \mathbf{set} \rangle$. An *object term* is any object constant or object variable. A *set term* is any set variable, the special constant $\emptyset$, an expression $\{t\}$ where t is an object term, or, recursively, an expression $(T_1 \cup T_2)$ where T_1, T_2 are set terms. We consider $\{t_1, \ldots, t_n\}$ an abbreviation for $(\{t_1\} \cup (\{t_2\} \cup \ldots \cup \{t_n\} \ldots))$ and omit parentheses for $\cup$. Unless otherwise stated, we use lower-case letters for object terms and upper-case letters for set terms.

An *atom* is an expression $p(t_1, \ldots, t_n)$ where $p \in \mathbf{P}$ with $sig(p) = \langle \mathbf{S_1}, \ldots, \mathbf{S_n} \rangle$ and t_i is a term of sort $\mathbf{S_i}$. We write $\in(t, S)$ as $t \in S$, and $\subseteq(S_1, S_2)$ as $S_1 \subseteq S_2$. A *Datalog(S) rule* is an expression of the form $B_1 \wedge \ldots \wedge B_\ell \rightarrow H_1 \wedge \ldots \wedge H_k$, with atoms $B_1, \ldots, B_\ell$ (the *body*) and $H_1, \ldots, H_k$ (the *head*), and where we require that:

- every object variable in the head occurs in a body atom,
- every set variable in the rule occurs in a body atom that uses some non-special predicate $p \in \mathbf{P} \setminus \{\in, \subseteq\}$, and
- the head does not use a special predicate.

A *Datalog(S) program* is a set of Datalog(S) rules. We admit rules with empty body for representing Datalog(S) *facts*.

Example 1. *Before defining Datalog(S) semantics formally, we give an example that can be understood with an intuitive reading of Datalog(S) rules. Consider input facts forming a chain $succ(1, 2), succ(2, 3), \ldots, succ(\ell - 1, \ell)$, starting at the constant 1 (the other constant names are immaterial). The following program entails facts for binary predicate n that form an n-chain of length 2^ℓ:*

$$full(\emptyset) \qquad n(\emptyset, \{1\}) \qquad parts(\{1\}, 1, \emptyset) \tag{1}$$

$$n(U, V) \wedge parts(V, x, V') \wedge n(V', W') \rightarrow n(V, \{x\} \cup W') \wedge parts(\{x\} \cup W', x, W') \tag{2}$$

$$n(U, V) \wedge parts(V, x, V') \wedge full(V') \wedge succ(x, y) \rightarrow n(V, \{y\}) \wedge parts(\{y\}, y, \emptyset) \wedge full(V) \tag{3}$$

Elements of the n-chain are represented by sets, ordered as if each input constant in the $succ$-chain defines one bit of a binary number. $full(S)$ means that the represented number is of the form $0 \cdots 01 \cdots 1$; $parts(S, x, S')$ means that the number S has its most significant bit at position x from the left, and that $S = \{x\} \cup S'$. The three facts (1) *start the chain, and the two other rules define the next n-successor for when the most significant bit stays the same* (2)*, or moves to the left* (3)*, respectively.*

Using an exponentially long chain structure, it is easy to simulate exponential time deterministic Turing machine computations in Datalog [Dantsin *et al.*, 2001]. Based on Example 1, we can apply a similar construction to Datalog(S) to obtain the following result, which should be contrasted to the polynomial data complexity of regular Datalog:

Lemma 1. *There is a fixed Datalog(S) program without special predicates $\in$ and $\subseteq$ for which fact entailment checking is* EXPTIME*-hard with respect to the size of the input data.*

The formal semantics of Datalog(S) codifies the intuitive idea of the set functions and relations. An *interpretation* $\mathcal{I}$ of Datalog(S) is defined by an *object domain* $\mathbf{obj}^{\mathcal{I}}$ and an interpretation function $\cdot^{\mathcal{I}}$. The *set domain* $\mathbf{set}^{\mathcal{I}}$ of $\mathcal{I}$ is the powerset of $\mathbf{obj}^{\mathcal{I}}$. The interpretation maps non-special predicates p of signature $sig(p) = \langle \mathbf{S_1}, \ldots, \mathbf{S_n} \rangle$ to relations $p^{\mathcal{I}} \subseteq \mathbf{S}_\mathbf{1}^{\mathcal{I}} \times \ldots \times \mathbf{S}_\mathbf{n}^{\mathcal{I}}$, and special predicates as expected: $\in^{\mathcal{I}} = \{\langle \delta, \Gamma \rangle \mid \delta \in \Gamma \in \mathbf{set}^{\mathcal{I}}\}$ and $\subseteq^{\mathcal{I}} = \{\langle \Gamma_1, \Gamma_2 \rangle \mid \Gamma_1 \subseteq \Gamma_2 \in \mathbf{set}^{\mathcal{I}}\}$. Object constants c are mapped to elements $c^{\mathcal{I}} \in \mathbf{obj}^{\mathcal{I}}$. A *variable assignment* $\mathcal{Z}$ is a function mapping object variables x to elements $\mathcal{Z}(x) \in \mathbf{obj}^{\mathcal{I}}$, and set variables Y to sets $\mathcal{Z}(Y) \in \mathbf{set}^{\mathcal{I}}$. Given an arbitrary (set or object) term t, we define $t^{\mathcal{I},\mathcal{Z}}$ recursively as follows: $c^{\mathcal{I},\mathcal{Z}} = c^{\mathcal{I}}$ for $c \in \mathbf{C_{obj}}$, $x^{\mathcal{I},\mathcal{Z}} = \mathcal{Z}(x)$ for $x \in \mathbf{V_{obj}} \cup \mathbf{V_{set}}$, $\emptyset^{\mathcal{I},\mathcal{Z}} = \emptyset$, $\{s\}^{\mathcal{I},\mathcal{Z}} = \{s^{\mathcal{I},\mathcal{Z}}\}$, and $(S_1 \cup S_2)^{\mathcal{I},\mathcal{Z}} = S_1^{\mathcal{I},\mathcal{Z}} \cup S_2^{\mathcal{I},\mathcal{Z}}$.

