Implicit quantification made explicit: How to interpret blank nodes and universal variables in Notation3 Logic

Dörthe Arndt[a,*], Tom Schrijvers[b], Jos De Roo[c], Ruben Verborgh[a]

[a]*Ghent University – imec, IDLab, Department of Electronics and Information Systems, Technologiepark-Zwijnaarde 122, 9052 Gent, Belgium*
[b] *KU Leuven – Department of Computer Science, Celestijnenlaan 200A, 3001 Leuven, Belgium*
[c] *Agfa Healthcare – Moutstraat 100, 9000 Ghent, Belgium*

Abstract

Since the invention of Notation3 Logic, several years have passed in which the theory has been refined and applied in different reasoning engines like Cwm, EYE, and FuXi. But despite these developments, a clear formal definition of Notation3's semantics is still missing. This does not only form an obstacle for the formal investigation of that logic and its relations to other formalisms, it has also practical consequences: in many cases the interpretations of the same formula differ between reasoning engines. In this paper we tackle one of the main sources of that problem, namely the uncertainty about implicit quantification. This refers to Notation3's ability to use bound variables for which the universal or existential quantifiers are not explicitly stated, but implicitly assumed. We provide a tool for clarification through the definition of a core logic for Notation3 that only supports explicit quantification. We specify an attribute grammar which maps Notation3 formulas to that logic according to the different interpretations and thereby define the semantics of Notation3. This grammar is then implemented and used to test the impact of the differences between interpretations on practical cases. Our dataset includes Notation3 implementations from former research projects and test cases developed for the reasoner EYE. We find that 31% of these files are understood differently by different reasoners. We further analyse these cases and categorize them in different classes of which we consider one most harmful: if a file is manually written by a user and no specific built-in predicates are used (13% of our critical files), it is unlikely that this user is aware of possible differences. We therefore argue the need to come to an agreement on implicit quantification, and discuss the different possibilities.

Keywords: N3, Scoping, RDF, Attribute Grammar, Formal Semantics

1. Introduction

The invention of Notation3 (N3) [1–3] brought two improvements to the Semantic Web: firstly, it provides an alternative to the rather verbose RDF/XML syntax [4] to represent RDF [5]; secondly, it extends the RDF model [6] with (1) universal variables, (2) the possibility to cite formulas, and (3) several logical operators, including in particular the implication operator. Together with these syntactic extensions a new rule-based logic was proposed, Notation3 Logic (N3Logic). While the first contribution has already had a big impact several years ago – it highly influenced the W3C recommendation Turtle [7] – the acceptance of the logic is still lagging behind. Even though N3Logic is implemented by several reasoners such as Cwm [8], EYE [9], and FuXi [10], its semantics has never been fully formalised. That not only forms a major obstacle for N3Logic becoming a standard, it has also practical consequences: the interpretations of certain formulas differ between the above mentioned engines. Considering the fact that N3Logic was designed for the Semantic Web, where interoperability is crucial, this problem deserves special attention.

In this paper we focus on the formalization of *implicit quantification*: N3Logic allows the user to express universal and existential quantification of variables implicitly, but the informal explanation of how this has to be interpreted [1, 2] is unclear about the scope of such variables. In our previous work [11] we have illustrated the differences in the reasoning results of the two reasoners

[*]Corresponding author
Email addresses: `doerthe.arndt@ugent.be` (Dörthe Arndt), `tom.schrijvers@cs.kuleuven.be` (Tom Schrijvers), `jos.deroo@agfa.com` (Jos De Roo), `ruben.verborgh@ugent.be` (Ruben Verborgh)

EYE and Cwm on a number of examples, and proposed a *direct semantics* for Cwm's interpretation of the W3C team submission [2]. This paper builds on the previous findings and addresses the following research questions:

(i) How can we formally express the difference between two interpretations of the same N3 formula?
(ii) How do existing interpretations of N3Logic conceptually differ in their way of handling implicit quantification?
(iii) How often does this conceptual difference lead to conflicting interpretations of formulas used in practical applications?
(iv) Which kinds of constructs cause these conflicting interpretations in practice and how likely is it that a file containing these constructs is actually subject to the problem?

To provide a better picture of the differences in the interpretations, we provide an *elaboration semantics*. We express the different interpretations in a *core logic* where only explicit quantification is allowed. We compare these translations for several N3 files, which were used in practical applications implemented in former research projects and provide an analysis of our findings. The contributions of this paper are the following:

- We define a core logic for N3 which only features explicitly quantified variables. The logic supports all concepts which are crucial for N3 such as the possibility to cite formulas.
- We clarify the differences in the interpretations of implicitly scoped formulas by the reasoners Cwm and EYE in terms of the core logic.
- We implement a translator from N3 into the different interpretations expressed in core logic. This translator can be used to test for conflicts and to avoid ambiguities.
- We perform a quantitative and qualitative evaluation of the occurrence of different interpretations for N3 files which are used in practice to better understand the practical impact of the problem.

