# Notation3 as an Existential Rule Language

Dörthe Arndt[1] and Stephan Mennicke[2]

[1] Computational Logic Group, Technische Universität Dresden, Dresden, Germany
[2] Knowledge-Based Systems Group, Technische Universität Dresden, Dresden, Germany
`{firstname.lastname}@tu-dresden.de`

**Abstract.** Notation3 Logic (N3) is an extension of RDF which allows the user to write rules introducing new blank nodes to RDF graphs. Many applications (e.g., ontology mapping) rely on this feature as blank nodes – used directly or in auxiliary constructs – are omnipresent on the Web. However, the number of fast N3 reasoners fully supporting blank node introduction is rather limited. On the other hand, there are engines like VLog or Nemo not directly supporting Semantic Web rule formats but developed for very similar constructs: existential rules. In this paper we investigate the relation between N3 rules with blank nodes in their heads and existential rules. We identify a subset of N3 which can be mapped directly to existential rules and define such a mapping preserving the equivalence of N3 formulae. To also illustrate that N3 reasoning could benefit from our translation, we employ this mapping in an implementation to compare the performance of the N3 reasoners EYE and cwm to VLog and Nemo on N3 rules and their mapped counterparts. Our tests show that the existential rule reasoners perform particularly well for use cases containing many facts while the EYE reasoner is very fast when dealing with a high number of dependent rules. We thus provide a tool enabling the Semantic Web community to directly use existing and future existential rule reasoners and benefit from the findings of this active community.

**Keywords:** Notation3 · RDF · Blank Nodes · Existential rules.

## 1 Introduction

Notation3 Logic (N3) [28,9] is an extension of the Resource Description Framework (RDF) which allows the user to quote graphs, to express rules, and to apply built-in functions on the components of RDF triples. Facilitated by reasoners like cwm [7], Data-Fu [21], or EYE [27], N3 rules directly consume and produce RDF graphs. This makes N3 well-suited for rule exchange on the Web. N3 supports the introduction of new blank nodes through rules, that is, if a blank node appears in the head[3] of a rule, each new match for the rule body produces a new instance of the rule's head containing *fresh* blank nodes. This feature is interesting for many use cases – mappings between different vocabularies include blank nodes, workflow composition deals with unknown existing instances [26] – but it also impedes reasoning tasks: from a logical point of view

[3] To stay consistent across frameworks, we use the terms *head* and *body* throughout the whole paper. The head is the part of the rule occurring at the end of the implication arrow, the body the part at its beginning (backward rules: "head $\leftarrow$ body", forward rules: "body $\rightarrow$ head").

these rules contain existentially quantified variables in their heads. Reasoning with such rules is known to be undecidable in general and very complex on decidable cases [5,25].

Even though recent projects like jen3[4] or RoXi [12] aim at improving the situation, the number of fast N3 reasoners fully supporting blank node introduction is low. This is different for reasoners acting on existential rules, a concept very similar to blank-node-producing rules in N3, but developed for databases. Sometimes it is necessary to uniquely identify data by a value that is not already part of the target database. One tool to achieve that are *labeled nulls* which – just as blank nodes – indicate *the existence* of a value. This problem from databases and the observation that rules may provide a powerful, yet declarative, means of computing has led to more extensive studies of existential rules [5,14]. Many reasoners like for example VLog [15] or Nemo [23] apply dedicated strategies to optimize reasoning with existential rules.

This paper aims to make existing and future optimizations on existential rules usable in the Semantic Web. We introduce a subset of N3 supporting existential quantification but ignoring features of the language not covered in existential rules, like for example built-in functions or lists. We provide a mapping between this logic and existential rules: The mapping and its inverse both preserve equivalences of formulae, enabling N3 reasoning via existential rule technologies. We implement this mapping in python and compare the reasoning performance of the existential rule reasoners Vlog and Nemo, and the N3 reasoners EYE and cwm for two benchmarks: one applying a fixed set of rules on a varying size of facts, and one applying a varying set of highly dependent rules to a fixed set of facts. In our tests VLog and Nemo together with our mapping outperform the traditional N3 reasoners EYE and cwm when dealing with a high number of facts while EYE is the fastest on large dependent rule sets. This is a strong indication that our implementation will be of practical use when extended by further features.

We motivate our approach by providing examples of N3 and existential rule formulae, and discuss how these are connected, in Sect. 2. In Sect. 3 we provide a more formal definition of Existential N3 ($N3^{\exists}$), introduce its semantics and discuss its properties. We then formally introduce existential rules, provide the mapping from $N3^{\exists}$ into this logic, and prove its truth-preserving properties in Sect. 4. Sect. 5 discusses our implementation and provides an evaluation of the different reasoners. In Sect. 6 we discuss the related work to then conclude our paper with Sect. 7. We outsourced all formal proofs to a technical appendix [3]. Furthermore, the code needed for reproducing our experiments is available on GitHub (`https://github.com/smennicke/n32rules`).

## 2 Motivation

N3 has been inroduced as a rule-based extension of RDF. As in RDF, N3 knowledge is stated in triples consisting of *subject*, *predicate*, and *object*. In ground triples these can either be Internationalized Resource Identifiers (IRIs) or literals. The expression

```
:lucy :knows :tom.                                                    (1)
```

means[5] that *"lucy knows tom"*. Sets of triples are interpreted as their conjunction. Like RDF, N3 supports blank nodes, usually starting with `_:`, which stand for (implicitly)

[4] `https://github.com/william-vw/jen3`

[5] We omit name spaces for brevity.

existentially quantified variables. The statement

```
:lucy :knows _:x.                                                  (2)
```

means *"there exists someone who is known by lucy"*. N3 furthermore supports implicitly universally quantified variables, indicated by a leading question mark (`?`), and implications which are stated using graphs, i.e., sets of triples, surrounded by curly braces (`{}`) as body and head connected via an arrow (`=>`). The formula

```
{:lucy :knows ?x}=>{?x :knows :lucy}.                              (3)
```

means that *"everyone known by Lucy also knows her"*. Furthermore, N3 allows the use of blank nodes in rules. These blank nodes are not quantified outside the rule like the universal variables, but in the rule part they occur in, that is either in its body or its head.

```
{?x :knows :tom}=>{?x :knows _:y. _:y :name "Tom"}.                (4)
```

means *"everyone knowing Tom knows* someone *whose name is* Tom*"*.

