Chasing Sets: How to Use Existential Rules for Expressive Reasoning

IJCAI Submission #6523

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 any existential rule language that guarantees the so-called *skolem chase* to be finite [Marnette, 2009]. This covers almost every known chase termination criterion [Cuenca Grau *et al.*, 2013]. The only exception we are aware of is a termination criterion by Carral et al. [2017], but their existential rule fragment has P data complexity as well. In spite of the recent advances in theory and practice, we thus 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.

We show that, surprisingly, this apparent limitation to polynomial data complexity does not exist, and that even current rule engines can decide complex logical reasoning tasks given a fixed "program" of rules. This opens a new avenue towards exploiting rule engines for logical reasoning. To provide convenient access to this expressive power, we propose Datalog(S), an extension of Datalog that can reason 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 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 fragment that guarantees chase termination.

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 [Anon., 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. 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 $k \geq 1$.*

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

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.

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 $succ(1,2), succ(2,3), \ldots, succ(n-1,n)$, starting at the constant 1 (the other constant names are immaterial). The following program entails an n-chain of length 2^n:*

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

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

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

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* (40)*, 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 $p^\mathcal{I} \subseteq \mathbf{S_1}^\mathcal{I} \times \ldots \times \mathbf{S_n}^\mathcal{I}$, 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 $B_1 \wedge \ldots \wedge B_\ell \rightarrow H_1 \wedge \ldots \wedge H_k$ 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 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:

$$\rightarrow \exists v.\mathit{empty}(v) \wedge \mathit{set}(v) \tag{4}$$

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

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

Intuitively, $\mathit{set}(s)$ means "s is a set" and $\mathit{empty}(s)$ means "$s = \emptyset$." *SU* is for *Singleton Union*: $SU(a,s,t)$ means "$\{a\} \cup s = t$," and $\mathit{getSU}(a,s)$ 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 $\alpha(\mathit{empty}) = \langle \mathbf{set} \rangle$, $\alpha(SU) = \langle \mathbf{obj}, \mathbf{set}, \mathbf{set} \rangle$, and $\alpha(\mathit{getSU}) = \langle \mathbf{obj}, \mathbf{set} \rangle$. 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:*

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

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

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

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

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

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) 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. □

As the proof 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)

$$getU(v,w) \wedge empty(v) \rightarrow U(v,w,w) \quad (9)$$
$$getU(v,w) \wedge SU(x,v_-,v) \rightarrow getSU(x,w) \quad (10)$$
$$getU(v,w) \wedge SU(x,v_-,v) \wedge SU(x,w,w_+) \rightarrow getU(v_-,w_+) \quad (11)$$
$$getU(v,w) \wedge SU(x,v_-,v) \wedge SU(x,w,w_+) \wedge U(v_-,w_+,u) \rightarrow U(v,w,u) \quad (12)$$
$$SU(x,u,u) \rightarrow in(x,u) \quad (13)$$
$$set(v) \wedge set(w) \rightarrow ckSub(v,v,w) \quad (14)$$
$$ckSub(u,v,w) \wedge SU(x,u_-,u) \wedge in(x,w) \rightarrow ckSub(u_-,v,w) \quad (15)$$
$$ckSub(u,v,w) \wedge empty(u) \rightarrow sub(v,w) \quad (16)$$

Figure 1: Rules for arbitrary unions, membership, and containment

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.

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).

Theorem 3. *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. □

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.

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) 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 $[A] = A(x)$, $[B] = B(x)$, $[\exists R.C] =$

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

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

$\exists y.R(x, y) \wedge C(y)$, $[\forall R.C] = \forall y.R(x, y) \wedge C(y)$, and $[A \sqcap B] = A(x) \wedge B(x)$. $\top$ represents truth, and $\bot$ falsity.

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

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

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-$\mathcal{ALC}$.

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 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:

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

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

Confirmed non-memberships are recorded as $ckNotin(a, u)$. The approach is similar to 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 $\alpha \in \mathcal{A}$:

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

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 3, we obtain a similarly small existential rule set for which the Datalog-first chase terminates:

Theorem 4. *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* $\mathbf{P}$.

7 Evaluation

We conduct two experiments: (A) compute all consequences of the calculus in Figure 2, and (B) compute all entailed axioms of type (21) (class retrieval, CR). All experiments use the rule engine VLog as an implementation for the Datalog-first chase [Urbani *et al.*, 2018] on a MacBookPro (2.4GHz Intel Core i5, 8GB RAM).

Ontology	#Ax.	Time	#Set	#SC	#Ex
40/GO-Anat.	223K	432s	2K	1051K	334K
48/GO-Taxon	142K	387s	19	718K	171K
477/Gazetteer	318K	0.5s	0	162K	167K
533/Chebi	159K	132s	0	965K	351K
786/NCI	152K	549s	12K	2283K	978K

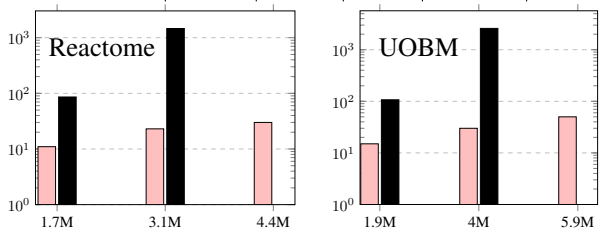

Figure 3: Above, (A) Classification Results; note the log scale; below, (B) CR Results for VLog (pink/grey) and Konclude (black)

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 [Anon., 2019]. We classified five large and diverse ontologies from the Oxford Ontology Repository (OOR, https://www.cs.ox.ac.uk/isg/ontologies/), whose statistics and evaluation results are shown in Figure 3. We show each ontology's OOR-ID, axiom count, reasoning time for VLog, number of non-singleton "set terms" introduced, and count of SC and Ex facts derived.

Columns #Set, #SC, and #Ex confirm that the ontologies require significant reasoning effort. VLog reasoning times were dominated by join computation, especially for rule $(\mathbf{R}_\exists^-)$, whereas our rules for simulating sets had only a minor impact (although they dominate theoretical worst-case complexity). While this shows practicality, highly optimised DL reasoners like Konclude [Steigmiller *et al.*, 2014] are significantly faster, showing potential for rule engine optimisation.

For (B), we extend $\mathcal{R}_{Cl}$ with three rules to solve CR [Anon., 2019], obtaining a rule set $\mathcal{R}_{CR}$. We used several benchmark datasets that Zhou *et al.* created by sampling data for two ontologies [2015]. Figure 3 shows the total preprocessing and reasoning times as measured for VLog and Konclude. For some of the larger samples, Konclude did not finish within a one-hour timeout – no times are reported there.

We observe that our approach leads to competitive performance in (B), which is expected since rule engines optimise for large datasets. We hypothesise that, in part, this is the case because VLog is more space efficient [Urbani *et al.*, 2016] than Konclude and hence, it can process the large amounts of data found in Reactome and UOBM.