An interpretation $\mathcal{I}$ and variable assignment $\mathcal{Z}$ *satisfy* an atom $p(t_1, \ldots, t_n)$, written $\mathcal{I}, \mathcal{Z} \models p(t_1, \ldots, t_n)$, if $\langle t_1^{\mathcal{I},\mathcal{Z}}, \ldots, t_n^{\mathcal{I},\mathcal{Z}} \rangle \in p^{\mathcal{I}}$. $\mathcal{I}$ satisfies a Datalog(S) rule $B_1 \wedge \ldots \wedge B_\ell \to H_1 \wedge \ldots \wedge H_k$ if, for all variable assignments $\mathcal{Z}$ for $\mathcal{I}$ such that $\mathcal{I}, \mathcal{Z} \models B_i$ for all $1 \leq i \leq \ell$, we also have $\mathcal{I}, \mathcal{Z} \models H_j$ for all $1 \leq j \leq k$. A program is satisfied if all of its rules are. A variable-free atom A is a *logical consequence* of a program P if $\mathcal{I} \models A$ for all $\mathcal{I}$ with $\mathcal{I} \models P$.

Since we require object and set variables to occur in (non-special) body predicates, it is easy to see that rules are only applicable to variable assignments that use values that correspond to (sets of) constants in the given input facts. A Datalog(S) program is therefore equivalent to its finite *grounding*, obtained by replacing variables by variable-free terms that represent all possible constants and sets of constants, respectively. This grounding results in an exponentially large propositional Horn theory for which reasoning tasks can be solved polynomially. Together with Lemma 1, we obtain:

Theorem 1. *Checking fact entailment for Datalog(S) is* EXPTIME*-complete for both data and combined complexity.*

4 From Datalog(S) to Existential Rules

We now show how to polynomially translate Datalog(S) into existential rules while preserving (suitably translated) consequences. The heart of our approach is the following set $\mathcal{R}_{\mathsf{SU}}$ of existential rules (4)–(6), which can represent sets by nulls:

$$\to \exists v.\, empty(v) \wedge set(v) \tag{4}$$

$$getSU(x,u) \to \exists v.\, SU(x,u,v) \wedge SU(x,v,v) \wedge set(v) \tag{5}$$

$$SU(x,u,v) \wedge SU(y,u,u) \to SU(y,v,v) \tag{6}$$

Intuitively, $set(s)$ means "s is a set" and $empty(s)$ means "$s = \emptyset$." *SU* is for *Singleton Union*: $SU(a,s,t)$ means "$\{a\} \cup s = t$," and $getSU(a,s)$ means "$\{a\} \cup s$ should be computed." Note that all terms are object terms in this one-sorted logic. Given an atom set $\mathcal{A}$ and a constant or null $s \in \mathbf{C} \cup \mathbf{N}$, we therefore define the *corresponding set* $[s]_{\mathcal{A}} = \{a \in \mathbf{C} \cup \mathbf{N} \mid SU(a,s,s) \in \mathcal{A}\}$. We will combine $\mathcal{R}_{\mathsf{SU}}$ with additional non-generating rules, but some restrictions are needed to guarantee correctness.

Definition 3. *A* type assignment α *is a function that maps each predicate $p \in \mathbf{P}$ to a tuple $\alpha(p) = \langle \mathbf{S_1}, \ldots, \mathbf{S_n} \rangle$ with $n = ar(p)$ and $\mathbf{S_i} \in \{\mathbf{obj}, \mathbf{set}\}$ for $1 \leq i \leq n$, and where we require that $\alpha(empty) = \langle \mathbf{set} \rangle$, $\alpha(SU) = \langle \mathbf{obj}, \mathbf{set}, \mathbf{set} \rangle$, and $\alpha(getSU) = \langle \mathbf{obj}, \mathbf{set} \rangle$. We say position i in p is* of type $\mathbf{S_i}$*. A non-generating rule ρ is* admissible *if there is a type assignment such that:*

(i) constants in ρ only occur on positions of type $\mathbf{obj}$*,*

(ii) variables in ρ only occur on positions of a single type,

(iii) $empty$ and SU do not occur in the head of ρ, and

(iv) $getSU$ does not occur in the body of ρ.

A set of rules (or facts) is admissible if all of its elements are.

Example 2. *$\mathcal{R}_{\mathsf{SU}}$ already suffices to simulate some Datalog(S) programs using further admissible rules. For example, rule* (2) *can be expressed with the following admissible rules, where the type assignment is as defined by the signature of the corresponding Datalog(S) predicates:*

$$n(u,v) \wedge parts(v,x,v') \wedge n(v',w') \to getSU(x,w') \tag{7}$$

$$n(u,v) \wedge parts(v,x,v') \wedge n(v',w') \wedge SU(x,w',w) \to n(v,w) \wedge parts(w,x,w') \tag{8}$$

The original rule is split into two: (7) *requests the creation of an element to represent the set "$\{x\} \cup w'$" by rule* (5)*, and* (8) *uses this new element to instantiate the original head atoms.*

Example 2 illustrates how admissible rules can "call" the rules in $\mathcal{R}_{\mathsf{SU}}$ to provision "sets" as required. The following main correctness result confirms that this works as expected:

Theorem 2. *For every admissible set of non-generating rules $\mathcal{R}$, the chase $\mathcal{C} = \mathsf{chase}(\mathcal{R} \cup \mathcal{R}_{\mathsf{SU}})$ is such that*

(a) $empty(s) \in \mathcal{C}$ implies $[s]_{\mathcal{C}} = \emptyset$, and

(b) $SU(a,s,t) \in \mathcal{C}$ implies $\{a\} \cup [s]_{\mathcal{C}} = [t]_{\mathcal{C}}$.

If n is the size of $\mathcal{R}$, then the size of $\mathcal{C}$ is in $O(2^{n \log n})$.

Proof. It is clear that rule (4) is applied exactly once, irrespective of $\mathcal{R}$, introducing some null $n_\emptyset$ for v. By admissibility, the only rule heads with SU in $\mathcal{R} \cup \mathcal{R}_{\mathsf{SU}}$ are in (5) and (6), and an easy induction shows that $n_\emptyset$ can never occur in such facts. Hence, $[n_\emptyset]_{\mathcal{C}} = \emptyset$, showing claim (a).