This last example shows, that N3 supports rules concluding the *existence* of certain terms which makes it easy to express them as *existential rules*. An existential rule is a first-order sentence of the form

$$\forall \mathbf{x}, \mathbf{y}.\, \varphi[\mathbf{x}, \mathbf{y}] \to \exists \mathbf{z}.\, \psi[\mathbf{y}, \mathbf{z}] \tag{5}$$

where $\mathbf{x}, \mathbf{y}, \mathbf{z}$ are mutually disjoint lists of variables, $\varphi$ and $\psi$ are conjunctions of atoms using only variables from the given lists, and $\varphi$ is referred to as the *body* of the rule while $\psi$ is called the *head*. Using the basic syntactic shape of (5) we go through all the example N3 formulae (1)–(4) again and represent them as existential rules. To allow for the full flexibility of N3 and RDF triples, we translate each RDF triple, just like the one in (1) into a first-order atom $tr(\texttt{:lucy}, \texttt{:knows}, \texttt{:tom})$. Here, $tr$ is a ternary predicate holding subject, predicate, and object of a given RDF triple. This standard translation makes triple predicates (e.g., `:knows`) accessible as terms. First-order atoms are also known as *facts*, finite sets of facts are called *databases*, and (possibly infinite) sets of facts are called *instances*. Existential rules are evaluated over instances (cf. Sect. 4).

Compared to other rule languages, the distinguishing feature of existential rules is the use of existentially quantified variables in the head of rules (cf. $\mathbf{z}$ in (5)). The N3 formula in (2) contains an existentially quantified variable and can, thus, be encoded as

$$\to \exists x.\, tr(\texttt{:lucy}, \texttt{:knows}, x) \tag{6}$$

Rule (6) has an empty body, which means the head is unconditionally true. Rule (6) is satisfied on instances containing any fact $tr(\texttt{:lucy}, \texttt{:knows}, \_)$ (e.g., $tr(\texttt{:lucy}, \texttt{:knows}, \texttt{:tim})$ so that variable $x$ can be bound to `:tim`).

The implication of (3) has

$$\forall x.\, tr(\texttt{:lucy}, \texttt{:knows}, x) \to tr(x, \texttt{:knows}, \texttt{:lucy}) \tag{7}$$

as its (existential) rule counterpart, which does not contain any existentially quantified variables. Rule (7) is satisfied in the instance

$$\mathcal{I}_1 = \{tr(\texttt{:lucy}, \texttt{:knows}, \texttt{:tom}), tr(\texttt{:tom}, \texttt{:knows}, \texttt{:lucy})\}$$

but not in

$$\mathcal{K}_1 = \{tr(\texttt{:lucy}, \texttt{:knows}, \texttt{:tom})\}$$

since the only fact in $\mathcal{K}_1$ matches the body of the rule, but there is no fact reflecting on its (instantiated) head (i.e., the required fact $tr(\texttt{:tom}, \texttt{:knows}, \texttt{:lucy})$ is missing). Ultimately, the implication (4) with blank nodes in its head may be transferred to a rule with an existential quantifier in the head:

$$\forall x.\ tr(x, \texttt{:knows}, \texttt{:tom}) \to \exists y.\ tr(x, \texttt{:knows}, y) \wedge tr(y, \texttt{:name}, \texttt{"Tom"}). \qquad (8)$$

It is clear that rule (8) is satisfied in instance

$$\mathcal{I}_2 = \{tr(\texttt{:lucy}, \texttt{:knows}, \texttt{:tom}), tr(\texttt{:tom}, \texttt{:name}, \texttt{"Tom"})\}.$$

However, instance $\mathcal{K}_1$ does not satisfy rule (8) because although the only fact satisfies the rule's body, there are no facts jointly satisfying the rule's head.

Note, for query answering over databases and rules, it is usually not required to decide for a concrete value of $y$ (in rule (8)). Many implementations, therefore, use some form of abstraction: for instance, Skolem terms. VLog and Nemo implement the *standard chase* which uses another set of terms, so-called *labeled nulls*. Instead of injecting arbitrary constants for existentially quantified variables, (globally) fresh nulls are inserted in the positions existentially quantified variables occur. Such a labeled null embodies the existence of a constant on the level of instances (just like blank nodes in RDF graphs). Let $n$ be such a labeled null. Then $\mathcal{I}_2$ can be generalized to

$$\mathcal{I}_3 = \{tr(\texttt{:lucy}, \texttt{:knows}, \texttt{:tom}), tr(\texttt{:lucy}, \texttt{:knows}, n), tr(n, \texttt{:name}, \texttt{"Tom"})\},$$

on which rule (8) is satisfied, binding null $n$ to variable $y$. $\mathcal{I}_3$ is, in fact, more general than $\mathcal{I}_2$ by the following observation: There is a mapping from $\mathcal{I}_3$ to $\mathcal{I}_2$ that is a homomorphism (see Sect. 4.1 for a formal introduction) but not vice versa. The homomorphism here maps the null $n$ (from $\mathcal{I}_3$) to the constant `:tom` (in $\mathcal{I}_2$). Intuitively, the existence of a query answer (for a conjunctive query) on $\mathcal{I}_3$ implies the existence of a query answer on $\mathcal{I}_2$. Existential rule reasoners implementing some form of *the chase* aim at finding the most general instances (*universal models*) in this respect [18].

In the remainder of this paper, we further analyze the relation between N3 and existential rules. First, we give a brief formal account of the two languages and then provide a correct translation function from N3 to existential rules.

## 3 Existential N3

In the previous section we introduced essential elements of N3, namely triples and rules. N3 also supports more complex constructs like lists, nesting of rules, and quotation. As these features are not covered by existential rules, we define a subset of N3 excluding them, called *existential N3* ($N3^{\exists}$).[6] We base our definitions on so-called *simple N3 formulae* [4, Chapter 7], these are N3 formulae which do not allow for nesting.

### 3.1 Syntax

$N3^{\exists}$ relies on the RDF alphabet. As the distinction is not relevant in our context, we consider IRIs and literals together as constants. Let $C$ be a set of such constants, $U$ a set of universal variables (starting with `?`), and $E$ a set of existential variables (i.e., blank nodes). If the sets $C$, $U$, $E$, and $\{\texttt{\{}, \texttt{\}}, \texttt{=>}, \texttt{.}\}$ are mutually disjoint, we call $\mathfrak{A} := C \cup U \cup E \cup \{\texttt{\{}, \texttt{\}}, \texttt{=>}, \texttt{.}\}$ an *N3 alphabet*. Fig. 1 provides the syntax of $N3^{\exists}$ over $\mathfrak{A}$.