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.

References

[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.

[Anon., 2019] Anon. Chasing sets: How to use existential rules for expressive reasoning. https://www.dropbox.com/sh/n9rrslu4ya5sfl2/AADNdtNxWOERTgZZuSe0xvvba?dl=0, 2019.

[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, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, second edition, 2007.

[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 Nick Bassiliades, Georg Gottlob, Fariba Sadri, Adrian Paschke, and Dumitru Roman, editors, *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.*, 2014] David Carral, Cristina Feier, Bernardo Cuenca Grau, Pascal Hitzler, and Ian Horrocks. *EL*-ifying ontologies. In Stéphane Demri, Deepak Kapur, and Christoph Weidenbach, editors, *Proc. of the 7th Int. Joint Conf. onAutomated Reasoning (IJCAR'18)*, volume 8562 of *Lecture Notes in Computer Science*, pages 464–479. Springer, 2014.

[Carral *et al.*, 2017] David Carral, Irina Dragoste, and Markus Krötzsch. Restricted chase (non)termination for existential rules with disjunctions. In Carles Sierra, editor, *Proc. 26th Int. Joint Conf. on Artificial Intelligence (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'16)*, pages 339–348. AAAI Press, 2018.

[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. of Artificial Intelligence Research*, 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.

[Eiter *et al.*, 2012] Thomas Eiter, Magdalena Ortiz, Mantas Simkus, Trung-Kien Tran, and Guohui Xiao. Query rewriting for horn-shiq plus rules. In *Proc. 26th AAAI Conf. on Artif. Intell. (AAAI'12)*. AAAI Press, 2012.

[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.

[ijc, 2011] *Proc. 22nd Int. Joint Conf. on Artif. Intell. (IJCAI'11)*. AAAI Press/IJCAI, 2011.

[Kazakov, 2009] Yevgeny Kazakov. Consequence-driven reasoning for Horn $\mathcal{SHIQ}$ ontologies. In *Proc. 21st Int. Joint Conf. on Artificial Intelligence (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)* [2011], pages 2668–2673.

[Marnette, 2009] Bruno Marnette. Generalized schema-mappings: from termination to tractability. In *Proc. 28th Symposium on Principles of Database Systems (PODS'09)*, pages 13–22. ACM, 2009.

[Nenov *et al.*, 2015] Yavor Nenov, Robert Piro, Boris Motik, Ian Horrocks, Zhe Wu, and Jay Banerjee. RDFox: A highly-scalable RDF store. In Marcelo Arenas et al., editor, *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)* [2011], pages 1093–1098.

[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 Artificial Intelligence (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.

(init)	$conceptName(a) \wedge empty(E) \rightarrow getSU(a, E)$
($\mathbf{S_A}$)	$in(a, H) \rightarrow SC(H, a)$
($\mathbf{S}^0_\sqcap$)	$SC(H, a) \wedge ax_\sqsubseteq(c_\top, c) \rightarrow SC(H, c)$
($\mathbf{S}^1_\sqcap$)	$SC(H, a_1) \wedge ax_\sqsubseteq(a_1, c) \rightarrow SC(H, c)$
($\mathbf{S}^2_\sqcap$)	$SC(H, a_1) \wedge SC(H, a_2) \wedge ax_{\sqcap\sqsubseteq}(a_1, a_2, c) \rightarrow SC(H, c)$
($\mathbf{S}^+_\exists$)	$SC(H, a) \wedge ax_{\sqsubseteq\exists}(a, r, b) \wedge empty(E) \wedge SU(b, E, B) \rightarrow Ex(H, r, B)$
($\mathbf{S}^-_\exists$)	$Ex(H, r, K) \wedge SC(K, a) \wedge ax_{\exists\sqsubseteq}(a, r, b) \rightarrow SC(H, b)$
($\mathbf{S}^\perp_\exists$)	$Ex(H, r, K) \wedge SC(K, c_\perp) \rightarrow SC(H, c_\perp)$
($\mathbf{S}_\forall\alpha$)	$Ex(H, r, K) \wedge SC(H, a) \wedge ax_{\sqsubseteq\forall}(a, r, b) \rightarrow getSU(b, K)$
($\mathbf{S}_\forall\beta$)	$SU(b, K, U) \wedge Ex(H, r, K) \wedge SC(H, a) \wedge ax_{\sqsubseteq\forall}(a, r, b) \rightarrow Ex(H, r, U)$

Figure 4: A simplified translation of the Datalog(S) program in Figure 2 into admissible rules.

A Evaluation Details

A.1 Classification Experiment

Input Ontology Preprocessing

The ontologies selected for this experiment are not in Horn-$\mathcal{ALC}$. Therefore, we first normalise these into the normal form given in [Carral *et al.*, 2014] and then filter all axioms that are not Horn-$\mathcal{ALC}$.

Simplified Classification Rules $\mathcal{R}_{Cl}$

In order to run our experiments, we need to translate the Datalog(S) classification program discussed in Section 5 into an existential rule set, as shown in Section 4. This translation of the Datalog(S) rules from Figure 2 is shown in Figure 4, where rules **($\mathbf{S}_\forall\alpha$)**, **($\mathbf{S}_\forall\beta$)** represent the translation to existential rules of Datalog(S) rule **($\mathbf{R}_\forall$)**, rule **($\mathbf{S}^0_\sqcap$)** is the translation of rule **($\mathbf{R}_\sqcap$)** with $n = 0$, etc. The resulting existential rule set $\mathcal{R}_{Cl}$ – consisting of the rules in Figure 4, rules (4)–(6) and (13), and rule set $\{\rightarrow conceptName(a) | a \in \mathbf{N_C} \cap \mathcal{O}\}$ – has been simplified to optimise performance, and we discuss these optimisations below.

Firstly, we observe that there is a one-to-one correspondence between the active conjunctions derived by the calculus in Figure 2 (left) and the non-empty sets derived by the related Datalog(S) program (right). We know that each set derived by a Datalog(S) program is represented by at least one null introduced during reasoning over the existential rules translation of such a program. Except for the null representing the empty set, each such null corresponds to an active conjunction. Therefore, in the existential rules translation, unary predicate Act - used to signal the correspondence between a null and an active conjunction - becomes redundant. We can then simplify the existential rule set by discarding all atoms with Act predicate (used in rules **($\mathbf{R_A}$)**, **($\mathbf{R}_\sqcap$)** with $n = 0$, **($\mathbf{R}^+_\exists$)**, **($\mathbf{R}_\forall$)**, and rules of type (30)). Instead of the initialisation of active sets as exemplified in (30), we assert $conceptName(a)$ for each concept name a occurring in the input ontology, and add the rule ***(init)*** (Figure 4) to our set. Rule ***(init)*** generates nulls representing the initial singleton sets for each concept name of the input ontology, while rule **($\mathbf{S}_\forall\alpha$)** generates nulls representing new sets (conjunctions which, in turn, become active). For this reason, predicate in (rule **($\mathbf{S_A}$)**) will only apply to the nulls introduced to satisfy rule (5), which represent sets that, in turn, correspond to the active conjunctions.