For (b), first note that the claim follows from the definition of $[t]_{\mathcal{C}}$ if $s = t$. Facts $SU(a,s,t)$ with $s \neq t$ are only derived by (5), and each null t occurs in at most one such fact. By the same rule, we get $SU(a,t,t)$, showing $a \in [t]_{\mathcal{C}}$. If $s = n_\emptyset$, then rule (6) cannot produce further consequences, and we get $[t]_{\mathcal{C}} = \{a\}$ as required. If $s \neq n_\emptyset$, then we have a fact $SU(b,s,s)$ for each $b \in [s]_{\mathcal{C}}$ by definition of $[s]_{\mathcal{C}}$. Rule (6)

$$getU(v,w) \wedge empty(v) \rightarrow U(v,w,w) \quad (9)$$

$$getU(v,w) \wedge SU(x,v_-,v) \rightarrow getSU(x,w) \quad (10)$$

$$getU(v,w) \wedge SU(x,v_-,v) \wedge SU(x,w,w_+) \rightarrow getU(v_-,w_+) \quad (11)$$

$$getU(v,w) \wedge SU(x,v_-,v) \wedge SU(x,w,w_+) \wedge U(v_-,w_+,u) \rightarrow U(v,w,u) \quad (12)$$

$$SU(x,u,u) \rightarrow in(x,u) \quad (13)$$

$$set(v) \wedge set(w) \rightarrow ckSub(v,v,w) \quad (14)$$

$$ckSub(u,v,w) \wedge SU(x,u_-,u) \wedge in(x,w) \rightarrow ckSub(u_-,v,w) \quad (15)$$

$$ckSub(u,v,w) \wedge empty(u) \rightarrow sub(v,w) \quad (16)$$

Figure 1: Rules for arbitrary unions, membership, and containment (subscripts $+/-$ are part of variable names; they hint at sets with additional/fewer elements)

therefore derives $SU(b,t,t)$, i.e., $b \in [t]_{\mathcal{C}}$. No other facts $SU(b,t,t)$ can be derived, therefore $[t]_{\mathcal{C}} = \{a\} \cup [s]_{\mathcal{C}}$.

For the size of $\mathcal{C}$, note that, all facts of the form $SU(a,s,s) \in \mathcal{C}$ are derived either when introducing s in (5), or immediately afterwards when closing under non-generating rule (6) (which is prioritised in the Datalog-first chase). Therefore, whenever (5) is applied with a substitution σ to introduce a fresh null $n = \sigma(v)$, all facts $SU(b,\sigma(u),\sigma(u))$ for $b \in [\sigma(u)]_{\mathcal{C}}$ were already derived. Since (5) is applicable, $\sigma(x) \notin [\sigma(u)]_{\mathcal{C}}$. Let $\mathbf{C}_{\mathcal{R}}$ be the finite set of constants that occur in $\mathcal{R}$. By admissibility (and a simple induction), $\sigma(x) \in \mathbf{C}_{\mathcal{R}}$, and by claim (b), $[\sigma(u)]_{\mathcal{C}} \subsetneq [n]_{\mathcal{C}} \subseteq 2^{\mathbf{C}_{\mathcal{R}}}$. With $\ell = |\mathbf{C}_{\mathcal{R}}|$, there are at most $\ell! < \ell^\ell \in O(2^{\ell \log \ell})$ possible sequences of applications of rule (5). This bounds the number of terms in the chase, and hence the number of derived facts, as claimed. □

Note that the proof relies on the prioritisation of rule (6) in the Datalog-first chase. However, by adapting a technique by Krötzsch et al. [2019, Proof of Theorem 13], one can also obtain a (fixed) rule set that achieves the same termination and worst-case complexity properties for the standard chase. We omit this here as it makes rules more complicated and presumably less efficient in practice (since it artificially prevents the parallel application of rule (6) to all set elements).

As the proof of Theorem 2 suggests, each set might be represented by several elements in the simulation via $\mathcal{R}_{\mathsf{SU}}$. The elements generated by $\mathcal{R}_{\mathsf{SU}}$ correspond to sequences of unions of mutually distinct singleton sets, and several sequences might result in the same set. This has to be taken into account when simulating $\subseteq$ and when performing joins over set terms. Singleton sets and the empty set have at most one representative.

$\mathcal{R}_{\mathsf{SU}}$ can only simulate the empty set and unions with singletons. Arbitrary unions are supported by the rules (9)–(12) in Figure 1. Unions are constructed recursively by adding single elements from the first to the second input set. Rule (9) defines the base case. The other three rules split off a single element, use $\mathcal{R}_{\mathsf{SU}}$ to construct unions with singletons, and use the output to define the overall union. The rules are admissible for the intuitive type assignment $\alpha(getU) = \langle \mathbf{set}, \mathbf{set} \rangle$ and $\alpha(U) = \langle \mathbf{set}, \mathbf{set}, \mathbf{set} \rangle$.

As the last ingredient, the rules (13)–(16) in Figure 1 define predicates in and sub that capture the built-in Datalog(S) predicates $\in$ and $\subseteq$, respectively. The complicated part is sub: facts of the form $ckSub(u,v,w)$ ("check subset") express "$v \setminus u \subseteq w$." We initialise this for all pairs of sets (14), and recursively compare elements of the first argument (15) until reaching the empty set (16).

This completes the required set of auxiliary rules. Before specifying the intended translation, we slightly *normalise* Datalog(S) rules. Let $S_1 = S_2$ be a shortcut for $S_1 \subseteq S_2 \wedge S_2 \subseteq S_1$. First, we replace every set term of the form $S_1 \cup S_2$ in a non-special predicate in the body with a fresh set variable S, and we add body atoms $S = S_1 \cup S_2$. Second, as long as a set variable S occurs more than once in a non-special predicate in the body of a rule, we replace one of these occurrences by a fresh variable S' and add $S = S'$ to the body. It is easy to see that these transformations preserve semantics. Rules of the resulting form are called *normalised*. Normalisation ensures that set equality checks are performed using sub.