[6] This fragment is expressive enough to support basic use cases like user-defined ontology mapping. Here it is important to note that RDF lists can be expressed using first-rest pairs.

| | | | | | |
|---|---|---|---|---|---|
| `f ::=` | | formulae: | `t ::=` | | terms: |
| | `t t t.` | atomic formula | | `ex` | existential variables |
| | `{e}=>{e}.` | implication | | `c` | constants |
| | `f f` | conjunction | | | |
| `n ::=` | | N3 terms: | `e ::=` | | expressions: |
| | `uv` | universal variables | | `n n n.` | triple expression |
| | `t` | terms | | `e e` | conjunction expression |

**Fig. 1.** Syntax of N3$^{\exists}$

$N3^{\exists}$ fully covers RDF – RDF formulae are conjunctions of atomic formulae – but allows literals and blank nodes to occur in subject, predicate, and object position. On top of the triples, it supports rules containing existential and universal variables. Note, that the syntax allows rules having new universal variables in their head like for example

```
{:lucy :knows :tom}=>{?x :is :happy}.                (9)
```

which results in a rule expressing *"if lucy knows tom, everyone is happy"*. This implication is problematic: Applied on triple (1), it yields `?x :is :happy.` which is a triple containing a universal variable. Such triples are not covered by our syntax, the rule thus introduces a fact we cannot express. Therefore, we restrict $N3^{\exists}$ rules to *well-formed implications* which rely on *components*. A *component* of a formula or an expression is an N3 term which does not occur nested in a rule. More formally, let $f$ be a formula or an expression over an alphabet $\mathfrak{A}$. The set $comp(f)$ of *components* of $f$ is defined as:

– If $f$ is an atomic formula or a triple expression of the form $t_1$ $t_2$ $t_3$., $comp(f) = \{t_1, t_2, t_3\}$.
– If $f$ is an implication of the form $\{e_1\}$=>$\{e_2\}$., then $comp(f) = \{\{e_1\}, \{e_2\}\}$.
– If $f$ is a conjunction of the form $f_1 f_2$, then $comp(f) = comp(f_1) \cup comp(f_2)$.

A rule $\{\mathtt{e}_1\}$=>$\{\mathtt{e}_2\}$. is called *well-formed* if $(comp(\mathtt{e}_2) \setminus comp(\mathtt{e}_1)) \cap U = \emptyset$. For the remainder of this paper we assume all implications to be well-formed.

## 3.2 Semantics

In order to define the semantics of $N3^{\exists}$ we first note, that in our fragment of N3 all quantification of variables is only defined implicitly. The blank node in triple (2) is understood as an existentially quantified variable, the universal in formula (3) as universally quantified. Universal quantification spans over the whole formula – variable `?x` occurring in body and head of rule (3) is universally quantified for the whole implication – while existential quantification is local – the conjunction in the head of rule (4) is existentially quantified there. Adding new triples as conjuncts to formula (4) like

```
:lucy :knows _:y. _:y :likes :cake.                (10)
```

leads to the new statement that *"lucy knows someone who likes cake"* but even though we are using the same blank node identifier `_:y` in both formulae, the quantification of the variables in this formula is totally seperated and the person named "Tom" is not necessarily related to the cake-liker. With the goal to deal with this locality of blank node scoping, we define substitutions which are only applied on components of formulae and leave nested elements like for example the body and head of rule (3) untouched.

A *substitution* $\sigma$ is a mapping from a set of variables $X \subset U \cup E$ to the set of N3 terms. We *apply* $\sigma$ to a term, formula or expression $x$ as follows:

- $x\sigma = \sigma(x)$ if $x \in X$,
- $(s\ p\ o)\sigma = (s\sigma)(p\sigma)(o\sigma)$ if $x = s\ p\ o$ is an atomic formula or a triple expression,
- $(f_1 f_2)\sigma = (f_1\sigma)(f_2\sigma)$ if $x = f_1 f_2$ is a conjunction,
- $x\sigma = x$ else.

For formula $f =$ `_:x :p :o. {_:x :b :c}=>{_:x :d :e}.`, substitution $\sigma$ and `_:x` $\in \mathrm{dom}(\sigma)$, we get: $f\sigma = \sigma($`_:x`$)$`:p :o. {_:x :b :c}=>{_:x :d :e}.`[7] We use the substitution to define the semantics of $N3^{\exists}$ which additionally makes use of *N3 interpretations* $\mathfrak{I} = (\mathfrak{D}, \mathfrak{a}, \mathfrak{p})$ consisting of (1) the domain of $\mathfrak{I}$, $\mathfrak{D}$; (2) $\mathfrak{a} : C \to \mathfrak{D}$ called the object function; (3) $\mathfrak{p} : \mathfrak{D} \to 2^{\mathfrak{D}\times\mathfrak{D}}$ called the predicate function.

Just as the function IEXT in RDF's simple interpretations [22], N3's predicate function maps elements from the domain of discourse to a set of pairs of domain elements and is not applied on relation symbols directly. This makes quantification over predicates possible while not exceeding first-order logic in terms of complexity. To introduce the *semantics of* $N3^{\exists}$, let $\mathfrak{I} = (\mathfrak{D}, \mathfrak{a}, \mathfrak{p})$ be an N3 interpretation. For an $N3^{\exists}$ formula $f$:

1. If $W = \mathrm{comp}(f) \cap E \neq \emptyset$, then $\mathfrak{I} \models f$ iff $\mathfrak{I} \models f\mu$ for some substitution $\mu : W \to C$.
2. If $\mathrm{comp}(f) \cap E = \emptyset$:
   (a) If $f$ is an atomic formula $t_1\, t_2\, t_3$, then $\mathfrak{I} \models t_1\, t_2\, t_3.$ iff $(\mathfrak{a}(t_1), \mathfrak{a}(t_3)) \in \mathfrak{p}(\mathfrak{a}(t_2))$.
   (b) If $f$ is a conjunction $f_1 f_2$, then $\mathfrak{I} \models f_1 f_2$ iff $\mathfrak{I} \models f_1$ and $\mathfrak{I} \models f_2$.
   (c) If $f$ is an implication, then $\mathfrak{I} \models$ `{`$e_1$`}=>{`$e_2$`}` iff $\mathfrak{I} \models e_2\sigma$ if $\mathfrak{I} \models e_1\sigma$ for all substitutions $\sigma$ on the universal variables $\mathrm{comp}(\mathsf{e}_1) \cap U$ by constants.

The semantics as defined above uses a substitution into the set of constants instead of a direct assignment to the domain of discourse to interpret quantified variables. This design choice inherited from N3 ensures referential opacity of quoted graphs and means, in essence, that quantification always refers to named domain elements.

With that semantics, we call an interpretation $\mathfrak{M}$ *model* of a dataset $\Phi$, written as $\mathfrak{M} \models \Phi$, if $\mathfrak{M} \models f$ for each formula $f \in \Phi$. We say that two sets of $N3^{\exists}$ formulae $\Phi$ and $\Psi$ are *equivalent*, written as $\Phi \equiv \Psi$, if for all interpretations $\mathfrak{M}$: $\mathfrak{M} \models \Phi$ iff $\mathfrak{M} \models \Psi$. If $\Phi = \{\phi\}$ and $\Psi = \{\psi\}$ are singleton sets, we write $\phi \equiv \psi$ omitting the brackets.

*Piece Normal Form* $N3^{\exists}$ formulae consist of conjunctions of triples and implications. For our goal of translating such formulae to existential rules, it is convenient to consider sub-formulae seperately. Below, we therefore define the so-called *Piece Normal Form* (PNF) for $N3^{\exists}$ formulae and show that each such formula $f$ is equivalent to a set of sub-formulae $\Phi$ (i.e., $\Phi \equiv f$) in PNF. We proceed in two steps.