Secondly, we see that the singleton sets appearing in the head of rule **($\mathbf{R}^+_\exists$)** have already been generated by the application of rule ***(init)***, as variable b can only apply to a concept name. Hence, rule **($\mathbf{R}^+_\exists$)** can be simplified to **($\mathbf{S}^+_\exists$)** as shown in Figure 4. Also, note that the translation of Datalog(S) rule **($\mathbf{R}_\forall$)** into existential rules **($\mathbf{S}_\forall\alpha$)** and **($\mathbf{S}_\forall\beta$)** has been optimised. We can directly obtain the union between the object b and the set K as we do in **($\mathbf{S}_\forall\alpha$)**, instead of first obtaining the union between object b and the empty set, and then between the resulting singleton set $\{b\}$ and the set K, as discussed in Section 4.

Additionally, we observe that in order to simulate the Datalog(S) program in Figure 2, only a subset of the rules that encode set features is needed: it is sufficient to use rules (4)–(6) for encoding the empty set and union with a singleton set (required by rules ***(init)***, **($\mathbf{S}^+_\exists$)**, and **($\mathbf{S}_\forall$)**), and rule (13) for encoding set membership (required by **($\mathbf{S_A}$)**). In fact, (4)–(5) can be modified to eliminate the atom $set(v)$ from the rule heads, as it is only used in rule (14), which is not needed here.

Finally, note that even though rules **($\mathbf{R}^2_\sqcap$)**, **($\mathbf{R}^-_\exists$)**, **($\mathbf{R}^\perp_\exists$)**, **($\mathbf{R}_\forall$)** contain multiple occurrences of the same set-admitting variable in non-special predicates in the body, the normalisation discussed in Section 4 is not necessary for $\mathcal{R}_{Cl}$. In Theorem 5, we show that $\mathcal{R}_{Cl}$ preserves completeness, as distinct nulls that represent the same sets appear interchangeably in the same SC facts generated during the chase (see Lemma 2). Therefore, for a given Horn-$\mathcal{ALC}$ ontology $\mathcal{O}$, the existential rule set $\mathcal{R}_{Cl}$ correctly solves classification.

For the remainder of this section, we refer to $\mathsf{chase}(\mathcal{R}_{Cl})$ as $\mathcal{C}$.

Lemma 2. *Consider two nulls N and N' occurring in $\mathcal{C}$ such that $[N]_\mathcal{C} = [N']_\mathcal{C}$.*

- *If* $SC(N,t) \in \mathcal{C}$ *for some* $t \in \mathbf{T}$*, then* $SC(N',t) \in \mathcal{C}$.
- *If* $Ex(N,r,K) \in \mathcal{C}$ *for some* $r, K \in \mathbf{T}$*, then* $Ex(N',r,K) \in \mathcal{C}$.

Proof. Let $\mathcal{F}_0, \ldots, \mathcal{F}_m$ be some sequence of sets of facts that satisfies all of the following.

- $\mathcal{F}_0 = \emptyset$ and $\mathcal{F}_m = \mathcal{C}$.
- For all $1 \leq i \leq m$, there is a rule $\rho_i = \beta \rightarrow \exists y.\eta \in \mathcal{R}_{Cl}$ and a substitution σ_i such that $\beta\sigma_i \subseteq \mathcal{F}_{i-1}$, $\mathcal{F}_i = \mathcal{F}_{i-1} \cup \eta\sigma_i$, and $\eta\sigma_i \notin \mathcal{F}_{i-1}$.

We show via induction on $i \geq 0$ that:

- If $SC(N,t) \in \mathcal{F}_i$ for some $t \in \mathbf{T}$, then $SC(N',t) \in \mathcal{C}$.
- If $Ex(N,r,K) \in \mathcal{F}_i$ for some $r, K \in \mathbf{T}$, then $Ex(N',r,K) \in \mathcal{C}$.

The base case is trivial, since $\mathcal{F}_0 = \emptyset$. For the induction step, we prove that the claim holds for $1 \leq i \leq m$ assuming it to hold for $i-1$, following a case-by-case analysis depending on ρ_i. Note that we are only interested in rules that produce facts of the form $SC(N,t)$ or $Ex(N,r,K)$ (i.e., rules **($\mathbf{S_A}$)**, **($\mathbf{S}^0_\sqcap$)**, **($\mathbf{S}^1_\sqcap$)**, **($\mathbf{S}^2_\sqcap$)**, **($\mathbf{S}^+_\exists$)**, **($\mathbf{S}^-_\exists$)**, **($\mathbf{S}^\perp_\exists$)**, and **($\mathbf{S}_\forall\beta$)**). For the other rules, the claim trivially holds by induction hypothesis (IH).

($\mathbf{S_A}$) Let $\rho = in(a,H) \rightarrow SC(H,a)$, and $\sigma_i(H) = N$. Then, $in(\sigma_i(a), N) \in \mathcal{F}_{i-1}$. Since $[N]_\mathcal{C} = [N']_\mathcal{C}$ and by rule (13), we have that $in(\sigma_i(a), N') \in \mathcal{C}$. Hence, $SC(N', \sigma_i(a)) \in \mathcal{C}$.

($\mathbf{S}^0_\sqcap$) Let $\rho = SC(H,a) \wedge ax_\sqsubseteq(c_\top, c) \rightarrow SC(H,c)$, and $\sigma_i(H) = N$. Then, $\{SC(N, \sigma_i(a)), ax_\sqsubseteq(c_\top, \sigma_i(c)\} \subseteq \mathcal{F}_{i-1}$. By IH, $SC(N', \sigma_i(a_1)) \in \mathcal{C}$. Hence, $SC(N', \sigma_i(c)) \in \mathcal{C}$.

($\mathbf{S}^1_\sqcap$) Let $\rho_i = SC(H,a_1) \wedge ax_\sqsubseteq(a_1, c) \rightarrow SC(H,c)$, and $\sigma_i(H) = N$. Then, $\{SC(N, \sigma_i(a_1)), ax_\sqsubseteq(\sigma_i(a_1), \sigma_i(c))\} \subseteq \mathcal{F}_{i-1}$. By IH, $SC(N', \sigma_i(a_1)) \in \mathcal{C}$. Hence, $SC(N', \sigma_i(c)) \in \mathcal{C}$.