The remainder of the paper is structured as follows: Section 2 provides an informal introduction to N3Logic with a special focus on implicit quantification to make the reader familiar with the related problems. After that, Section 3 defines the core logic for N3 which we then use to represent different interpretations of N3Logic. To obtain these different interpretations we make use of an attribute grammar which we define in in Section 4. This attribute grammar is then used to implement a translator which performs the mapping from N3 into its different interpretations in our core logic. In Section 5 we explain this implementation and use it to study the impact of the differences on files which are used in real applications and to analyse the cases where the interpretations differ in further detail. In Section 6 we discuss possible solutions for the problem. Section 7 puts our findings in context to other related work. Section 8 concludes the paper and provides an outlook on future work.

2. Background and Motivation

Before providing the formal definition of N3Logic and its semantics later in this paper, this section introduces the topic by examples. Our aim is to make the reader familiar with the uncertainties related to implicit quantification. If not indicated differently, the semantics presented here is based on the different official sources which can be found for N3Logic, in particular the W3C team submission [2], and the paper introducing the logic [1].

2.1. Simple triples and conjunctions

N3Logic is an extension of RDF. As in the latter, simple statements in N3 can be expressed in triples of the form *subject*, *predicate* and *object*, such as:

```
:Kurt :knows :Albert.        (1)
```

This triple means "*Kurt knows Albert*". Each ground component in N3 is represented by either an Internationalized Resource Identifier (IRI) [12], as done here, or by a literal. Different from RDF, literals can occur in all positions of a triple, not only in object position. In the example the IRI is abbreviated [7] with the empty prefix which here and in the remainder of this paper refers to an example namespace:

```
@prefix : <http://example.org/ex#>.
```

`:Kurt`, for instance, stands for the full IRI

```
<http://example.org/ex#Kurt>
```

If two triples occur together as in:

```
:Kurt :knows :Albert. :Albert :knows :Kurt.
```

This is understood as the conjunction of the formulas:

"*Kurt knows Albert* and *Albert knows Kurt.*"

2.2. *Referring to Formulas*

The formulas shown above, but also more complex formulas, can be cited using curly brackets. The statement

```
:John :says {:Kurt :knows :Albert.}.   (2)
```

means "*John says that Kurt knows Albert.*" The same notation of curly brackets is used, if one formula implies another, the implication symbol is expressed by "=>". The formula

```
{:Kurt :knows :Albert.} =>
              {:Albert :knows :Kurt.}.   (3)
```

represents the rule "*If Kurt knows Albert then Albert knows Kurt.*"

2.3. *Implicit Quantification*

The examples given so far did not contain variables and their interpretation was rather straightforward (maybe with the exception of cited formulas which can lead to discussions [13]). This is different when it comes to implicit quantification. Just like RDF, N3 allows the usage of implicitly existentially quantified variables, called *blank nodes*. Blank nodes either start with "`_:`" or are expressed using square brackets "`[ ]`". The formula

```
_:x :knows :Albert.   (4)
```

means "*There exists someone who knows Albert.*" In the formula

```
:John :knows [ :knows :Albert ].   (5)
```

the bracket stands for a "fresh" existentially quantified variable. The meaning is "*John knows someone who knows Albert.*" In N3, blank nodes can occur in every position, they can be subject, predicate or object.

Additionally to this kind of variables, N3 allows a second type of implicitly quantified variables, universals. These start with a question mark `?` and are understood as implicitly universally quantified. The formula

```
{:Kurt :knows ?x.} =>
                   {?x :knows :Kurt.}.   (6)
```

means "*Everyone Kurt knows also knows Kurt.*"

When universals and existentials occur together in the same formula the scope of the universals is always outside the scope of the existentials [2].

```
_:x :thinks {?y :is :pretty}.   (7)
```

is interpreted as

$$\forall y \exists x : \text{thinks}(x, \text{is}(y, \text{pretty})) \tag{7a}$$

"*For everyone there is someone who thinks that he/she is pretty*"

and not as

$$\exists x \forall y : \text{thinks}(x, \text{is}(y, \text{pretty})) \tag{7b}$$

"*There is someone who thinks that everyone is pretty.*"

In the cases above, the interpretation of the implicitly quantified formulas was rather easy: the variables are existentially and universally quantified at the top of the formula; if they co-occur in the same formula, the universal quantification dominates the existential. When implicitly quantified variables occur in deeply nested formulas, their intended meaning becomes less intuitive. This is the topic of the following subsections. As examples for different understandings we take the reasoning results of the reasoners Cwm [8] and EYE [14]. The reason for this choice is the coverage of N3 by these reasoners – in contrast to for example FuXi [10], they both support rather complex constructs like nested rules – and the fact that they conceptually differ in their way to handle implicit universal quantification. Further details about how differences can be detected can be found in our previous paper [11].