First, we separate formulae based on their blank node components. If two parts of a conjunction share a blank node component, as in formula (10), we cannot split the

[7] Note that the semantics of *simple formulae* on which $N3^{\exists}$'s semantics is based, relies on two ways to apply a substitution which is necessary to handle nested rules, since such constructs are excluded in $N3^{\exists}$, we simplified here.

formula into two since the information about the co-reference would get lost. However, if conjuncts either do not contain blank nodes or only contain disjoint sets of these, we can split them into so-called *pieces*: Two formulae $f_1$ and $f_2$ are called *pieces* of a formula $f$ if $f = f_1 f_2$ and $\mathrm{comp}(f_1) \cap \mathrm{comp}(f_2) \cap E = \emptyset$. For such formulae we know:

**Lemma 1 (Pieces).** *Let $f = f_1 f_2$ be an $N3^{\exists}$ conjunction and let* $\mathrm{comp}(f_1) \cap \mathrm{comp}(f_2) \cap E = \emptyset$*, then for each interpretation $\mathfrak{I}$, $\mathfrak{I} \models f$ iff $\mathfrak{I} \models f_1$ and $\mathfrak{I} \models f_2$.*

If we recursively divide all pieces into sub-pieces, we get a maximal set $F = \{f_1, f_2, \ldots, f_n\}$ for each formula $f$ such that $F \equiv \{f\}$ and for all $1 \leq i, j \leq n$, $\mathrm{comp}(f_i) \cap \mathrm{comp}(f_j) \cap E \neq \emptyset$ implies $i = j$.

Second, we replace all blank nodes occurring in rule bodies by *fresh* universals. The rule `{_:x :likes :cake}=>{:cake :is :good}.` becomes `{?y :likes :cake}=>{:cake :is :good}.` Note that both rules have the same meaning, namely *"if someone likes cake, then cake is good."*. We generalize that:

**Lemma 2 (Eliminating Existentials).** *Let $f =$ `{`$e_1$`}=>{`$e_2$`}` and $g =$ `{`$e'_1$`}=>{`$e_2$`}` be $N3^{\exists}$ implications such that $e'_1 = e_1\sigma$ for some injective substitution $\sigma : \mathrm{comp}(e_1) \cap E \to U \setminus \mathrm{comp}(e_1)$ of the existential variables of $e_1$ by universals. Then: $f \equiv g$*

For a rule $f$ we call the formula $f'$ in which all existentials occurring in its body are replaced by universals following Lemma 2 the *normalized* version of the rule. We call an $N3^{\exists}$ formula $f$ *normalized*, if all rules occurring in it as conjuncts are normalized. This allows us to introduce the *Piece Normal Form*:

**Theorem 1 (Piece Normal Form).** *For every well-formed $N3^{\exists}$ formula $f$, there exists a set $F = \{f_1, f_2, \ldots, f_k\}$ of $N3^{\exists}$ formulae such that $F \equiv \{f\}$ and $F$ is in* piece normal form *(PNF). That is, all $f_i \in F$ are normalized formulae and $k \in \mathbb{N}$ is the maximal number such that for $1 \leq i, j \leq k$, $comp(f_i) \cap comp(f_j) \cap E \neq \emptyset$ implies $i = j$. If $f_i$ ($1 \leq i \leq k$) is a conjunction of atomic formulae, we call $f_i$ an* atomic piece.

Since the piece normal form $F$ of $N3^{\exists}$ formula $f$ is obtained by only replacing variables and separating conjuncts of $f$ into the set form, the overall size of $F$ is linear in $f$.

## 4 From N3 to Existential Rules

Without loss of generality, we translate sets $F$ of $N3^{\exists}$ formulae in PNF (cf. Theorem 1) to sets of existential rules $\mathcal{T}(F)$. As a preliminary step, we introduce the language of existential rules formally. Later on, we explain and define the translation function that has already been sketched in Sect. 2. The section closes with a correctness argument, establishing a strong relationship between existential rules and $N3^{\exists}$.

### 4.1 Foundations of Existential Rule Reasoning

For existential rules, we also consider a first-order vocabulary, consisting of constants ($\mathbf{C}$) and variables ($\mathbf{V}$), and additionally so-called (labeled) nulls ($\mathbf{N}$)[8]. As already mentioned in Sect. 2, we use the same set of constants as N3 formulae, meaning $\mathbf{C} = C$.

[8] We choose here different symbols to disambiguate between existential rules and N3, although vocabularies partially overlap.

Furthermore, let $\mathbf{P}$ be a set of *relation names*, where each $p \in \mathbf{P}$ comes with an arity $ar(p) \in \mathbb{N}$. $\mathbf{C}$, $\mathbf{V}$, $\mathbf{N}$, and $\mathbf{P}$ are countably infinite and pair-wise disjoint. We use the ternary relation name $tr \in \mathbf{P}$ to encode N3 triples in Sect. 2. If $p \in \mathbf{P}$ and $t_1, t_2, \ldots, t_{ar(p)}$ is a list of terms (i.e., $t_i \in \mathbf{C} \cup \mathbf{N} \cup \mathbf{V}$), $p(t_1, t_2, \ldots, t_{ar(p)})$ is called an *atom*. We often use $\mathbf{t}$ to summarize a term list like $t_1, \ldots, t_n$ ($n \in \mathbb{N}$), and treat it as a set whenever order is irrelevant. An atom $p(\mathbf{t})$ is *ground* if $\mathbf{t} \subseteq \mathbf{C}$. An *instance* is a (possibly infinite) set $\mathcal{I}$ of variable-free atoms and a finite set of ground atoms $\mathcal{D}$ is called a *database*.

For a set of atoms $\mathcal{A}$ and an instance $\mathcal{I}$, we call a function $h$ from the terms occurring in $\mathcal{A}$ to the terms in $\mathcal{I}$ a *homomorphism from* $\mathcal{A}$ *to* $\mathcal{I}$, denoted by $h : \mathcal{A} \to \mathcal{I}$, if (1) $h(c) = c$ for all $c \in \mathbf{C}$ (occurring in $\mathcal{A}$), and (2) $p(\mathbf{t}) \in \mathcal{A}$ implies $p(h(\mathbf{t})) \in \mathcal{I}$. If any homomorphism from $\mathcal{A}$ to $\mathcal{I}$ exists, write $\mathcal{A} \to \mathcal{I}$. Please note that if $n$ is a null occurring in $\mathcal{A}$, then $h(n)$ may be a constant or null.