Now consider a normalised Datalog(S) rule ρ. We will translate it to a set $\mathsf{ER}(\rho)$ of existential rules. For every set term S, let $v(S)$ be a fresh (non-sorted) variable. Let $S_1, \dots, S_k$ be a sequence of all set terms in ρ that are neither variables nor $\emptyset$, such that no sub-term of any S_i occurs to its right. For each term S_i, we define atoms α_i and β_i:

- if $S_i = \{t\}$ then $\alpha_i = getSU(t, v(\emptyset))$ and $\beta_i = SU(t, v(\emptyset), v(S_i))$,
- if $S_i = T_1 \cup T_2$, then $\alpha_i = getU(v(T_1), v(T_2))$ and $\beta_i = U(v(T_1), v(T_2), v(S_i))$.

Finally, set $\gamma_0 = empty(v(\emptyset))$, and $\gamma_{j+1} = \gamma_j \wedge \beta_{j+1}$ for all $0 \leq j < k$. Now rule $\rho = \varphi \rightarrow \psi$ is translated into the set $\mathsf{ER}(\rho)$ of the following $k+1$ rules:

(1) for each $0 \leq i < k$, a rule $\varphi' \wedge \gamma_i \rightarrow \alpha_{i+1}$,

(2) a rule $\varphi' \wedge \gamma_k \rightarrow \psi'$,

where φ' and ψ' are obtained from φ and ψ, respectively, by replacing all atoms $t \in S$ by $in(t,S)$, all atoms $S \subseteq S'$ by $sub(S,S')$, and all set terms S by $v(S)$.

Example 3. *Let $\rho = p(a,S,T,\{a\} \cup S) \rightarrow q(S \cup T)$. Its normalisation ρ' has the body $\varphi = p(a,S,T,R) \wedge (R \subseteq \{a\} \cup S) \wedge (\{a\} \cup S \subseteq R)$. A possible order of set terms is $S_1 = \{a\}$, $S_2 = \{a\} \cup S$, $S_3 = S \cup T$. The translated body is $\varphi' = p(a, v(S), v(T), v(R)) \wedge sub(v(R), v(S_2)) \wedge sub(v(S_2), v(R))$. We obtain the following rules in $\mathsf{ER}(\rho')$:*

$$\varphi' \wedge empty(v(\emptyset)) \rightarrow getSU(a, v(\emptyset)) \quad (17)$$

$$\dots \wedge SU(a, v(\emptyset), v(S_1)) \rightarrow getU(v(S_1), v(S)) \quad (18)$$

$$\dots \wedge U(v(\{a\}), v(S), v(S_2)) \rightarrow getU(v(S), v(T)) \quad (19)$$

$$\dots \wedge U(v(S), v(T), v(S_3)) \rightarrow q(v(S_3)) \quad (20)$$

where each "…" abbreviates the body of the previous rule.

Definition 4. *For a Datalog(S) program $\mathcal{P}$, let $\mathsf{ER}(\mathcal{P})$ be the union of all rule sets $\mathsf{ER}(\rho)$ for $\rho \in \mathcal{P}$ and the rules* (4)–(6) *and* (9)–(16).

The following result is immediate from this definition:

Theorem 3. *For a Datalog(S) program $\mathcal{P}$, the size of $\mathsf{ER}(\mathcal{P})$ is polynomial in the size of $\mathcal{P}$.*

Theorem 4. *For a Datalog(S) program $\mathcal{P}$ and Datalog(S) fact α, $\mathcal{P} \models \alpha$ if and only if $\mathsf{ER}(\mathcal{P}) \models \mathsf{ER}(\alpha)$. Moreover, the Datalog-first chase decides $\mathsf{ER}(\mathcal{P}) \models \mathsf{ER}(\alpha)$ in* ExpTime.

Proof sketch. The correctness of the extended set of rules (9)–(16) can be shown with a suitable extension of Theorem 2. We can restrict the notion of admissibility to also avoid the misuse of auxiliary predicates in these rules while requiring the expected type signatures. The correctness of the translation is easy; especially the translated rules are admissible when using the Datalog(S) signatures to define type signatures. The size bound on the chase carries over from Theorem 2, which establishes the second part of the claim. □

Together with Theorem 1, this correctness result implies that fact entailment is also ExpTime-complete w.r.t. data complexity for the sets of rules obtained from our transformation. Therefore, these transformed sets are not covered by any decidable fragment of existential rules of P (or lower) data complexity, including previously proposed criteria that ensure termination of the skolem chase [Cuenca Grau *et al.*, 2013] and Datalog-first chase [Carral *et al.*, 2017], as well as first-order rewritability [Baget *et al.*, 2011]. Weakly (frontier) guarded rules are one of the few fragments with similarly high data complexity [Gottlob *et al.*, 2014], but already the rules in Figure 1 are not weakly frontier guarded.

5 Description Logics Reasoning

To demonstrate the utility of our approach, we look at description logics (DLs) first. DLs are highly expressive logics with many applications, and for which numerous reasoning procedures have been proposed [Baader *et al.*, 2007]. Among the many DLs that give rise to ExpTime-complete reasoning tasks, we select Horn-$\mathcal{ALC}$ as a comparatively simple case that allows us to focus on our main ideas.[2]

DL is based on mutually disjoint sets of *concept names* $\mathbf{N_C}$, *role names* $\mathbf{N_R}$, and *individual names* $\mathbf{N_I}$. We define Horn-$\mathcal{ALC}$ using axioms in normalised form [Krötzsch *et al.*, 2013]. Given $A, B, C \in \mathbf{N_C}$, $R \in \mathbf{N_R}$, and $a, b \in \mathbf{N_I}$, the following are axioms of Horn-$\mathcal{ALC}$ in normal form:

$$A(a) \quad (21) \qquad \top \sqsubseteq C \quad (24) \qquad A \sqsubseteq \bot \quad (27)$$
$$R(a,b) \quad (22) \qquad \exists R.A \sqsubseteq C \quad (25) \qquad A \sqsubseteq \forall R.C \quad (28)$$
$$A \sqsubseteq C \quad (23) \qquad A \sqcap B \sqsubseteq C \quad (26) \qquad A \sqsubseteq \exists R.C \quad (29)$$

Axioms (21) and (22), called *class assertions*, respectively *role assertions*, are *ABox axioms*, corresponding to unary and binary facts. The others are *TBox axioms*, with $D \sqsubseteq E$ representing a first-order formula $\forall x.[D] \rightarrow [E]$, where we define $[A] = A(x)$, $[B] = B(x)$, $[\exists R.C] = \exists y.R(x,y) \wedge C(y)$, $[\forall R.C] = \forall y.R(x,y) \rightarrow C(y)$, and $[A \sqcap B] = A(x) \wedge B(x)$. $\top$ represents truth, and $\bot$ falsity.