2.3.1. *Existentials*

We start by taking a closer look at a formula with nested implicit existential quantification:

```
_:x :says {_:x :knows :Albert.}.   (8)
```

The blank nodes stand for existentially quantified variables, but there are different ways to interpret these. The formula could either mean

$$\exists x : \text{says}(x, \text{knows}(x, \text{Albert})) \tag{8a}$$

"*There exists someone who says about himself that he knows Albert.*"

or

$$\exists x_1 : \text{says}(x_1, (\exists x_2 : \text{knows}(x_2, \text{Albert}))) \tag{8b}$$

"*There exists someone who says that there exists someone (possibly someone else) who knows Albert.*"

Here, the reasoners Cwm and EYE follow the second interpretation (8b) – and so does the informal semantics of N3Logic [2]:

"*When formulae are nested, _: blank nodes syntax* [is] *used to only identify* [a] *blank node in the formula it occurs directly in.*" (I)

Reading this last statement, there is still some uncertainty. What exactly is the *direct* formula? In N3 this formula is either the next higher formula in curly brackets `{ }` or, if no such formula exists, the current formula as a whole.

The reason for this design choice is that such direct formulas in N3 represent graphs, just as RDF graphs we encounter in different locations in the Web, and the scoping for blank nodes in such Web graphs is always local, ie on graph level.

2.3.2. Universals

Similarly to the case of implicit existential quantification, the interpretation of universals is not always evident and even differs between reasoners. To understand this, consider the following case of nested implications:

```
{{?x :p :a.} => {?x :q :b.}.} =>
        {{?x :r :c.} => {?x :s :d.}.}.   (9)
```

Are all `?x` the same? If not, which ones do we have to understand as equal and where are they quantified? Of the several options to interpret this formula, two seem to be most likely:

$$\forall x : ((p(x,a) \rightarrow q(x,b)) \rightarrow (r(x,c) \rightarrow s(x,d))) \quad (9a)$$

or

$$(\forall x_1 : p(x_1,a) \rightarrow q(x_1,b)) \rightarrow (\forall x_2 : r(x_2,c) \rightarrow s(x_2,d)) \quad (9b)$$

But which interpretation is correct? To answer this question we take a look at the W3C team submission which says:

"There is a also a shorthand syntax ?x which [...] implies that x is universally quantified not in the formula but in its parent formula." (II)

This raises a new question: Which formula is this *parent formula*? Neither the W3C team submission nor the paper introducing N3 provide an answer. We therefore look, how the implementers of EYE and Cwm understand the term.

For interpretation 9a the formula as a whole is the parent. All universals, independent of their exact position in the formula, are understood as quantified on top level, ie at the beginning of that parent formula. This is the interpretation EYE applies. For the remainder of this paper we refer to this concept of the term by adding the subscript e, ie *parent*$_e$ refers to the parent formula understood as the top formula.

In interpretation 9b, the universals in the antecedent and consequent of the main implication reside in two different scopes. This reflects Cwm's interpretation. Here, it is important that Formula 9 is composed of the subformulas:

```
{?x :p :a.} => {?x :q :b.}.   (10)
```

and

```
{?x :r :c.} => {?x :s :d.}.   (11)
```

Syntactically this division is marked by the use of curly brackets `{}`. Formula 10 has two direct subformulas (again marked by brackets), namely:

`{?x :p :a.}` and `{?x :q :b.}` (10a,b)

while

`{?x :r :c.}` and `{?x :s :d.}` (11a,b)

are subformulas of Formula 11. The term *parent formula* of a formula f is understood as the next formula g either occurring in curly brackets $\{g\}$ or being the top formula for which $\{f\}$ is a direct component (this is explained in detail in [11]). In our example, Formula 10 is the parent formula of Formulas 10a and b, and Formula 11 is the parent formula of Formulas 11a and b. This explains the scoping in interpretation 9b. We refer to this understanding of the concept parent by using the subscript c, we write *parent*$_c$.

The previous example reveals a general problem: We needed to explain how the W3C team submission is *interpreted* by the reasoners EYE and Cwm. Of course the term *parent formula* could be understood differently and only specific testing [11] made us come to our conclusion that the above are EYE's and Cwm's interpretations. Most users are not even aware of the differences in interpretations and the lack of a formalism renders it difficult to discuss or even just express them. To come to a clear and unambiguous definition of the logic, N3 needs to be formalised and there needs to be a formalism to express the differences of existing interpretations.