For an *(existential) rule* $r\colon \forall \mathbf{x}, \mathbf{y}.\ \varphi[\mathbf{x}, \mathbf{y}] \to \exists \mathbf{z}.\ \psi[\mathbf{y}, \mathbf{z}]$ (cf. (5)), rule bodies ($\mathsf{body}(r)$) and heads ($\mathsf{head}(r)$) will also be considered as sets of atoms for a more compact representation of the semantics. Let $r$ be a rule and $\mathcal{I}$ an instance. We call a homomorphism $h : \mathsf{body}(r) \to \mathcal{I}$ a *match for* $r$ *in* $\mathcal{I}$. A match $h$ is *satisfied for* $r$ *in* $\mathcal{I}$ if there is an extension $h^\star$ of $h$ (i.e., $h \subseteq h^\star$) such that $h^\star(\mathsf{head}(r)) \subseteq \mathcal{I}$. If all matches of $r$ are satisfied in $\mathcal{I}$, we say that $r$ is satisfied in $\mathcal{I}$, denoted by $\mathcal{I} \models r$. For a rule set $\Sigma$ and database $\mathcal{D}$, we call an instance $\mathcal{I}$ a *model of* $\Sigma$ *and* $\mathcal{D}$, denoted by $\mathcal{I} \models \Sigma, \mathcal{D}$, if $\mathcal{D} \subseteq \mathcal{I}$ and $\mathcal{I} \models r$ for each $r \in \Sigma$. We say that two rule sets $\Sigma_1$ and $\Sigma_2$ are *equivalent*, denoted $\Sigma_1 \leftrightarrows \Sigma_2$, iff for all instances $\mathcal{I}$, $\mathcal{I} \models \Sigma_1$ iff $\mathcal{I} \models \Sigma_2$.

Labeled nulls play the role of fresh constants without further specification, just like blank nodes in RDF or N3. The chase is a family of algorithms that soundly produces models of rule sets by continuously applying rules for unsatisfied matches. If some rule head is instantiated, existential variables are replaced by fresh nulls in order to facilitate for arbitrary constants. Although the chase is not guaranteed to terminate, it always produces a (possibly infinite) model[9] [18].

## 4.2 The Translation Function from N3 to Existential Rules

The translation function $\mathcal{T}$ maps sets $F = \{f_1, \ldots, f_k\}$ of $N3^\exists$ formulae in PNF to sets of rules $\Sigma$. Before we go into the details of the translation for every type of piece, we consider an auxiliary function $\mathbb{T} : C \cup E \cup U \to \mathbf{C} \cup \mathbf{V}$ mapping N3 terms to terms in our rule language (cf. previous subsection):

$$\mathbb{T}(t) := \begin{cases} v^\forall_{\mathtt{x}} & \text{if } t = ?\mathtt{x} \in U \\ v^\exists_{\mathtt{y}} & \text{if } t = \_:\mathtt{y} \in E \\ t & \text{if } t \in C, \end{cases}$$

where $v^\forall_{\mathtt{x}}, v^\exists_{\mathtt{y}} \in \mathbf{V}$ and $t \in \mathbf{C}$ (i.e., we assume $C \subseteq \mathbf{C}$). While variables in N3 belong to either $E$ or $U$, this separation is lost under function $\mathbb{T}$. For enhancing readability of subsequent examples, the identity of the variable preserves this information by using superscripts $\exists$ and $\forall$. We provide the translation for every piece $f_i \in F$ ($1 \leq i \leq k$) and later collect the full translation of $F$ as the union of its translated pieces.

[9] Not just any model, but a universal model, which is a model that has a homomorphism to any other model of the database and rule set. Up to homomorphisms, universal models are the smallest among all models.

*Translating Atomic Pieces.* If $f_i$ is an atomic piece, it has the form $f_i = g_1\ \ g_2\ \ \ldots\ \ g_l$ for some $l \geq 1$ and each $g_j$ ($1 \leq j \leq l$) is an atomic formula. The translation of $f_i$ is the singleton set $\mathcal{T}(f_i) = \{\to \exists \mathbf{z}.\ tr(\mathbb{T}(g_1)) \wedge tr(\mathbb{T}(g_2)) \wedge \ldots \wedge tr(\mathbb{T}(g_l))\}$, where $\mathbb{T}(g_j) = \mathbb{T}(t_j^1), \mathbb{T}(t_j^2), \mathbb{T}(t_j^3)$ if $g_j = t_j^1\ \ t_j^2\ \ t_j^3$ and $\mathbf{z}$ is the list of translated existential variables (via $\mathbb{T}$) from existentials occurring in $f$. For example, the formula in (10) constitutes a single piece $f_{(10)}$ which translates to a set containing the rule

$$\to \exists v_{\mathtt{y}}^{\exists}.\ tr(\mathtt{:lucy}, \mathtt{:knows}, v_{\mathtt{y}}^{\exists}) \wedge tr(v_{\mathtt{y}}^{\exists}, \mathtt{:likes}, \mathtt{:cake}).$$

*Translating Rules.* For $f_i$ being a rule `{`$e_1$`}=>{`$e_2$`}` we also obtain a single rule. Recall that the PNF ensures all variables of $e_1$ to be universals and all universal variables of $e_2$ to also occur in $e_1$. If $e_1 = g_1^1\ \ g_1^2\ \ \cdots\ \ g_1^m$ and $e_2 = g_2^1\ \ g_2^2\ \ \cdots\ \ g_2^n$, $\mathcal{T}(f_i) = \{\forall \mathbf{x}.\ \bigwedge_{j=1}^{m} tr(\mathbb{T}(g_1^j)) \to \exists \mathbf{z}.\ \bigwedge_{j=1}^{n} tr(\mathbb{T}(g_2^j))\}$ where $\mathbf{x}$ and $\mathbf{z}$ are the lists of translated universals and existentials, respectively. Applying the translation to the N3 formula in (4), which is a piece according to our definitions, we obtain again a singleton set, now containing the rule

$$\forall v_{\mathtt{x}}^{\forall}.\ tr(v_{\mathtt{x}}^{\forall}, \mathtt{:knows}, \mathtt{:tom}) \to \exists v_{\mathtt{y}}^{\exists}.\ tr(v_{\mathtt{x}}^{\forall}, \mathtt{:knows}, v_{\mathtt{y}}^{\exists}) \wedge tr(v_{\mathtt{y}}^{\exists}, \mathtt{:name}, \mathtt{"Tom"}),$$

which is the same rule as (8) up to a renaming of (bound) variables ($\alpha$-conversion [19]).

*Translating Sets.* For the set $F = \{f_1, f_2, \ldots, f_k\}$ of $N3^{\exists}$ formulae in PNF, $\mathcal{T}(F)$ is the union of all translated constituents (i.e., $\mathcal{T}(F) = \bigcup_{i=1}^{k} \mathcal{T}(f_i)$). Please note that $\mathcal{T}$ does not exceed a polynomial overhead of its input.