[2] Our use of Horn DLs is not related to the fact that existential rules are in Horn logic. Rule-based reasoning methods for non-Horn DLs, e.g., from [Simančík *et al.*, 2011], could also be implemented.

Classification is the task of computing all axioms of form (23) that are entailed by an *ontology*—i.e. a DL axiom set. Kazakov [2009] proposes a *consequence-driven* classification method for Horn-$\mathcal{ALC}$. Figure 2 (left) shows the corresponding rules of inference in the version of Simančík *et al.* [2011, Table 2]. Given a set $\mathcal{T}$ of TBox axioms, the rules produce inferences of the form $H \sqsubseteq B$ and $H \sqsubseteq \exists R.K$ where H, K are conjunctions of elements in $\mathbf{N_C} \cup \{\bot, \top\}$ (viewed as sets when convenient). A rule is applicable if its pre- (above the line) and side conditions (on the right) are satisfied; then it derives the conclusion (below the line). A conjunction H is *active* if it is either defined to be active initially, or occurs in some inference, and rules **($\mathbf{R_A}$)** and **($\mathbf{R_\sqcap}$)** are restricted to such conjunctions. For classification, one initially sets all singleton conjunctions (i.e., concept names) to be active.

The right of Figure 2 shows the corresponding Datalog(S) rules. It is easy to see how each rule corresponds to the inference rule to its left. We represent inferences by predicates SC and Ex with $sig(SC) = \langle \mathbf{set}, \mathbf{obj} \rangle$ and $sig(Ex) = \langle \mathbf{set}, \mathbf{obj}, \mathbf{set} \rangle$, respectively. Predicates $ax_\sqsubseteq$, $ax_{\sqcap\sqsubseteq}$, $ax_{\sqsubseteq\forall}$, $ax_{\exists\sqsubseteq}$, and $ax_{\sqsubseteq\exists}$ encode Horn-$\mathcal{ALC}$ axioms as facts. All elements of $\mathbf{N_C} \cup \mathbf{N_R}$ correspond to object constants in Datalog(S), and conjunctions are represented as sets. Only rules **($\mathbf{R^+_\exists}$)** and **($\mathbf{R_\forall}$)** can activate new conjunctions, as recorded by predicate Act. To initialise the computation, we need to activate all singleton sets, which can be done by rules such as

$$ax_{\exists\sqsubseteq}(a, r, b) \rightarrow Act(\{a\}) \wedge Act(\{b\}) \qquad (30)$$

and analogous rules for all other types of axioms. It is obvious that this translation is faithful and therefore leads to an ExpTime-complete (hence worst-case optimal) chase-based classification algorithm for Horn-$\mathcal{ALC}$.

We also note that the translation exhibits similar pay-as-you-go characteristics as the original approach. In particular, in $\mathcal{EL}$-type description logics, which do not include axioms of type (28), only a linear number of singleton sets is needed, and the chase likewise terminates in polynomial time. It is an advantage of our approach that it can be used to make an existential rule engine perform essentially the same derivation steps as the original abstract inference-rule calculus.

6 Guarded Rules Reasoning

We now apply our approach to reasoning with *guarded existential rules* (where all universally quantified variables appear in a single body atom). Ahmetaj *et al.* [2018] give an entailment-preserving translation from guarded rules with bounded predicate arity $\hat{a}$ to a polynomial Datalog program. However, they require predicate arities in the order of $2^{\hat{a}}$.

Their translation actually simulates sets in Datalog. For maximal arity $\hat{a}$, we consider the set $\mathcal{A}$ of all atoms that can be formed from a predicate in the guarded rules and generic variables $x_1, \ldots, x_{\hat{a}}$. A *type* is a subset of $\mathcal{A}$, represented in Datalog by vectors of $|\mathcal{A}|$ constants 0 or 1. Datalog(S) rules are obtained, in essence, by replacing these vectors by sets, where we use constants c_α for each $\alpha \in \mathcal{A}$.

The rules frequently test for the absence of an element in a set. While not in Datalog(S), we can define this using an *inequality predicate* $\not\approx$ (even without logical inequality, this

($\mathbf{R_A}$)	$\frac{}{H \sqsubseteq A}$: H is active and $A \in H$	$Act(H) \wedge a \in H \rightarrow SC(H, a)$
($\mathbf{R_\sqcap}$)	$\frac{\{H \sqsubseteq A_i\}_{i=1}^n}{H \sqsubseteq C}$: $n = 0$, H active, $\top \sqsubseteq C \in \mathcal{T}$, or $n = 1$, $A_1 \sqsubseteq C \in \mathcal{T}$, or $n = 2$, $A_1 \sqcap A_2 \sqsubseteq C \in \mathcal{T}$	$Act(H) \wedge ax_\sqsubseteq(c_\top, c) \rightarrow SC(H, c)$ $SC(H, a_1) \wedge ax_\sqsubseteq(a_1, c) \rightarrow SC(H, c)$ $SC(H, a_1) \wedge SC(H, a_2) \wedge ax_{\sqcap\sqsubseteq}(a_1, a_2, c) \rightarrow SC(H, c)$
($\mathbf{R}^+_\exists$)	$\frac{H \sqsubseteq A}{H \sqsubseteq \exists R.B} : A \sqsubseteq \exists R.B \in \mathcal{T}$	$SC(H, a) \wedge ax_{\sqsubseteq\exists}(a, r, b) \rightarrow Ex(H, r, \{b\}) \wedge Act(\{b\})$
($\mathbf{R}^-_\exists$)	$\frac{H \sqsubseteq \exists R.K \quad K \sqsubseteq A}{H \sqsubseteq B} : \exists R.A \sqsubseteq B \in \mathcal{T}$	$Ex(H, r, K) \wedge SC(K, a) \wedge ax_{\exists\sqsubseteq}(a, r, b) \rightarrow SC(H, b)$
($\mathbf{R}^\bot_\exists$)	$\frac{H \sqsubseteq \exists R.K \quad K \sqsubseteq \bot}{H \sqsubseteq \bot}$	$Ex(H, r, K) \wedge SC(K, c_\bot) \rightarrow SC(H, c_\bot)$
($\mathbf{R}_\forall$)	$\frac{H \sqsubseteq \exists R.K \quad H \sqsubseteq A}{H \sqsubseteq \exists R.(K \sqcap B)} : A \sqsubseteq \forall R.B \in \mathcal{T}$	$Ex(H, r, K) \wedge SC(H, a) \wedge ax_{\sqsubseteq\forall}(a, r, b) \rightarrow Ex(H, r, \{b\} \cup K) \wedge Act(\{b\} \cup K)$