2.4. Explicit Quantification

Apart from implicit quantification as explained above, N3Logic also provides the possibility to explicitly quantify over variables. To do so, the quantifiers `@forSome` and `@forAll` are used. With explicit quantification, Interpretation 7a of Formula 7 can be expressed as follows:

```
@forAll :y.  @forSome :x.
          :x :thinks {:y :is :pretty}.
```

Seeing this example, the reader might think that the misunderstandings described above could be avoided by only using explicit quantification and that this notation could even be used to explain the differences. Unfortunately, that is not the case. Independently of the order they appear in the formula, universal quantifiers are always understood to be outside of existential quantifiers in N3 [2]. The formula

```
@forSome :x.  @forAll :y.
          :x :thinks {:y :is :pretty}.
```

has the exact same meaning as the previous one, namely Interpretation 7a.[1] To express Interpretation 7b, a more complicated construction is needed, which could then again lead to different interpretations.

The peculiarity described above leads to open questions. For example, what does the following valid N3 formula mean?

```
@forSome :x.  :x :knows :y.
         @forAll :y.  :y :knows :x.  (12)
```

The scope of the `@forAll` is outside of the scope of the `@forSome`, but what about the first occurrence of `:y` (underlined in the example)? Is `:y` a universally quantified variable or a constant? This formula is not supported by Cwm and its intended meaning is not specified in any source we are aware of. This and many similar examples make explicit quantification in N3 a complex topic on its own, which is outside the scope of this paper. Here we focus on implicit quantification.

3. Core Logic

The examples in the last section explain how N3 formulas are interpreted differently by different reasoners. To point out these variations, we used natural language together with a first-order-logic-like structure which allowed to cite formulas. However, both the natural language and logical structure do not have a fixed definition of their semantics and can thus still be understood in different ways. In order to dispose of this ambiguity when comparing interpretations, we now define a new *core logic* of N3. This logic supports all important features of N3 such as the possibility to refer to formulas or to use quantified variables in predicate position, but only allows *explicit* quantification. This logic can then be used to make N3's implicit quantification explicit.

Syntax:		
`t ::=`		terms:
	`v`	variables
	`c`	constants
	`e`	expressions
	`(k)`	lists
	`()`	empty list
`k ::=`		list content:
	`t`	
	`t k`	
`e ::=`		expressions:
	`<f>`	formula expression
	`<>`	true
	`false`	false
`f ::=`		formulas:
	`t t t`	atomic formula
	`e → e`	implication
	`f f`	conjunction
	`∀v.f`	universal formula
	`∃v.f`	existential formula

Figure 1: Syntax of the core language $\mathcal{L}$ over $\mathcal{V} \cup C$.

3.1. Syntax

Given disjoint countable sets of variables $\mathcal{V}$ and constants C we define the core language $\mathcal{L}$ of N3 over $C \cup \mathcal{V}$ as displayed in Figure 1. In core logic interpretation 8a is expressed as:

```
∃x.  x says <x knows Albert>
```

And Interpretation 8b:

```
∃x1.  x1 says <∃x2.  x2 knows Albert>
```

Note that this notation is close to the original N3 notation. To make a clear distinction between core logic and N3Logic, we use angle brackets instead of curly brackets and a different kind of arrow. For the same reason, we do not represent constants and variables using IRIs (ie we write x instead of `:x`) in our examples.[2] The main

[1] We suppose that this has historic reasons: When N3 was created, everything, including quantifiers, was represented in triples. The order of triples should not matter in any context.

[2] Note that the representation of the constants and variables only depends on the choice of C and $\mathcal{V}$.

difference between N3Logic and core logic is the symbol used for explicit quantification in the latter, which is taken from first order logic to emphasize that the quantifiers here are interpreted in the order they occur.

Variables in a formula can either occur free or bound:

Definition 1 (Free variables). The set of free variables of a language element $\mathtt{l} \in \mathcal{L}$, written $\mathrm{FV}(\mathtt{l})$ is defined as follows:

$$\begin{aligned}
\mathrm{FV}(\mathtt{v}) &= \{\mathtt{v}\} \\
\mathrm{FV}(\mathtt{c}) &= \emptyset \\
\mathrm{FV}(\mathtt{<f>}) &= \mathrm{FV}(\mathtt{f}) \\
\mathrm{FV}(\mathtt{<>}) &= \emptyset \\
\mathrm{FV}(\mathtt{false}) &= \emptyset \\
\mathrm{FV}(\mathtt{(t_1 \dots t_n)}) &= \mathrm{FV}(\mathtt{t_1}) \cup \ldots \cup \mathrm{FV}(\mathtt{t_n}) \\
\mathrm{FV}(\mathtt{()}) &= \emptyset \\
\mathrm{FV}(\mathtt{t_1 t_2 t_3}) &= \mathrm{FV}(\mathtt{t_1}) \cup \mathrm{FV}(\mathtt{t_2}) \cup \mathrm{FV}(\mathtt{t_3}) \\
\mathrm{FV}(\mathtt{e_1} \rightarrow \mathtt{e_2}) &= \mathrm{FV}(\mathtt{e_1}) \cup \mathrm{FV}(\mathtt{e_2}) \\
\mathrm{FV}(\mathtt{f_1 f_2}) &= \mathrm{FV}(\mathtt{f_1}) \cup \mathrm{FV}(\mathtt{f_2}) \\
\mathrm{FV}(\forall \mathtt{v.f}) &= \mathrm{FV}(\mathtt{f}) \setminus \{\mathtt{v}\} \\
\mathrm{FV}(\exists \mathtt{v.f}) &= \mathrm{FV}(\mathtt{f}) \setminus \{\mathtt{v}\}
\end{aligned}$$

We call every language element $\mathtt{l} \in \mathcal{L}$ with $\mathrm{FV}(\mathtt{l}) = \emptyset$ *ground*. We denote the set of ground language elements by $\mathcal{L}_g$, for the set $\mathcal{T}$ of terms as defined in Figure 1, we define $\mathcal{T}_g := \mathcal{L}_g \cap \mathcal{T}$ as the set of ground terms.

For example, in the formula

$$\forall \mathtt{x}.\quad \mathtt{x\ knows\ y}$$

the variable x occurs bound, while y is free.

3.2. Semantics

To define the semantics of the core language $\mathcal{L}$ over $\mathcal{V} \cup C$ we first introduce the notion of structure.

Definition 2 (Structure). A structure over a language $\mathcal{L}$ is a quadruple $\mathfrak{A} = (\mathcal{D}, \mathcal{P}, \mathfrak{a}, \mathfrak{p})$ containing:

- A non empty set $\mathcal{D}$ called the *domain*.
- A non empty set $\mathcal{P} \subset \mathcal{D}$ called the *set of properties*.
- A mapping $\mathfrak{a} : \mathcal{T}_g \rightarrow \mathcal{D}$ called the *object mapping*.
- A mapping $\mathfrak{p} : \mathcal{P} \rightarrow 2^{\mathcal{D} \times \mathcal{D}}$ called the *predicate mapping*.

Similarly to a structure in the classical first order sense, a core logic structure consists of a domain of discourse and maps from the terms of the language into that domain. There are two main differences:

Firstly, the interpretation function for relations does not directly act on symbols of the language but on a subset of the domain of discourse. The reason for that is that RDF and N3 – and thereby also its core logic – do not distinguish between constants and predicates. One single symbol can be used as predicate and subject or object at the same time:

```
knows a predicate.  Albert knows Kurt.
```

is a formula of the core language. An interpretation needs to take the connection between the two occurrences of the term "`knows`" into account.

The second difference is that our structure is only defined for ground terms and not, as it is in other logics, for terms containing variables. To define the semantics for these we make use of a grounding function:

Definition 3 (Grounding Function). A *grounding function* γ_g over a language $\mathcal{L}$ is a mapping from the set of variables $\mathcal{V}$ into the set of ground terms $\mathcal{T}_g$.

This function can be extended to all elements of the language:

Definition 4 (Extended Grounding Function). For a language $\mathcal{L}$ and a *grounding function* $\gamma_g : \mathcal{V} \rightarrow \mathcal{T}_g$ we define its extension $\gamma : \mathcal{L} \rightarrow \mathcal{L}_g$ as follows:

$$\begin{aligned}
\gamma(\mathtt{v}) &= \gamma_g(\mathtt{v}) \\
\gamma(\mathtt{c}) &= c \\
\gamma(\mathtt{<f>}) &= \mathtt{<}\gamma(\mathtt{f})\mathtt{>} \\
\gamma(\mathtt{<>}) &= \mathtt{<>} \\
\gamma(\mathtt{false}) &= \mathtt{false} \\
\gamma(\mathtt{(t_1 \dots t_n)}) &= (\gamma(\mathtt{t_1}) \ldots \gamma(\mathtt{t_n})) \\
\gamma(\mathtt{()}) &= \mathtt{()} \\
\gamma(\mathtt{t_1 t_2 t_3}) &= \gamma(\mathtt{t_1})\gamma(\mathtt{t_2})\gamma(\mathtt{t_3}) \\
\gamma(\mathtt{e_1} \rightarrow \mathtt{e_2}) &= \gamma(\mathtt{e_1}) \rightarrow \gamma(\mathtt{e_2}) \\
\gamma(\mathtt{f_1 f_2}) &= \gamma(\mathtt{f_1})\gamma(\mathtt{f_2}) \\
\gamma(\forall \mathtt{v.f}) &= \forall \mathtt{v}.\gamma[\mathtt{v} \mapsto \mathtt{v}](\mathtt{f}) \\
\gamma(\exists \mathtt{v.f}) &= \exists \mathtt{v}.\gamma[\mathtt{v} \mapsto \mathtt{v}](\mathtt{f})
\end{aligned}$$