($\mathbf{S}^2_\sqcap$) Let $\rho_i = SC(H,a_1) \wedge SC(H,a_2) \wedge ax_{\sqcap\sqsubseteq}(a_1, a_2, c) \rightarrow SC(H,c)$, and $\sigma_i(H) = N$. Then, $\{SC(N, \sigma_i(a_1)), SC(N, \sigma_i(a_2)), ax_{\sqcap\sqsubseteq}(\sigma_i(a_1), \sigma_i(a_2), \sigma_i(c))\} \subseteq \mathcal{F}_{i-1}$. By IH, $\{SC(N', \sigma_i(a_1)), SC(N', \sigma_i(a_2))\} \subseteq \mathcal{C}$. Hence, $SC(N', \sigma_i(c)) \in \mathcal{C}$.

($\mathbf{S}^+_\exists$) Let $\rho_i = SC(H,a) \wedge ax_{\sqsubseteq\exists}(a,r,b) \wedge empty(E) \wedge SU(b,E,B) \rightarrow Ex(H,r,B)$, and $\sigma_i(H) = N$. Then, $\{SC(N, \sigma_i(a)) \wedge ax_{\sqsubseteq\exists}(\sigma_i(a), \sigma_i(r), \sigma_i(b)) \wedge empty(\sigma_i(E)) \wedge SU(\sigma_i(b), \sigma_i(E), \sigma_i(B))\} \subseteq \mathcal{F}_{i-1}$. By IH, $SC(N', \sigma_i(a)) \in \mathcal{C}$. Hence, $Ex(N', \sigma_i(r), \sigma_i(B)) \in \mathcal{C}$.

($\mathbf{S}^-_\exists$) Let $\rho_i = Ex(H,r,K) \wedge SC(K,a) \wedge ax_{\exists\sqsubseteq}(a,r,b) \rightarrow SC(H,b)$, and $\sigma_i(H) = N$. Then, $\{Ex(N, \sigma_i(r), \sigma_i(K)), SC(\sigma_i(K), \sigma_i(a)), ax_{\exists\sqsubseteq}(\sigma_i(a), \sigma_i(r), \sigma_i(b))\} \subseteq \mathcal{F}_{i-1}$. By IH, $Ex(N', \sigma_i(r), \sigma_i(K)) \in \mathcal{C}$. Hence, $SC(N', \sigma_i(b)) \in \mathcal{C}$.

($\mathbf{S}^\perp_\exists$) For $\rho_i = Ex(H,r,K) \wedge SC(K, c_\perp) \rightarrow SC(H, c_\perp)$, the proof is analogous to the case above.

($\mathbf{S}_\forall\beta$) Let $\rho_i = SU(b,K,U) \wedge Ex(H,r,K) \wedge SC(H,a) \wedge ax_{\sqsubseteq\forall}(a,r,b) \rightarrow Ex(H,r,U)$, and $\sigma_i(H) = N$. Then, $\{SU(\sigma_i(b), \sigma_i(K), \sigma_i(U)), Ex(N, \sigma_i(r), \sigma_i(K)), SC(N, \sigma_i(a)), ax_{\sqsubseteq\forall}(\sigma_i(a), \sigma_i(r), \sigma_i(b))\} \subseteq \mathcal{F}_{i-1}$. By IH, $\{Ex(N', \sigma_i(r), \sigma_i(K)), SC(N', \sigma_i(a))\} \subseteq \mathcal{C}$. Hence, $Ex(N', \sigma_i(r), \sigma_i(U)) \in \mathcal{C}$.

□

For a function $f : \mathbf{N} \rightarrow \mathbf{N}$ and a fact ϕ, we denote as $f(\phi)$ the fact obtained by replacing each syntactic occurrence of a null n in ϕ with $f(n)$. For a set of facts $\mathcal{F}$, we denote as $f(\mathcal{F})$ the set of facts $\bigcup_{\phi \in \mathcal{F}} f(\phi)$. We abuse notation and say that two sets of facts $\mathcal{F}_1$ and $\mathcal{F}_2$ are *equal* ($\mathcal{F}_1 = \mathcal{F}_2$), if there is a bijection f defined on the nulls in $\mathcal{F}_1$ with values in the nulls in $\mathcal{F}_2$ such that $f(\mathcal{F}_1) = \mathcal{F}_2$ and $f^-(\mathcal{F}_2) = \mathcal{F}_1$. We also say that $\mathcal{F}_1 \subseteq \mathcal{F}_2$, if there is a set of facts $\mathcal{F}'_1$ such that $\mathcal{F}_1 = \mathcal{F}'_1$ (according to the above equality definition), and $\mathcal{F}'_1 \subseteq \mathcal{F}_2$.

Theorem 5. *The set* $\mathsf{chase}(\mathcal{R}_{Cl})$ *is equal to* $\mathsf{chase}(\mathcal{R}'_{Cl})$*, where* $\mathcal{R}'_{Cl}$ *is the normalisation of the rules in* $\mathcal{R}_{Cl}$.

Proof. For brevity, we denote $\mathsf{chase}(\mathcal{R}'_{Cl})$ as $\mathcal{C}'$. Note that $\mathcal{C} \subseteq \mathcal{C}'$ follows from the fact that, for a rule ρ, its normalisation ρ', and a set of facts $\mathcal{F}$, we have that $\rho(\mathcal{F}) \subseteq \rho'(\mathcal{F})$ (considering fact set equality as discussed above). We proceed to prove that $\mathcal{C}' \subseteq \mathcal{C}$. Let $\mathcal{F}_0, \ldots, \mathcal{F}_m$ be some sequence of sets of facts that satisfies all of the following.

- $\mathcal{F}_0 = \emptyset$ and $\mathcal{F}_m = \mathcal{C}'$.
- For all $1 \leq i \leq m$, there is a rule $\rho'_i = \beta' \rightarrow \exists y.\eta \in \mathcal{R}'_{Cl}$ and a substitution σ_i such that $\beta'\sigma_i \subseteq \mathcal{F}_{i-1}$ and $\mathcal{F}_i = \mathcal{F}_{i-1} \cup \eta\sigma_i$.

We show via induction on $i \geq 0$ that $\mathcal{F}_i \subseteq \mathcal{C}$. The base case is trivial, since $\mathcal{F}_0 = \emptyset$. For the induction step, we prove that the claim holds for $1 \leq i \leq m$ assuming it to hold for $i-1$, following a case-by-case analysis depending on $\rho'_i \in \mathcal{R}'_{Cl}$. Let $\rho_i = \beta \rightarrow \exists y.\eta \in \mathcal{R}_{Cl}$ be a rule such that, if we normalise it, we obtain ρ'_i.

The rules in $\mathcal{R}_{Cl}$ that are not normalised are: **($\mathbf{S}^2_\sqcap$)**, **($\mathbf{S}^-_\exists$)**, **($\mathbf{S}^\perp_\exists$)**, and **($\mathbf{S}_\forall$)**. Even though rule **($\mathbf{S}^+_\exists$)** has multiple occurrences of variable E, the atom $empty(E)$ in the rule body insures that E can be matched by only a single term (i.e., the null introduced to satisfy the empty set existential restriction). Thus, we do not need to consider normalisation for this rule.