The correctness argument for $\mathcal{T}$ splits into *soundness* – whenever we translate two equivalent $N3^{\exists}$ formulae, their translated rules turn out to be equivalent as well – and *completeness* – formulae that are not equivalent are translated to rule sets that are not equivalent. Although the different formalisms have quite different notions of models, models of a translated rule sets $\mathcal{M}$ can be converted into models of the original N3 formula by using a Herbrand argument. The full technical lemma and the construction is part of our technical appendix [3]. Our correctness proof also considers completeness since, otherwise, a more trivial translation function would have sufficed: Let $\mathcal{T}_0$ be a function mapping all $N3^{\exists}$ formulae to the empty rule set: All equivalent $N3^{\exists}$ formulae are mapped to equivalent rule sets (always $\emptyset$), but also formulae that are not equivalent yield equivalent rule sets under $\mathcal{T}_0$. Having such a strong relationship between N3 and existential rules allows us to soundly use the translation function $\mathcal{T}$ in practice.

**Theorem 2.** *For PNFs F and G of $N3^{\exists}$ formulae, $F \equiv G$ iff $\mathcal{T}(F) \leftrightarrows \mathcal{T}(G)$.*

Beyond the correctness of $\mathcal{T}$, we have no further guarantees. As $N3^{\exists}$ reasoning does not necessarily stop, there is no requirement for termination of the chase over translated rule sets. We expect that the similarity between $N3^{\exists}$ and existential rules allows for the adoption of sufficient conditions for finite models (e.g., by means of acyclicity [17]).

## 5 Evaluation

The considerations provided above allow us to use existential rule reasoners to perform $N3^{\exists}$ reasoning. We would like to find out whether our finding is of practical relevance, that is whether we can identify datasets on which existential rule reasoners, running on the rule translations, outperform classical N3 reasoners provided with the original data.

In order to do this we have implemented $\mathcal{T}$ as a python script that takes an arbitrary $N3^{\exists}$ formula $f$, constructs its set representation $F$ in PNF, and produces the set of rules $\mathcal{T}(F)$. This script and some additional scripts to translate existential rules (with at most binary predicates) to $N3^{\exists}$ formulae are available on GitHub. Our implementation allows us to compare N3 reasoners with existential rule reasoners, performance-wise. As existential rule reasoners we chose VLog [15], a state-of-the-art reasoning engine designed for working with large piles of input data, and Nemo [23], a recently released rust-based reasoning engine. As N3 reasoners we chose cwm [7] and EYE [27] which – due to their good coverage of N3 features – are most commonly used. All experiments have been performed on a laptop with 11th Gen Intel Core i7-1165G7 CPU, 32GB of RAM, and 1TB disk capacity, running a Ubuntu 22.04 LTS.

## 5.1 Datasets

We performed our experiments on two datasets: LUBM from the *Chasebench* [6] provides a fixed set of 136 rules and varies in the number of facts these rules are applied; the DEEP TAXONOMY (DT) benchmark developed for the *WellnessRules* project [11] consists of one single fact and a varying number of mutually dependent rules.

The *Chasebench* is a benchmarking suite for existential rule reasoning. Among the different scenaria in Chasebench we picked LUBM for its direct compatibility with N3: all predicates in LUBM have at most arity 2. Furthermore, LUBM allows for a glimpse on scalability since LUBM comes in different database sizes. We have worked with LUBM 001, 010, and 100, roughly referring to dataset sizes of a hundred thousand, one million and ten million facts. We translated LUBM data and rules into a canonical N3 format. Predicate names and constants within the dataset become IRIs using the example prefix. An atom like *src_advisor*(*Student441*,*Professor8*) becomes the triple `:Student441 :src_advisor :Professor8.`. For atoms using unary predicates, like *TeachingAssistent*(*Student498*), we treat `:TeachingAssistent` as a class and relate `:Student498` via `rdf:type` to the class. For any atom $A$, we denote its canonical translation into triple format by $t(A)$. Note this canonical translation only applies to atoms of unary and binary predicates. For the existential rule

$$\forall \mathbf{x}.\, B_1 \wedge \ldots \wedge B_m \rightarrow \exists \mathbf{z}.\, H_1 \wedge \ldots \wedge H_n$$

we obtain the canonical translation by applying $t$ to all atoms, respecting universally and existentially quantified variables (i.e., universally quantified variables are translated to universal N3 variables and existentially quantified variables become blank nodes):

$$\texttt{\{}t(B_1)\texttt{.} \cdots t(B_m)\texttt{.\}=>\{}t(H_1)\texttt{.} \cdots t(H_n)\texttt{.\}.}$$

All N3 reasoners have reasoned over the canonical translation of data and rules which was necessary because of the lack of an N3 version of LUBM. Since we are evaluating VLog's and Nemo's performance on our translation $\mathcal{T}$, we converted the translated LUBM by $\mathcal{T}$ back to existential rules before reasoning. Thereby, former unary and binary atoms were turned into triples and then uniformly translated to *tr*-atoms via $\mathcal{T}$.

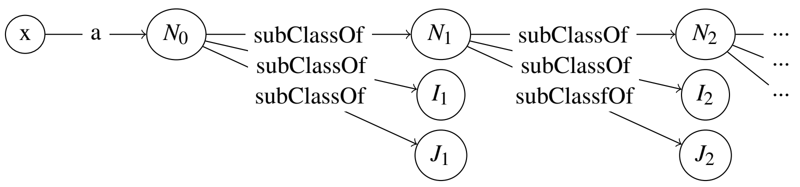

**Fig. 2.** Structure of the DEEP TAXONOMY benchmark.

The *Deep Taxonomy benchmark* simulates deeply nested RDFS-subclass reasoning[10]. It contains one individual which is member of a class. This class is subclass of three other classes of which one again is subclass of three more classes and so on. Figure 2 illustrates this idea. The benchmark provides different depths for the subclass chain and we tested with the depths of 1,000 and 100,000. The reasoning tests for the membership of the individual in the last class of the chain. For our tests, the subclass declarations were translated to rules, the triple `:N0 rdfs:subClassOf :N1.` became

```
{ ?x a :N0.}=>{ ?x a :N1.}.
```

This translation also illustrates why this rather simple reasoning case is interesting: we have a use case in which we depend on long chains of rules executed after each other. The reasoner EYE allows the user to decide per rule whether it is applied using forward- or backward-reasoning, at least if the head of the rule does not contain blank nodes. For this dataset, we evaluated full backward- and full forward-reasoning, separately.

### 5.2 Results

Table 1 presents the running times of the four reasoners and additionally gives statistics about the sizes of the given knowledge base (# facts) and the rule set (# rules). For DT we display two reasoning times for EYE, one produced by only forward reasoning (EYE-fw), one for only backward-reasoning (EYE-bw). Note, that for the latter, the reasoner does not produce the full deductive closure of the dataset, but answers a query instead. As LUBM contains rules with blank nodes in their haeds, full backward reasoning was not possible in that case, the table is left blank. EYE performs much better than VLog and Nemo for the experiments with DT. Its reasoning time is off by one order of magnitude. Conversely, VLog and Nemo could reason over all the LUBM datasets while EYE has thrown an exception after having read the input facts. The reasoning times of VLog are additionally significantly lower than the times for EYE. While Nemo shows a similar runtime on DT as VLog, it is slower on LUBM. However, we may be quite optimistic regarding its progress in runtime behavior, as Nemo already shows better running times on the original LUBM datasets. The reasoner cwm is consistently slower than the other three and from LUBM 010 on. All reasoners tried to find the query answers/deductive closures for at least ten minutes (i.e., — in Table 1 indicates a time-out).