Where $\gamma[x \mapsto t]$ denotes the extended grounding function which is identical to γ except for $x \mapsto t$.

For the definition of an interpretation and meaning we now use the grounding function instead of a classical valuation function which maps the set of variables into the domain of discourse. We do that to be able to make a distinction between terms occurring in a cited formula and subjects, predicates and objects of formulas directly occurring in a graph. The informal specification of N3's semantics claims that nested formulas are not "*referentially transparent*" [1, p.7]. This means that even if two terms in a cited formula denote the same object in the domain of discourse two cited formulas which only differ in the use of the referent should be considered as different. As a concrete example consider the following formulas:[3]

```
LoisLane believes <Superman can fly.>.
```

Even if we know that the term "`Superman`" denotes the same object as "`ClarkKent`", this should not imply the following formula:

```
LoisLane believes <ClarkKent can fly.>.
```

But cited formulas can also occur in combination with quantifiers which are outside of the quoted formula like for example:

```
∃x. LoisLane believes <x can fly.>.
```

To know whether or not the last statement is true, it needs to be possible to map an existentially quantified variable to its representation instead of directly mapping it to the resource it represents.

An interpretation for core formulas therefore makes use of a grounding function:

Definition 5 (Core logic interpretation). A core logic interpretation $\mathfrak{I}$ is a pair $(\mathfrak{A}, \gamma)$ consisting of a structure $\mathfrak{A}$ and a grounding function γ.

With the present definitions we can now define the meaning of formulas.

Definition 6 (Meaning of formulas). Let $\mathfrak{I} = (\mathfrak{A}, \gamma)$ be a core logic interpretation of a language $\mathcal{L}$. Then:

1. $\mathfrak{I} \models \mathtt{t_1 t_2 t_3}$ iff $(\mathfrak{a}(\gamma(\mathtt{t_1})), \mathfrak{a}(\gamma(\mathtt{t_3}))) \in \mathfrak{p}(\mathfrak{a}(\gamma(\mathtt{t_2})))$
2. $\mathfrak{I} \models \mathtt{<f_1>} \rightarrow \mathtt{<f_2>}$ iff $\mathfrak{I} \models \mathtt{f_2}$ if $\mathfrak{I} \models \mathtt{f_1}$.
3. $\mathfrak{I} \models \mathtt{false} \rightarrow \mathtt{<f>}$
4. $\mathfrak{I} \models \mathtt{<f>} \rightarrow \mathtt{<>}$
5. $\mathfrak{I} \models \mathtt{<f>} \rightarrow \mathtt{false}$ iff $\mathfrak{I} \not\models \mathtt{f}$.
6. $\mathfrak{I} \models \mathtt{<>} \rightarrow \mathtt{<f>}$ iff $\mathfrak{I} \models \mathtt{f}$.
7. $\mathfrak{I} \models \mathtt{f_1 f_2}$ iff $\mathfrak{I} \models \mathtt{f_1}$ and $\mathfrak{I} \models \mathtt{f_2}$.
8. $\mathfrak{I} \models \forall \mathtt{v}.\mathtt{f}$ iff $(\mathfrak{A}, \gamma[\mathtt{v} \mapsto \mathtt{t}]) \models \mathtt{f}$ for all $t \in \mathcal{T}_g$.
9. $\mathfrak{I} \models \exists \mathtt{v}.\mathtt{f}$ iff $(\mathfrak{A}, \gamma[\mathtt{v} \mapsto \mathtt{t}]) \models \mathtt{f}$ for some $t \in \mathcal{T}_g$.

We call a formula `f` *true* in $\mathfrak{I}$ iff $\mathfrak{I} \models \mathtt{f}$.

Note that in N3 lists are treated as "first-class citizens", this means that N3 does not use RDF's `rdf:first-rdf:rest` notation to construct lists but allows the treatment of lists as a proper data type (see also [1, p.6]). This is reflected in our core logic and its semantics, where we directly map ground lists into the domain of discourse. We do the same for cited formulas since such a treatment allows later refinement by specifying the mapping function further.