Figure 2: Horn-$\mathcal{ALC}$ inference rules by Simančík et al. (left) and corresponding Datalog(S) program (right), where $c_\top$ and $c_\bot$ are constants, lower-case letters are object variables, and upper-case letters are set variables

can be axiomatised for constants; this suffices). We can now derive $ckNotin(u, a, u)$ whenever a rule needs to know if "$a \notin u$", and then apply the following rules:

$$\begin{aligned} &ckNotin(u, z, w) \wedge \\ &SU(x, u_-, u) \wedge x \not\approx z \rightarrow ckNotin(u_-, z, w) \end{aligned} \tag{31}$$

$$ckNotin(u, z, w) \wedge empty(u) \rightarrow notin(z, w) \tag{32}$$

Confirmed non-memberships are recorded as $notin(a, u)$. The approach resembles our computation of sub in Figure 1.

With this additional feature, most rules of Ahmetaj *et al.* have a straightforward translation to Datalog(S). Their rules are organised in groups **(I)** to **(IX)**, where **(II)** and **(VI)** only matter in the disjunctive case. Even the remaining rules are too many to specify, but we explain the key steps. Rules **(I)** initialise non-set predicates of arity $\leq \hat{a}$, and require no modification. The rules in **(III)** define all types (vectors) and establish a linear order on them. This can be achieved in Datalog(S) along the lines of Example 1. Rules **(IV)** mark types based on the given guarded rules. Using a simple decomposition of rule bodies, we can assume that all guarded rules have the form $\rho = A \wedge B \rightarrow \exists \vec{y}.H$. Then, for each mapping h from rule variables to $\{x_1, \ldots, x_{\hat{a}}\}$, we define a rule:

$$Type(U) \wedge c_{h(A)} \in U \wedge c_{h(B)} \in U \wedge c_{h(H)} \notin U \rightarrow Marked(U)$$

If we assume that rules restrict to a common set of variables, there are only polynomially many possible mappings h due to the bounded arity. Therefore, instead of adding many rules, we can also encode those mappings and the rule ρ in Datalog(S) facts, and use a single Datalog(S) rule instead.

The rules **(V)** are again very close to Datalog(S). They use predicates such as $MarkedOne_{\sigma,h}$, named using rules σ and certain mappings h. As in **(IV)**, we can remove the dependence on these input-related parts by representing rules and mappings as elements in the facts, and adding two additional parameters to $MarkedOne$ to refer to them. Note that side conditions such as "$R(\vec{x}) \in \mathcal{A}$ with $\vec{x} \in h(\text{vars}(\sigma))$" can simply be precomputed and stored as facts.

Rules **(VII)** highlight some elements in types using a special constant 2 instead of 1, as in

$$Marked(\vec{u}, 0, \vec{v}) \wedge Marked(\vec{u}, 1, \vec{v}) \rightarrow Marked(\vec{u}, 2, \vec{v})$$

We achieve this *adding* another constant d_α for each $\alpha \in \mathcal{A}$:

$$\begin{aligned} Marked(X) \wedge c_{\alpha_i} \notin X \wedge Marked(X \cup \{c_{\alpha_i}\}) \\ \rightarrow Marked(X \cup (\{c_{\alpha_i}\} \cup \{d_{\alpha_i}\})) \end{aligned}$$

Such rules also depend on the given guarded rules that define $\mathcal{A}$, but this dependence can easily be moved to the facts by encoding $\mathcal{A}$ there. Rules **(VIII)** define *Horn types* as those missing exactly one atom. We can build such types iteratively, so as to avoid the explicit mentioning of all elements in one rule. The remaining rules are also easy to express. Finally, rules **(IX)** infer entailed facts from marked Horn types by removing elements from the type sets iteratively. Instead of removing elements, we can simply iterate by remembering the current position in an additional parameter.

By these transformations, we can obtain a Datalog(S) program that captures the original Datalog program in a polynomial number of rules that each use only a bounded number of variables and atoms (based on the arity $\hat{a}$). If we further encode guarded rules, atoms, and mappings in Datalog(S) facts as indicated, we obtain a fixed Datalog(S) program that can reason on arbitrary guarded existential rules (encoded as facts), depending only on the signature of relevant predicates. Applying Theorem 4, we obtain a similarly small existential rule set for which the Datalog-first chase terminates:

Theorem 5. *For any fixed finite set of predicates $\mathbf{P}$ with maximal arity $\hat{a}$, there is a set of existential rules $\mathcal{AOS}_\mathbf{P}$ that*

1. *consists of polynomially many rules (in the size of $\mathbf{P}$),*
2. *each with bounded number of atoms (independent of $\mathbf{P}$),*
3. *using predicates of arity bounded by $\max(\hat{a}, 3)$, and*
4. *where $\mathsf{chase}(\mathcal{AOS}_\mathbf{P} \cup \mathcal{F})$ is exponential for every set of facts $\mathcal{F}$, such that*

every set of guarded existential rules $\mathcal{R}$ over $\mathbf{P}$ can be translated into a polynomial set of facts $\mathcal{F}_\mathcal{R}$ such that $\mathcal{AOS}_\mathbf{P} \cup \mathcal{F}_\mathcal{R}$ and $\mathcal{R}$ entail the same ground facts over $\mathbf{P}$.