[10] N3 available at: `http://eulersharp.sourceforge.net/2009/12dtb/`.

**Table 1.** Experimental Results

| Dataset | # facts | # rules | cwm | EYE-fw | EYE-bw | VLog | Nemo |
|---|---|---|---|---|---|---|---|
| DT 1000 | 1 | 3001 | 180 s | 0.1 s | 0.001 s | 1.6 s | 1.7 s |
| DT 100000 | 1 | 30,001 | — | 0.3 s | 0.003 s | — | — |
| LUBM 001 | 100,543 | 136 | 117.4 s | 3.4 s | | 0.2 s | 2.4 s |
| LUBM 010 | 1,272,575 | 136 | — | 44.8 s | | 4.3 s | 31.2 s |
| LUBM 100 | 13,405,381 | 136 | — | — | | 47.3 s | 362 s |

### 5.3 Discussion

In all our tests we observe a very poor performance of cwm which is not surprising, given that this reasoner has not been updated for some time. The results for EYE, VLog and Nemo are more interesting as they illustrate the different strengths of the reasoners.

For very high numbers of rules compared to the amount of data, EYE performs much better than VLog and Nemo. The good results of 0.1 and 0.3 seconds can even be improved by using backward reasoning. This makes EYE very well-suited for use cases where we need to apply complex rules on datasets of low or medium size. This could be interesting in decentralized set-ups such as policy-based access control for the Solidproject.[11] On the other hand we see that VLog and Nemo perform best when provided with large datasets and lower numbers of rules. This could be useful use cases involving bigger datasets in the Web like Wikidata or DBpedia[12].

From the perspective of this paper, these two findings together show the relevance of our work: we observed big differences between the tools' reasoning times and these differences depended on the use cases. In other words, there are use cases which could benefit from our translation and we thus do not only make the first steps towards having more N3 reasoners available but also broaden the scope of possible N3 applications.

## 6 Related work

When originally proposed as a W3C member submission [8], the formal semantics of N3 was only introduced informally. As a consequence, different systems, using N3, interpreted concepts like nested formulae differently [1]. Since then, the relation of N3 to other Web standards has been studied from a use-case perspective [4] and a W3C Community group has been formed [28], which recently published the semantics of N3 without functions [2]. Even with these definitions, the semantic relation of the logic to other standards, especially outside the Semantics Web, has not been studied thoroughly.

For N3's subset RDF, de Bruijn and Heymans [13] provide a translation to first-order logic and F-Logic using similar embeddings (e.g., a tenary predicate to represent triples) to the ones in this paper, but do not cover rules. Boley [10] supports N3 in his RuleML Knowledge-Interoperation Hub providing a translation of N3 to PSOA RuleML. This can be translated to other logics. But the focus is more on syntax than on semantics.

[11] `https://solidproject.org/`.

[12] `https://www.wikidata.org/` *and* `https://www.dbpedia.org/`

In Description Logics (DL), rewritings in rule-based languages have their own tradition (see, e.g., [16] for a good overview of existing rewritings and their complexity, as well as more references). The goal there is to (1) make state-of-the-art rule reasoners available for DLs and, thereby, (2) use a fragment of a rule language that reflects on the data complexity of the given DL fragment. Also practical tools have been designed to capture certain profiles of the Web Ontology Language (OWL), like the Orel system [24] and, more recently, DaRLing [20]. To the best of our knowledge, a rewriting for N3 as presented in this paper did not exist before. Also, existential rule reasoning engines have not been compared to the existing N3 reasoners.

## 7 Conclusion

In this paper we studied the close relationship between N3 rules supporting blank node production and existential rules. N3 without special features like built-in functions, nesting of rules, or quotation can be directly mapped to existential rules with unary and binary predicates. In order to show that, we defined a mapping between $N3^{\exists}$, N3 without the aforementioned features, and existential rules. We argued that this mapping and its inverse preserve the equivalence and non-equivalence between datasets. This result allows us to trust the reasoning results when applying the mapping in practice, that is, when (1) translating $N3^{\exists}$ to existential rules, (2) reasoning within that framework, and (3) using the inverse mapping to transfer the result back into N3.

We applied that strategy and compared the reasoning times of the N3 reasoners cwm and EYE with the existential rule reasoners VLog and Nemo. The goal of that comparison was to find out whether there are use cases for which N3 reasoning can benefit from the findings on existential rules. We tested the reasoners on two datasets: DT consisting of one single fact and a varying number of mutually dependent rules and LUBM consisting of a fixed number of rules and a varying number of facts. EYE performs better on DT while VLog and Nemo showed their strength on LUBM. We see that as an indication that for use cases of similar nature, that is, reasoning on large numbers of facts, our approach could be used to improve reasoning times. More generally, we see that reasoners differ in their strengths and that by providing the revertible translation between $N3^{\exists}$ and existential rules we increase the number of reasoners (partly) supporting N3 and the range of use cases the logic can support in practice. We see our work as an important step towards fully establishing rule-based reasoning in the Semantic Web.

As many N3 use cases rely on N3's powerful built-in functions and logical features such as support for graph terms, lists and nested rules, future work should include the extension of our translation towards full coverage of N3. Another direction of future work could be to investigate the differences and similarities we found in our evaluation in more detail: while showing differences in their performance, the reasoners produced the exact same result sets (modulo isomorphism) when acting on rules introducing blank nodes. That is, the different reasoning times do not stem from the handling of existentially quantified rule heads but from other optimization techniques. Fully understanding these differences will help the N3 and the existential rule community to further improve their tools. In that context, it would also be interesting to learn if EYE's capability to combine forward and backward reasoning could improve the reasoning times for data sets including existentially quantified rule heads.

We thus hope that our research on existential N3 will spawn further investigations of powerful data-centric features in data-intensive rule reasoning as well as significant progress in tool support towards these features. Ultimately, we envision a Web of data and rule exchange, fully supported by the best tools available as converging efforts of the N3 community, the existential rule reasoning community, and possibly many others.