Another remark we want to make here is that by using grounding for quantified predicates and mapping them first to a single element of the domain before assigning them to the set of pairs of domain elements for which the relation they denote holds ensures that we stay in first order logic.[4] We do not quantify over $2^{\mathfrak{D}\times\mathfrak{D}}$ but over the countable subset of relations which have a name in $\mathcal{L}$ and can therefore follow the idea of Henkin Semantics [15] to map core logic to first order logic.

The definition of a model is similar to first order logic:

Definition 7 (Model). Let Φ be a set of formulas. We call a core logic interpretation $\mathfrak{I}$ a *model* of Φ iff every formula in Φ is true in $\mathfrak{I}$.

We finish this section by defining the classical concepts of logical consequence and equivalence:

Definition 8 (Logical consequence). Let Φ be a set of ground formulas. A ground formula `f` is called a *logical consequence* of Φ (written: $\Phi \models \mathtt{f}$) iff `f` is true in every model of Φ.

Instead of $\{\mathtt{f_1}\} \models \mathtt{f_2}$ we sometimes write $\mathtt{f_1} \models \mathtt{f_2}$.

Definition 9 (Logical equivalence). Two formulas ϕ and ψ are called logically equivalent ($\phi \equiv \psi$) iff $\phi \models \psi$ and $\psi \models \phi$.

4. Interpretations of N3

The core logic enables us to give more formal descriptions of the N3 interpretations explained in Section 2. To be able to map an N3 formula to its possible interpretation we make use of attribute grammars [16, 17]. Below, we explain this concept and specify the different attributes needed for this mapping.

4.1. N3 Syntax

We start by giving a more accurate definition of N3's syntax. An N3 alphabet contains all symbols which can

[3]This example is often called the "superman problem" and has been broadly discussed in the context of RDF reification. See for example: `https://www.w3.org/2001/12/attributions/#superman`.

[4]Strictly speaking one of the two mechanisms would already be enough to ensure that we stay in first order logic.

`s ::=`		start:
	`f`	formula
`f ::=`		formulas:
	`t t t.`	atomic formula
	`e => e.`	implication
	`f f`	conjunction
`t ::=`		terms:
	`uv`	universal variables
	`ex`	existential variables
	`c`	constants
	`e`	expressions
	`(k)`	lists
	`()`	empty list
`k ::=`		list content:
	`t`	term
	`t k`	term tail
`e ::=`		expressions:
	`{f}`	formula expression
	`{}`	true
	`false`	false

Figure 2: Overview N3 Syntax

appear in the concrete representation of an N3 formula:

Definition 10 (N3 Alphabet). Let C be a set of constants, U a set of universal variables and E a set of existential variables. Let these sets be mutually disjoint and disjoint with $\{$`{`, `}`, `(`, `)`, `=>`, `false`, `.`$\}$. Then we call

$$\mathcal{A} := C \cup U \cup, E \cup \{\texttt{\{}, \texttt{\}}, \texttt{(}, \texttt{)}, \texttt{=>}, \texttt{false}, \texttt{.}\}$$

an *N3 alphabet*.

In concrete representations constants can either be literal values or IRIs. Universal variables start with the symbol `?`, existential variables with `_:`.

Given such an N3 alphabet $\mathcal{A}$, we define an N3 grammar over $\mathcal{A}$ as displayed in Figure 2 where the node `ex` stands for an existential variable, ie an element of E, `uv` for a universal variable, ie an element of U, and `c` for a constant, ie an element of C.

It is easy to see that N3 syntax is very similar to the syntax of the core logic. Some differences only apply to the choice of symbols: the curly brackets `{}` can be translated into angle brackets `< >` and the N3 arrow `=>` into a simple $\rightarrow$. The formula

```
{:s1 :p1 :o1.} => {:s2 :p2 :o2.}.        (13)
```

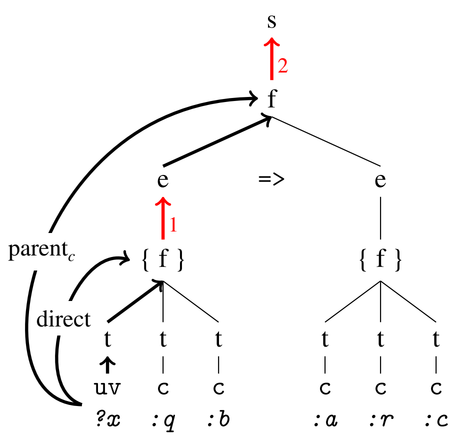

Figure 3: Direct and parent$_c$ formula.

can be expressed as

$$\texttt{<s1 p1 o1>} \rightarrow \texttt{<s2 p2 o2>} \qquad (13')$$

A major difference between core logic and N3 as displayed here is the lack of explicit quantification in the latter[5] and the existence of two different kinds of variables instead. To translate these implicitly quantified variables of N3 into the explicit ones of the core logic we use the structure of the syntax tree.