ID	#Ax.	#Set	#SC	#Ex	VLog (sec)
00040	223K	2K	1051K	334K	432
00048	142K	19	718K	171K	387
00477	318K	0	162K	167K	1
00533	159K	0	965K	351K	132
00786	152K	12K	2283K	978K	549

Figure 3: Ontologies and results for classification (A) showing: axiom count; number of non-singleton "set terms" introduced (#Set); number of SC and Ex facts derived; reasoning time in VLog

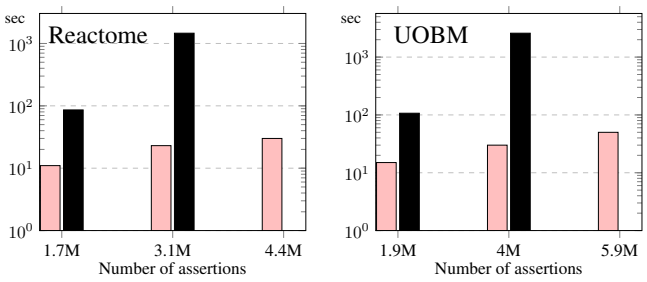

Figure 4: Experimental results for class retrieval (B) in VLog (pink/grey) and Konclude (black); note the log scale

7 Evaluation

We conducted two experiments: (A) compute all consequences of the calculus in Figure 2, and (B) compute all entailed axioms of type (21) (class retrieval). All experiments where run on a MacBookPro (2.4GHz Intel Core i5, 8GB RAM). The rule engine VLog was used as an implementation for the Datalog-first chase [Urbani *et al.*, 2018]. Further evaluation details can be found in [Carral *et al.*, 2019].

For (A), we created a simplified rule set $\mathcal{R}_{Cl}$ from the rules in Figure 2, exploiting the fact that they do not need normalisation or all Datalog(S) features [Carral *et al.*, 2019]. We classified five large and diverse ontologies from the Oxford Ontology Repository:[3] 00040 (GO x-anatomy), 00048 (GO x-taxon), 00477 (Gazetteer), 00533 (ChEBI mol. function), and 00786 (NCI). The ontology statistics and results are shown in Figure 3. We measured pure reasoning time without the time needed to read the input file from disk.

Columns #Set, #SC, and #Ex confirm that the ontologies require significant reasoning effort, which is also reflected in most VLog reasoning times. We observed that our rules for simulating sets made only small contributions to these times, although they dominate theoretical worst-case complexity. Most time instead was taken up by the computation of a small number of particularly slow joins – used to find rule matches in VLog –, especially for rule ($\mathbf{R}_{\exists}^{-}$).

While our results show practicality, optimised DL reasoners are significantly faster: the DL reasoner Konclude [Steigmiller *et al.*, 2014] needed between 2sec (00533) and 14sec (00786) for the tasks. This is not surprising, since Konclude is the leading implementation after decades of research on how to optimise this very computation, whereas VLog has never before been applied for such a task. Indeed, our approach leads to rules that are very different from the ones used in benchmarking rule engines today, and can therefore guide further optimisations of existing implementations.

[3] https://www.cs.ox.ac.uk/isg/ontologies/

For experiment (B) we extend $\mathcal{R}_{Cl}$ with the following rules:

$$CA(a,c) \wedge SU(a,E,H) \wedge empty(E) \wedge SC(H,b) \rightarrow CA(b,c) \quad (33)$$

$$RA(r,c,d) \wedge ax_{\exists\sqsubseteq}(a,r,b) \wedge CA(a,d) \rightarrow CA(b,c) \quad (34)$$

$$RA(r,c,d) \wedge ax_{\sqsubseteq\forall}(a,r,b) \wedge CA(a,c) \rightarrow CA(b,d) \quad (35)$$

Predicates CA and RA encode assertions: $CA(A,c)$ encodes $A(c)$; and $RA(R,a,b)$ encodes $R(a,b)$. We used several benchmark datasets that Zhou *et al.* [2015] created by sampling data for two large ontologies: the real-world knowledge base Reactome, and the synthetic benchmark UOBM.

Figure 4 shows the total times (precomputing and reasoning) for VLog and Konclude. No times are shown in cases where Konclude exceeded a one-hour timeout. We observe that our approach leads to competitive performance in (B), which might have been expected since rule engines optimise for large datasets. In particular, VLog might benefit from its compact in-memory presentation of large predicates, which also helps to find rule matches more quickly if many facts can be processed together [Urbani *et al.*, 2016].

8 Conclusions

We introduced Datalog(S) as a convenient high-level language for encoding complex reasoning algorithms, which can be "compiled" into existential rules that can be successfully executed on rule engines available today. This outlines an appealing new method for prototyping reasoning algorithms in an elegant and declarative way, potentially even resulting in highly scalable systems. It also indicates a promising new research direction for the development of modern rule engines.

Acknowledgments

This work is partly supported by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) in project number 389792660 (TRR 248, Center for Perspicuous Systems) and Emmy Noether grant KR 4381/1-1 (DIAMOND).

References

[Abiteboul *et al.*, 1994] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison Wesley, 1994.

[Ahmetaj *et al.*, 2018] Shqiponja Ahmetaj, Magdalena Ortiz, and Mantas Simkus. Rewriting guarded existential rules into small datalog programs. In *Proc. 21st Int. Conf. on Database Theory (ICDT'18)*, volume 98 of *LIPIcs*, pages 4:1–4:24. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

[Aref *et al.*, 2015] Molham Aref, Balder ten Cate, Todd J. Green, Benny Kimelfeld, Dan Olteanu, Emir Pasalic, Todd L. Veldhuizen, and Geoffrey Washburn. Design and implementation of the LogicBlox system. In *Proc. 2015 ACM SIGMOD Int. Conf. on Management of Data*, pages 1371–1382. ACM, 2015.

[Baader *et al.*, 2007] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, second edition, 2007.

[Baget *et al.*, 2011] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. On rules with existential variables: Walking the decidability line. *Artif. Intell.*, 175(9–10):1620–1654, 2011.