- If ρ_i is one of the rules (4)–(6), (13), ***(init)***, **($\mathbf{S_A}$)**, **($\mathbf{S}^0_\sqcap$)**, **($\mathbf{S}^1_\sqcap$)**, and **($\mathbf{S}^+_\exists$)**, or belongs to the rule set $\{\rightarrow conceptName(a) | a \in \mathbf{N_C} \cap \mathcal{O}\}$, then $\rho'_i = \rho_i$, which implies that $\beta'\sigma = \beta\sigma$. By induction hypothesis (IH), we have that $\mathcal{F}_{n-1} \subseteq \mathcal{C}$.
  - If ρ_i is a generating rule ((4) or (5)), then either $\langle \rho_i, \sigma_i \rangle$ is applicable to $\mathcal{F}_{n-1}$, or there is a substitution $\pi \supseteq \sigma_i \setminus \sigma_i(v)$, v being the existential variable in (4) and (5), such that $\eta\pi \subseteq \mathcal{F}_{n-1}$. In any case, we have that $\eta\pi \subseteq \mathcal{C}$. It is easy to see that there is a bijective function f with $f(\sigma_i(v)) = \pi(v)$, such that $f(\mathcal{F}_n) = f(\mathcal{F}_{n-1} \cup \eta'\sigma_i) = \mathcal{F}_{n-1} \cup \eta\pi \subseteq \mathcal{C}$. Therefore, $\mathcal{F}_n \subseteq \mathcal{C}$.
  - If ρ_i is one of the remaining non-generating rules, then either $\langle \rho_i, \sigma_i \rangle$ is applicable to $\mathcal{F}_{n-1}$, or $\eta\sigma_i \subseteq \mathcal{F}_{n-1}$. In any case, we obtain that $\eta\sigma_i \subseteq \mathcal{C}$. Since $\mathcal{F}_n = \mathcal{F}_{n-1} \cup \eta\sigma_i$, our proof for this rule is complete.
- If ρ_i is **($\mathbf{S}^2_\sqcap$)**, then $\beta' = SC(H, a_1) \wedge SC(H', a_2) \wedge H = H' \wedge ax_{\sqcap\sqsubseteq}(a_1, a_2, c)$. Since $\beta'\sigma_i \subseteq \mathcal{F}_{n-1}$, we have that $\{sub(\sigma_i(H), \sigma_i(H')), sub(\sigma_i(H'), \sigma_i(H))\} \subseteq \mathcal{F}_{n-1} \subseteq \mathcal{C}$ (by IH). This implies that $[\sigma(H)]_\mathcal{C} = [\sigma(H')]_\mathcal{C}$. By Lemma 2, we obtain that, if $SC(\sigma_i(H'), \sigma_i(a_2)) \in \mathcal{C}$, then $SC(\sigma_i(H), \sigma_i(a_2)) \in \mathcal{C}$. This implies that $\mathcal{C}$ contains the application of $\langle \rho_i, \sigma_i \rangle$ has been applicable in $\mathcal{C}$, hence $\eta\sigma_i \in \mathcal{C}$.
- If ρ_i is **($\mathbf{S}^-_\exists$)**, then $\beta' = Ex(H, r, K) \wedge SC(K', a) \wedge K = K' \wedge ax_{\exists\sqsubseteq}(a, r, b)$. Since $\beta'\sigma_i \subseteq \mathcal{F}_{n-1}$, we have that $\{sub(\sigma_i(K), \sigma_i(K')), sub(\sigma_i(K'), \sigma_i(K))\} \subseteq \mathcal{F}_{n-1} \subseteq \mathcal{C}$ (by IH). This implies that $[\sigma(K)]_\mathcal{C} = [\sigma(K')]_\mathcal{C}$. By Lemma 2, we obtain that, if $SC(\sigma_i(K'), \sigma_i(a)) \in \mathcal{C}$, then $SC(\sigma_i(K), \sigma_i(a)) \in \mathcal{C}$. This implies that $\mathcal{C}$ contains the application of $\langle \rho_i, \sigma_i \rangle$, hence $\eta\sigma_i \in \mathcal{C}$.
- If ρ_i is **($\mathbf{S}^\perp_\exists$)**, the proof is analogous to the case above.
- If ρ_i is **($\mathbf{S}_\forall\alpha$)**, then $\beta' = Ex(H, r, K) \wedge SC(H', a) \wedge H = H' \wedge ax_{\sqsubseteq\forall}(a, r, b)$. Since $\beta'\sigma_i \subseteq \mathcal{F}_{n-1}$, we have that $\{sub(\sigma_i(H), \sigma_i(H')), sub(\sigma_i(H'), \sigma_i(H))\} \subseteq \mathcal{F}_{n-1} \subseteq \mathcal{C}$ (by IH). This implies that $[\sigma(H)]_\mathcal{C} = [\sigma(H')]_\mathcal{C}$. By Lemma 2, we obtain that, if $SC(\sigma_i(H'), \sigma_i(a)) \in \mathcal{C}$, then $SC(\sigma_i(H), \sigma_i(a)) \in \mathcal{C}$. This implies that $\mathcal{C}$ contains the application of $\langle \rho_i, \sigma_i \rangle$, hence $\eta\sigma_i \in \mathcal{C}$.
- If ρ_i is **($\mathbf{S}_\forall\beta$)**, then $\beta' = SU(b, K, U) \wedge Ex(H, r, K) \wedge SC(H', a) \wedge H = H' \wedge ax_{\sqsubseteq\forall}(a, r, b)$. Note that variable K does not need to be normalised, as the normalisation step must occur before translating the Datalog(S) rule to an existential rule, and SU predicate is only introduced by this translation. The rest of the proof is analogous to the case above.

□

A.2 Class Retrieval Experiment

Input Ontology Pre-processing

Neither Reactome nor Uniprot are Horn-$\mathcal{ALC}$ ontologies. Therefore, we first normalised these into the normal form given in [Carral *et al.*, 2014], filtered out all axioms not in Horn-$\mathcal{ALCHI}$ [Krötzsch *et al.*, 2013], and then normalised the resulting ontologies into Horn-$\mathcal{ALC}$ [Carral *et al.*, 2014].

Class Retrieval Rules $\mathcal{R}_{CR}$

In order to solve class retrieval, we extend the rule set $\mathcal{R}_{Cl}$ with the following rules.

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

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

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

Atoms over predicates CA and RA are used to encode class and role assertions, respectively. That is, the fact $CA(A, c)$ corresponds to the assertion $A(c)$; the fact $RA(R, a, b)$ corresponds to the assertion $R(a, b)$. The correctness of our approach follows from the correctness of the classification calculus discussed above and the results from [Eiter *et al.*, 2012].

Experiment Details

Figure 3 shows the results of our experiment. The horizontal axis represents the size of each ABox sample—i.e., the number of axioms of type (21) and (22). The vertical axis shows the reasoning time in seconds for both VLog and Konclude. Note that we calculate Konclude reasoning time as the total time taken for precomputing, classification and realisation. Loading and pre-processing durations are not taken into consideration for any of the evaluated reasoners.