*Acknowledgements.* This reasearch is partly supported by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) in project number 389792660 (TRR 248, Center for Perspicuous Computing), by Bundesministerium für Bildung und Forschung (BMBF, Federal Ministry of Education and Research) and DAAD (German Academic Exchange Service) in project 57616814 (SECAI, School of Embedded and Composite AI), and by the BMBF under project 13GW0552B (KIMEDS, KI-assistierte Zertifizierung medizintechnischer Software) in the Center for Scalable Data Analytics and Artificial Intelligence (ScaDS.AI),

## References

1. Arndt, D., Schrijvers, T., De Roo, J., Verborgh, R.: Implicit quantification made explicit: How to interpret blank nodes and universal variables in Notation3 Logic. Journal of Web Semantics **58**, 100501 (Oct 2019). `https://doi.org/10.1016/j.websem.2019.04.001`
2. Arndt, D., Champin, P.A.: Notation3 semantics. W3C community group report, W3C (Jul 2023), `https://w3c.github.io/N3/reports/20230703/semantics.html`
3. Arndt, D., Mennicke, S.: Notation3 as an Existential Rule Language. CoRR **abs/2308.07332** (2023)
4. Arndt, Dörthe: Notation3 as the unifying logic for the semantic web. Ph.D. thesis, Ghent University (2019)
5. Baget, J.F., Leclère, M., Mugnier, M.L., Salvat, E.: On rules with existential variables: Walking the decidability line. Artificial Intelligence **175**(9-10), 1620–1654 (2011)
6. Benedikt, M., Konstantinidis, G., Mecca, G., Motik, B., Papotti, P., Santoro, D., Tsamoura, E.: Benchmarking the chase. In: Sallinger, E., den Bussche, J.V., Geerts, F. (eds.) Proc. 36th Symposium on Principles of Database Systems (PODS'17). pp. 37–52. ACM (2017)
7. Berners-Lee, T.: cwm (2000–2009), `http://www.w3.org/2000/10/swap/doc/cwm.html`
8. Berners-Lee, T., Connolly, D.: Notation3 (N3): A readable RDF syntax. W3C Team Submission (Mar 2011), `http://www.w3.org/TeamSubmission/n3/`
9. Berners-Lee, T., Connolly, D., Kagal, L., Scharf, Y., Hendler, J.: N3Logic: A logical framework for the World Wide Web. Theory and Practice of Logic Programming (3), 249–269 (2008). `https://doi.org/10.1017/S1471068407003213`
10. Boley, H.: The ruleml knowledge-interoperation hub. In: Alferes, J.J., Bertossi, L., Governatori, G., Fodor, P., Roman, D. (eds.) Rule Technologies. Research, Tools, and Applications. pp. 19–33. Springer International Publishing, Cham (2016)
11. Boley, H., Osmun, T.M., Craig, B.L.: Wellnessrules: A web 3.0 case study in ruleml-based prolog-n3 profile interoperation. In: Governatori, G., Hall, J., Paschke, A. (eds.) Rule Interchange and Applications. pp. 43–52. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
12. Bonte, P., Ongenae, F.: Roxi: a framework for reactive reasoning. In: ESWC2023, the First International Workshop on Semantic Web on Constrained Things (2023)

13. de Bruijn, J., Heymans, S.: Logical foundations of (e)rdf(s): Complexity and reasoning. In: Aberer, K., Choi, K.S., Noy, N., Allemang, D., Lee, K.I., Nixon, L., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) The Semantic Web. pp. 86–99. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
14. Calì, A., Gottlob, G., Pieris, A.: Query answering under non-guarded rules in Datalog+/-. In: Hitzler, P., Lukasiewicz, T. (eds.) Proc. 4th Int. Conf. on Web Reasoning and Rule Systems (RR 2010). LNCS, vol. 6333, pp. 1–17. Springer (2010)
15. Carral, D., Dragoste, I., González, L., Jacobs, C., Krötzsch, M., Urbani, J.: Vlog: A rule engine for knowledge graphs. In: International Semantic Web Conference. pp. 19–35. Springer (2019)
16. Carral, D., Krötzsch, M.: Rewriting the description logic ALCHIQ to disjunctive existential rules. In: Bessiere, C. (ed.) Proceedings of the 29th International Joint Conference on Artificial Intelligence, IJCAI 2020. pp. 1777–1783. ijcai.org (2020)
17. Cuenca Grau, B., Horrocks, I., Krötzsch, M., Kupke, C., Magka, D., Motik, B., Wang, Z.: Acyclicity notions for existential rules and their application to query answering in ontologies. J. of Artificial Intelligence Research **47**, 741–808 (2013)
18. Deutsch, A., Nash, A., Remmel, J.B.: The chase revisited. In: Lenzerini, M., Lembo, D. (eds.) Proc. 27th Symposium on Principles of Database Systems (PODS'08). pp. 149–158. ACM (2008)
19. Ebbinghaus, H.D., Flum, J., Thomas, W.: Semantics of First-Order Languages, pp. 27–57. Springer New York, New York, NY (1994)
20. Fiorentino, A., Zangari, J., Manna, M.: DaRLing: A Datalog rewriter for OWL 2 RL ontological reasoning under SPARQL queries. Theory and Practice of Logic Programming **20**(6), 958–973 (2020)
21. Harth, A., Käfer, T.: Rule-based programming of user agents for linked data. In: Proceedings of the 11th International Workshop on Linked Data on the Web at the Web Conference (27th WWW). CEUR-WS (April 2018)
22. Hayes, P. (ed.): RDF Semantics. W3C Recommendation (10 February 2004), available at `http://www.w3.org/TR/rdf-mt/`
23. Ivliev, A., Ellmauthaler, S., Gerlach, L., Marx, M., Meißner, M., Meusel, S., Krötzsch, M.: Nemo: First glimpse of a new rule engine. In: 39th on Logic Programming, ICLP 2023 Technical Communications. EPTCS (to appear)
24. Krötzsch, M., Mehdi, A., Rudolph, S.: Orel: Database-driven reasoning for OWL 2 profiles. In: Haarslev, V., Toman, D., Weddell, G. (eds.) Proc. 23rd Int. Workshop on Description Logics (DL'10). CEUR Workshop Proceedings, vol. 573, pp. 114–124. CEUR-WS.org (2010)
25. Krötzsch, M., Marx, M., Rudolph, S.: The power of the terminating chase. In: Barceló, P., Calautti, M. (eds.) Proc. 22nd Int. Conf. on Database Theory (ICDT'19). LIPIcs, vol. 127, pp. 3:1–3:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2019)
26. Verborgh, R., Arndt, D., Van Hoecke, S., De Roo, J., Mels, G., Steiner, T., Gabarró, J.: The Pragmatic Proof: Hypermedia API Composition and Execution. Theory and Practice of Logic Programming (1), 1–48 (2017). `https://doi.org/10.1017/S1471068416000016`
27. Verborgh, R., De Roo, J.: Drawing Conclusions from Linked Data on the Web: The EYE Reasoner. IEEE Software (5), 23–27 (2015). `https://doi.org/10.1109/MS.2015.63`
28. Woensel, W.V., Arndt, D., Champin, P.A., Tomaszuk, D., Kellogg, G.: Notation3 language (Jul 2023), `https://w3c.github.io/N3/reports/20230703/`