[Baget *et al.*, 2015] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, Swan Rocher, and Clément Sipieter. Graal: A toolkit for query answering with existential rules. In *Proc. 9th Int. Web Rule Symposium (RuleML'15)*, volume 9202 of *LNCS*, pages 328–344. Springer, 2015.

[Benedikt *et al.*, 2014] Michael Benedikt, Julien Leblay, and Efthymia Tsamoura. PDQ: proof-driven query answering over web-based data. *PVLDB*, 7(13):1553–1556, 2014.

[Benedikt *et al.*, 2017] Michael Benedikt, George Konstantinidis, Giansalvatore Mecca, Boris Motik, Paolo Papotti, Donatello Santoro, and Efthymia Tsamoura. Benchmarking the chase. In *Proc. 36th Symposium on Principles of Database Systems (PODS'17)*, pages 37–52. ACM, 2017.

[Carral *et al.*, 2017] David Carral, Irina Dragoste, and Markus Krötzsch. Restricted chase (non)termination for existential rules with disjunctions. In *Proc. 26th Int. Joint Conf. on Artif. Intell. (IJCAI'17)*, pages 922–928. ijcai.org, 2017.

[Carral *et al.*, 2018] David Carral, Irina Dragoste, and Markus Krötzsch. The combined approach to query answering in Horn-$\mathcal{ALCHOIQ}$. In *Proc. 16th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'18)*, pages 339–348. AAAI Press, 2018.

[Carral *et al.*, 2019] David Carral, Irina Dragoste, Markus Krötzsch, and Christian Lewe. Chasing sets: How to use existential rules for expressive reasoning (extended technical report). Available at https://iccl.inf.tu-dresden.de/web/Inproceedings3212/en, 2019.

[Cuenca Grau *et al.*, 2013] Bernardo Cuenca Grau, Ian Horrocks, Markus Krötzsch, Clemens Kupke, Despoina Magka, Boris Motik, and Zhe Wang. Acyclicity notions for existential rules and their application to query answering in ontologies. *J. Artif. Intell. Res.*, 47:741–808, 2013.

[Dantsin *et al.*, 2001] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33(3):374–425, 2001.

[Deutsch *et al.*, 2008] Alin Deutsch, Alan Nash, and Jeffrey B. Remmel. The chase revisited. In *Proc. 27th Symposium on Principles of Database Systems (PODS'08)*, pages 149–158. ACM, 2008.

[Geerts *et al.*, 2014] Floris Geerts, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. That's all folks! LLUNATIC goes open source. *PVLDB*, 7(13):1565–1568, 2014.

[Gottlob *et al.*, 2014] Georg Gottlob, Sebastian Rudolph, and Mantas Simkus. Expressiveness of guarded existential rule languages. In *Proc. 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'14)*, pages 27–38. ACM, June 2014.

[Kazakov, 2009] Yevgeny Kazakov. Consequence-driven reasoning for Horn $\mathcal{SHIQ}$ ontologies. In *Proc. 21st Int. Joint Conf. on Artif. Intell. (IJCAI'09)*, pages 2040–2045. IJCAI, 2009.

[Krötzsch *et al.*, 2013] Markus Krötzsch, Sebastian Rudolph, and Pascal Hitzler. Complexities of Horn description logics. *ACM Trans. Comput. Logic*, 14(1):2:1–2:36, 2013.

[Krötzsch, 2011] Markus Krötzsch. Efficient rule-based inferencing for OWL EL. In *Proc. 22nd Int. Joint Conf. on Artif. Intell. (IJCAI'11)*, pages 2668–2673. AAAI Press/IJCAI, 2011.

[Krötzsch *et al.*, 2019] Markus Krötzsch, Maximilian Marx, and Sebastian Rudolph. The power of the terminating chase. In *Proc. 22nd Int. Conf. on Database Theory (ICDT'19)*, volume 127 of *LIPIcs*, pages 3:1–3:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019.

[Nenov *et al.*, 2015] Yavor Nenov, Robert Piro, Boris Motik, Ian Horrocks, Zhe Wu, and Jay Banerjee. RDFox: A highly-scalable RDF store. In *Proc. 14th Int. Semantic Web Conf. (ISWC'15), Part II*, volume 9367 of *LNCS*, pages 3–20. Springer, 2015.

[Ortiz *et al.*, 2010] Magdalena Ortiz, Sebastian Rudolph, and Mantas Simkus. Worst-case optimal reasoning for the Horn-DL fragments of OWL 1 and 2. In *Proc. 12th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'10)*, pages 269–279. AAAI Press, 2010.

[Simančík *et al.*, 2011] František Simančík, Yevgeny Kazakov, and Ian Horrocks. Consequence-based reasoning beyond Horn ontologies. In *Proc. 22nd Int. Joint Conf. on Artif. Intell. (IJCAI'11)*, pages 1093–1098. AAAI Press/IJCAI, 2011.

[Steigmiller *et al.*, 2014] Andreas Steigmiller, Thorsten Liebig, and Birte Glimm. Konclude: System description. *J. Web Semant.*, 27-28:78–85, 2014.

[Urbani *et al.*, 2016] Jacopo Urbani, Ceriel Jacobs, and Markus Krötzsch. Column-oriented Datalog materialization for large knowledge graphs. In *Proc. 30th AAAI Conf. on Artif. Intell. (AAAI'16)*, pages 258–264. AAAI Press, 2016.

[Urbani *et al.*, 2018] Jacopo Urbani, Markus Krötzsch, Ceriel J. H. Jacobs, Irina Dragoste, and David Carral. Efficient model construction for Horn logic with VLog: System description. In *Proc. 9th Int. Joint Conf. on Automated Reasoning (IJCAR'18)*, volume 10900 of *LNCS*, pages 680–688. Springer, 2018.

[Zhou *et al.*, 2015] Yujiao Zhou, Bernardo Cuenca Grau, Yavor Nenov, Mark Kaminski, and Ian Horrocks. PAGOdA: Pay-as-you-go ontology query answering using a Datalog reasoner. *J. of Artif. Intell. Res.*, 54:309–367, 2015.