To initialise the facts over predicates CA and RA in VLog, we add the following rules

$$classNTriple(x, \textit{rdf:type}, c) \rightarrow CA(c, x) \tag{36}$$

$$roleNTriple(x, r, y) \rightarrow RA(r, x, y) \tag{37}$$

where $classNTriple$ and $roleNTriple$ are the predicates associated to the input N-Triple files.

A.3 Tool Versions

In this evaluation, we have used the following versions of VLog and Konclude.

- VLog: current master branches of *knowsys/vlog4j* and *karmaresearch/vlog*.
  - Commit: https://github.com/knowsys/vlog4j/commit/861b7a8e4540203e82d1eba8bde00330b652a6d9.
  - Commit https://github.com/karmaresearch/vlog/commit/8419c2216095b228b286dbd7802fdc2c0dd6f8df.
- Konclude: current static version *Konclude-v0.6.2-544-OSX-x64-Static-Qt4.8.5*.

B Guarded Rules Reasoning

Given a guarded rule set $\mathcal{R}$ with bounded arity $\hat{a}$, we show how to—following the transformation proposed in [Ahmetaj *et al.*, 2018]—compute a Datalog(S) program $\mathcal{P}$ that preserves fact entailments over $\mathcal{R}$ and is polynomial in $\mathcal{R}$. To simplify our arguments, we assume that $\mathcal{R}$ satisfies all of the following.

1. Every Datalog rule in $\mathcal{R}$ contains (exactly) two atoms in the body and one in the head.
2. Every non-Datalog rule in $\mathcal{R}$ contains (exactly) one atom in the body and one in the head.
3. All universally quantified variables occurring in some rule in $\mathcal{R}$ are in the generic set of variables $\mathbf{X} = \{x_1, \ldots, x_{\hat{a}}\}$.

Note that all of the above can be established without loss of generality. Using simple decomposition of the bodies, we can satisfy (1) and (2). Since $\mathcal{R}$ is guarded and the maximum arity of a predicate in $\mathcal{R}$ is $\hat{a}$, the body of every rule may contain at most $\hat{a}$ distinct variables. Therefore, we can always bijectively rename the variables in any rule in $\mathcal{R}$ to produce an equivalent rule that satisfies (3). Let $\hat{t}$ be the number of all atoms that can be constructed using predicates from $\mathcal{R}$ and variables in $\mathbf{X}$, let $\alpha_1, \ldots, \alpha_{\hat{t}}$ be some sequence containing all such atoms, and let $\mathcal{A} = \{\alpha_1, \ldots, \alpha_{\hat{t}}\}$. A *type* is a subset of $\mathcal{A}$.

To compute $\mathcal{P}$, we study the transformation presented in [Ahmetaj *et al.*, 2018]. Their rules are organised in groups **(I)** to **(IX)**, where **(II)** and **(VI)** only matter if the rules in the input set are contain disjunctions in the head. To verify that $\mathcal{P}$ can be used to solve fact entailment over $\mathcal{R}$, we show a correspondence with the entailments of the rule set $\mathcal{D}$ that results from applying the transformation in [Ahmetaj *et al.*, 2018] to $\mathcal{R}$. We write $\mathcal{D}_{(\ell)}$ with $\ell \in \{\mathbf{I}, \ldots, \mathbf{IX}\}$ to refer to the subset of $\mathcal{D}$ that results from including the rules considered in groups **(I)** through (ℓ) (always excluding **(II)** and **(VI)**). Analogously, we write $\mathcal{P}_{(\ell)}$ to refer to the corresponding subset of $\mathcal{P}$.

(I) Collecting the constants

For all $p \in \mathbf{P}(\mathcal{R})$ and all $i \in \{1, \ldots, ar(p)\}$; we add the following rules to $\mathcal{P}$.

$$p(z_1, \ldots, z_{ar(p)}) \rightarrow const(z_i) \qquad const(z_i) \wedge \ldots \wedge const(z_i) \rightarrow const^i(z_1, \ldots, z_i) \tag{38}$$

After each group, we include a lemma (e.g., Lemma 3) to progressively establish the correspondence between the entailments of the Datalog rule set $\mathcal{D}$ and the Datalog(S) program $\mathcal{P}$.

Lemma 3. *Consider a fact set $\mathcal{F}$ and a fact ϕ of the form $const(c)$ or $const^k(c_1, \ldots, c_k)$. Then, $\mathcal{D}_{(\mathbf{I})} \cup \mathcal{F} \models \phi$ iff $\mathcal{P}_{(\mathbf{I})} \cup \mathcal{F} \models \phi$.*

(III) A linear order over types

We add the facts $first(\emptyset)$, $full(\emptyset)$, $next(\emptyset, \{c_{\alpha_1}\})$, $parts(\{c_{\alpha_1}\}, c_{\alpha_1}, \emptyset)$, $succ(c_{\alpha_1}, c_{\alpha_2})$, $succ(c_{\alpha_2}, c_{\alpha_3}), \ldots, succ(c_{\alpha_{n-1}}, c_{\alpha_n})$, $last(\{c_{\alpha_1}, \ldots, c_{\alpha_n}\}) \in \mathcal{P}$. Also, we add the following rules to $\mathcal{P}$.

$$next(U, V) \wedge parts(V, x, V') \wedge next(V', W') \rightarrow next(V, \{x\} \cup W') \wedge parts(\{x\} \cup W', x, W') \tag{39}$$

$$next(U, V) \wedge parts(V, x, V') \wedge full(V') \wedge succ(x, y) \rightarrow next(V, \{y\}) \wedge parts(\{y\}, y, \emptyset) \wedge full(V) \tag{40}$$

$$next(x, y) \rightarrow Type(x) \wedge Type(y) \tag{41}$$

We introduce some functions to formulate Lemma 4. For all types $\tau \subseteq \mathcal{A}$, let $VT(\tau) = r_1, \ldots, r_{\hat{t}}$ be the sequence over the special constants 0 and 1 with $r_i = 1$ iff $\alpha_i \in \tau$ for all $1 \leq i \leq \hat{t}$, and let $ST(\tau)$ be some (arbitrarily chosen) set term $\{c_{\beta_1}, \ldots, c_{\beta_n}\}$ with $\tau = \{\beta_i \mid c_{\beta_i}, 1 \leq i \leq n\}$.

Lemma 4. *Consider a set of facts $\mathcal{F}$. Then,*

- *$\mathcal{D}_{(\mathbf{III})} \cup \mathcal{F} \models Type(VT(\tau))$ and $\mathcal{P}_{(\mathbf{III})} \cup \mathcal{F} \models Type(ST(\tau))$ for all types $\tau \subseteq \mathcal{A}$,*
- *$\mathcal{D}_{(\mathbf{III})} \cup \mathcal{F} \models first^m(VT(\emptyset))$, $\mathcal{P}_{(\mathbf{III})} \cup \mathcal{F} \models first(ST(\emptyset))$,*
- *$\mathcal{D}_{(\mathbf{III})} \cup \mathcal{F} \models last^m(VT(\mathcal{A}))$, $\mathcal{P}_{(\mathbf{III})} \cup \mathcal{F} \models last(ST(\mathcal{A}))$, and*
- *$\mathcal{D}_{(\mathbf{III})} \cup \mathcal{F} \models next^m(VT(\tau), VT(\upsilon))$ iff $\mathcal{P}_{(\mathbf{III})} \cup \mathcal{F} \models next(ST(\tau), ST(\upsilon))$ for all types $\tau, \upsilon \subseteq \mathcal{A}$.*

(IV) Implementing Step ($\mathbf{M}_\forall$)

For all Datalog rules $B \wedge C \to H \in \mathcal{R}$ and all substitutions $\sigma : \mathbf{V}(B) \wedge \mathbf{V}(C) \to \mathbf{X}$, we add the fact $\mathit{DR}(c_{B\sigma}, c_{C\sigma}, c_{H\sigma}) \in \mathcal{P}$. Also, we add the following rule to $\mathcal{P}$.

$$\mathit{Type}(T) \wedge \mathit{DR}(b, c, h) \wedge b \in T \wedge c \in T \wedge h \notin T \to \mathit{Marked}(T) \tag{42}$$

Lemma 5. *Consider a set of facts $\mathcal{F}$ and a type $\tau \subseteq \mathcal{A}$. Then, $\mathcal{D}_{(\mathbf{IV})} \cup \mathcal{F} \models \mathit{Marked}(\mathit{VT}(\tau))$ iff $\mathcal{P}_{(\mathbf{IV})} \cup \mathcal{F} \models \mathit{Marked}(\mathit{ST}(\tau))$.*

(V) Implementing Step ($\mathbf{M}_\exists$)

For all non-Datalog rules $\rho = B \to \exists \vec{y}.H \in \mathcal{R}$ and all substitutions $\sigma : \mathbf{V}(B) \to \mathbf{X}$, we add $\mathit{RuleSubBod}(c_\rho, c_\sigma, c_{B\sigma})$, $\mathit{RuleSubFr}(c_\rho, c_\sigma, \{c_{u_1}, \ldots, c_{u_n}\}) \in \mathcal{P}$ where $\{u_1, \ldots, u_n\} = \sigma(\mathbf{V}(B) \cap \mathbf{V}(H))$. Let $\{\sigma_1, \ldots, \sigma_n\}$ be the set of all substitutions $\sigma' : \mathbf{V}(H) \to \mathbf{X}$ such that $\sigma(\mathbf{V}(B) \cap \mathbf{V}(H)) = \sigma'(\mathbf{V}(B) \cap \mathbf{V}(H))$. Then, we add $\{\mathit{RuleSubAllHeads}(c_\rho, c_\sigma, \{c_{H\sigma_1}, \ldots, c_{H\sigma_n}\})\} \cup \{\mathit{RuleSubHead}(c_\rho, c_\sigma, c_{H\sigma_i}) \mid 1 \leq i \leq n\} \in \mathcal{P}$. For all atoms α of the form $p(z_1, \ldots, z_n) \in \mathcal{A}$ with $p \in \mathbf{P}(\mathcal{R})$ and $z_1, \ldots, z_n \in \mathbf{X}$, we add $\mathit{AtomVars}(c_\alpha, \{c_{z_1}, \ldots, c_{z_n}\}) \in \mathcal{P}$.

Also, we add the following rules to $\mathcal{P}$.

$$\mathit{Type}(T) \wedge \mathit{Marked}(T') \wedge \mathit{RuleSubBod}(r, s, b) \to \mathit{MarkedOne}(r, s, T, T') \tag{43}$$

$$\mathit{Type}(T) \wedge \mathit{Type}(T') \wedge \mathit{AtomVars}(a, V) \wedge \mathit{RuleSubFr}(r, s, F) \wedge V \subseteq F \wedge a \in T \wedge a \notin T' \to \mathit{MarkedOne}(r, s, T, T') \tag{44}$$

$$\mathit{Type}(T) \wedge \mathit{Type}(T') \wedge \mathit{AtomVars}(a, V) \wedge \mathit{RuleSubFr}(r, s, F) \wedge V \subseteq F \wedge a \notin T \wedge a \in T' \to \mathit{MarkedOne}(r, s, T, T') \tag{45}$$

$$\mathit{Type}(T) \wedge \mathit{Type}(T') \wedge \mathit{RuleSubBody}(r, s, b) \to \mathit{Aux}(r, s, T, T', \emptyset) \tag{46}$$

$$\mathit{Aux}(r, s, T, T', X) \wedge \mathit{RuleSubHead}(r, s, h) \wedge h \notin T' \to \mathit{Aux}(r, s, T, T', X \cup \{h\}) \tag{47}$$

$$\mathit{Aux}(r, s, T, T', X) \wedge \mathit{RuleSubAllHeads}(r, s, X) \to \mathit{MarkedOne}(r, s, T, T') \tag{48}$$

$$\mathit{MarkedOne}(r, s, T, T') \wedge \mathit{first}(T') \to \mathit{MarkedUntil}(r, s, T, T') \tag{49}$$

$$\mathit{MarkedUntil}(r, s, S, T) \wedge \mathit{next}(T, V) \wedge \mathit{MarkedOne}(r, s, S, V) \to \mathit{MarkedUntil}(r, s, S, V) \tag{50}$$

$$\mathit{MarkedUntil}(r, s, S, T) \wedge \mathit{RuleSubBod}(r, s, b) \wedge b \in S \wedge \mathit{last}(T) \to \mathit{Marked}(S) \tag{51}$$

Lemma 6. *Consider a set of facts $\mathcal{F}$. Then,*

- *for all types $\tau, \tau' \subseteq \mathcal{A}$, rules ρ, and substitutions σ; $\mathcal{D}_{(\mathbf{V})} \cup \mathcal{F} \models \mathit{MarkedOne}_{\rho,\sigma}(\mathit{VT}(\tau), \mathit{VT}(\tau'))$ iff $\mathcal{P}_{(\mathbf{V})} \cup \mathcal{F} \models \mathit{MarkedOne}(c_\rho, c_\sigma, \mathit{ST}(\tau), \mathit{ST}(\tau'))$, and $\mathcal{D}_{(\mathbf{V})} \cup \mathcal{F} \models \mathit{MarkedUntil}_{\rho,\sigma}(\mathit{VT}(\tau), \mathit{VT}(\tau'))$ iff $\mathcal{P}_{(\mathbf{V})} \cup \mathcal{F} \models \mathit{MarkedUntil}(c_\rho, c_\sigma, \mathit{ST}(\tau), \mathit{ST}(\tau'))$, and*
- *$\mathcal{D}_{(\mathbf{V})} \cup \mathcal{F} \models \mathit{Marked}(\mathit{VT}(\tau))$ iff $\mathcal{P}_{(\mathbf{V})} \cup \mathcal{F} \models \mathit{Marked}(\mathit{ST}(\tau))$ for all types $\tau \subseteq \mathcal{A}$.*

(VII) Generating abstract types from marked types.

For all $i \in \{1, \ldots, \hat{t}\}$, we add the fact $\mathit{Paired}(c_{\alpha_i}, d_{\alpha_i}) \in \mathcal{P}$. Furthermore, we add the following rule to $\mathcal{P}$.

$$\mathit{Marked}(U) \wedge a \notin U \wedge \mathit{Marked}(U \cup \{a\}) \wedge \mathit{Paired}(a, a') \to \mathit{Marked}(U \cup \{a, a'\}) \tag{52}$$

We introduce a function to formulate Lemma 7. Let *VTS* be the function mapping a sequence $\vec{r} = r_1, \ldots, r_{\hat{t}}$ over the special constants $\{0, 1, 2\}$ to the set term $\mathit{VTS}(\vec{r}) = S$ defined as follows for all $1 \leq i \leq \hat{t}$: $c_{\alpha_i}, d_{\alpha_i} \notin S$ if $r_i = 0$, $c_{\alpha_i} \in S$ and $d_{\alpha_i} \notin S$ if $r_i = 1$, and $c_{\alpha_i}, d_{\alpha_i} \in S$ if $r_i = 2$.

Lemma 7. *For a fact set $\mathcal{F}$ and a sequence $\vec{r}$, $\mathcal{D}_{(\mathbf{VII})} \cup \mathcal{F} \models \mathit{Marked}(\vec{r})$ iff $\mathcal{P}_{(\mathbf{VII})} \cup \mathcal{F} \models \mathit{Marked}(\mathit{VTS}(\vec{r}))$.*

(VIII) Constructing all horn types.

For all $1 \leq i \leq \hat{t}$, we add $\mathit{Horn}(\{c_{\alpha_1}, \ldots, c_{\alpha_i - 1}, c_{\alpha_i + 1}, \ldots, c_{\alpha_i}\}) \in \mathcal{P}$. Also, we add the rules to $\mathcal{P}$ for all $1 \leq i \leq \hat{t}$.

$$\mathit{Horn}(S) \wedge c_{\alpha_i} \in S \to \mathit{Horn}(X \cup \{d_{\alpha_i}\}) \tag{53}$$

$$\mathit{Marked}(S) \wedge \mathit{Horn}(S) \to \mathit{MarkedHorn}(S) \tag{54}$$

Lemma 8. *For a fact set $\mathcal{F}$ and a sequence $\vec{r}$, $\mathcal{D}_{(\mathbf{VIII})} \cup \mathcal{F} \models \mathit{MarkedHorn}(\vec{r})$ iff $\mathcal{P}_{(\mathbf{VIII})} \cup \mathcal{F} \models \mathit{MarkedHorn}(\mathit{VTS}(\vec{r}))$.*

(IX) Implementing the full TGDs from Σ_{Horn}.

For all $i, j \in \{1, \ldots, \hat{t}\}$, we add the following rules to $\mathcal{P}$.

$$\mathit{const}^{\hat{a}}(x_1, \ldots, x_{\hat{a}}) \wedge \mathit{MarkedHorn}(S) \wedge c_{\alpha_i} \notin S \to \mathit{Proj}_i^{\hat{t}}(x_1, \ldots, x_{\hat{a}}, S) \tag{55}$$

$$\mathit{Proj}_i^j(x_1, \ldots, x_{\hat{a}}, S) \wedge c_{\alpha_j} \in S \wedge d_{\alpha_j} \notin S \wedge \alpha_j \to \mathit{Proj}_i^{j-1}(x_1, \ldots, x_{\hat{a}}, S) \tag{56}$$

$$\mathit{Proj}_i^j(x_1, \ldots, x_{\hat{a}}, S) \wedge c_{\alpha_j} \in S \wedge d_{\alpha_j} \in S \to \mathit{Proj}_i^{j-1}(x_1, \ldots, x_{\hat{a}}, S) \tag{57}$$

$$\mathit{Proj}_i^j(x_1, \ldots, x_{\hat{a}}, S) \wedge c_{\alpha_j} \notin S \wedge d_{\alpha_j} \in S \to \mathit{Proj}_i^{j-1}(x_1, \ldots, x_{\hat{a}}, S) \tag{58}$$

$$\mathit{Proj}_i^0(x_1, \ldots, x_{\hat{a}}, S) \to \alpha_i \tag{59}$$

Lemma 9. *Consider a fact set $\mathcal{F}$ and a fact ϕ of the form $Proj_i^0(\vec{r})$. Then, $\mathcal{D}_{(IX)} \cup \mathcal{F} \models \phi$ iff $\mathcal{P}_{(IX)} \cup \mathcal{F} \models Proj_i^0(\vec{r}, S)$ for some term set S.*

After considering all groups **(I)**-**(IX)**, we can establish that $\mathcal{P}$ can be used to solve fact entailment over $\mathcal{R}$ and that this program is of moderate size.

Theorem 6. *For a fact ϕ defined over the signature of $\mathcal{R}$, we have that $\mathcal{R} \models \phi$ iff $\mathcal{P} \models \phi$. Furthermore, (i) $\mathcal{P}$ is polynomial in the size of $\mathcal{R}$, and (ii) the number of variables and atoms in every rule in $\mathcal{R}$ is bounded.*

Proof. The first part of the theorem follows from Lemma 9. Statement (ii) can be easily verified by inspecting the rules added to $\mathcal{P}$. To show (ii), we count the number of facts added to $\mathcal{P}$ in every group (we ignore groups where facts are not added).

(III) $\hat{t} + 2$ facts are added in this group.

(IV) At most $|\mathcal{R}|$ are added in this group.

(V) All of the facts introduced in this group are defined over the predicates *RuleSubBod*, *RuleSubFron*, *RuleSubAllHeads*, *RuleSubHeads*, and *AtomVars*.

- We add at most $|\mathcal{R}| \times \hat{a}^2$ facts over the predicates *RuleSubBod* (resp. *RuleSubFron*, *RuleSubAllHeads*).
- We add at most $|\mathcal{R}| \times \hat{a}^2 \times \hat{a}^2$ over the predicate *RuleSubHeads*.
- We add at most $\hat{a}^2$ multiplied by the number of predicates in $\mathcal{R}$ over the predicate *AtomVars*.

(VI) The number of facts added in **(VI)** is $\hat{t}$.

(VII) The number of facts added in **(VI)** is $\hat{t}$.

Note that since $\hat{a}$ is bounded, $\hat{t}$ is polynomial. □