To illustrate the idea behind that, we take a closer look at one of the concepts used in the informal definitions of N3's quantification: according to the W3C team submission [2] existential variables are quantified on their *direct formula* (Section 2.3.1) and universal variables are quantified on their *parent formula* (Section 2.3.2). Informally such a *direct formula* f of a term t is the next formula surrounded by curly brackets { } that contains t, or, if such a formula does not exist, the top formula. In EYE's interpretation, the *parent formula* is now the top formula (concept *parent$_e$*). For Cwm, the *parent formula* g of f is the next higher formula surrounded by curly brackets, ie the direct formula of $\{f\}$ (*parent$_c$*). Being the top formula, the *parent$_e$ formula* is easy to find. For the *direct formula* and the *parent$_c$ formula* we use the syntax tree: The difference between a simple formula f and a formula in brackets $\{f\}$ is in the grammar point of view the difference between a *formula* `f` and a *formula expression* `e`. If we want to find the direct formula of a component and its parent$_c$, we

[5]As explained in Section 2.4 we exclude the discussion of N3's explicit quantification from this paper. It is therefore also not reflected in the grammar.

Grammar			Synthesized attribute *ds*		
`s`	`::=`	`n`	`s`.*ds*	$\leftarrow$	`n`.*ds*
`n`	`::=`	`d`	`n`.*ds*	$\leftarrow$	`d`
		$\mathtt{n}_1\mathtt{n}_2$		$\leftarrow$	$\mathtt{n}_1.ds + \mathtt{n}_2.ds$

Figure 4: Context-free grammar producing integers of arbitrary length (left) and definition of the attribute *ds* which composes the digit sum (right).

can go through the syntax tree from the bottom to the top. The *direct formula* is then the first formula on that path which is a direct child of either an expression `e` or the start symbol `s`. The *parent$_c$ formula* is the second such formula.[6] We illustrate this in Figure 3 on the syntax tree of the formula:

$$\texttt{\{?x :q :b\} => \{:a :r :c\}.} \qquad (14)$$

The direct formula of the terminal node `?x` is the formula `?x :q :b.`, the parent$_c$ formula is the top formula. This parent$_c$ formula carries the universal quantifier for the variable. The formula means according to Cwm:

$$\forall \texttt{x}.\quad \texttt{<x q b>} \rightarrow \texttt{<a r c>}. \qquad (14')$$

Attribute grammars provide a formalism to describe the above process of *going through the syntax tree* in a more precise way.

4.2. Attribute Grammars

In this section we give an introduction to attribute grammars [16, 17]. Attribute grammars are defined on top of context-free grammars like the one given above and extend this concept by so-called *attributes*. Such attributes are defined on the nodes of the syntax tree and can take values. These values can depend on other attribute values of the node itself and either its descendent nodes in the syntax tree – in this case the attribute is called *synthesized* – or on the attribute values of the parent node – in this case it is called *inherited*. The definitions used follow the notation of Paakki [18].

4.2.1. Example

Before providing a formal definition, we illustrate the idea of attribute grammars on a simpler example than the grammar provided above. Consider the context-free grammar displayed on the left side of Figure 4. If `d` can take any value of the alphabet $\mathcal{A} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 0\}$, this grammar produces integers of arbitrary length. A possible[7] syntax tree for 9487 is displayed in Figure 5. To ease the following discussion, the tree nodes of the same kind are numbered.

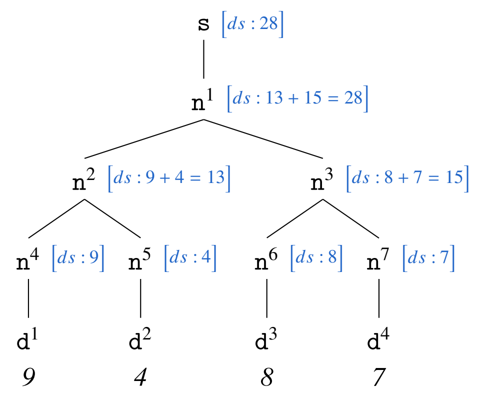

Figure 5: Syntax tree for 9487 produced by the grammar in Figure 4 with attribute values for *ds* (in *blue*).

If we now want to get the digit sum of an integer produced by that grammar, we can go through the syntax tree from the bottom to the top: from the nodes resulting in a digit, we store that digit. For every node higher in the syntax tree, we take the values of its children and sum them up to a new value. Then the values of the start node is the digit sum of the integer.

To formalise this process we define the synthesized attribute *ds*. This attribute takes values on each node and these values depend on the production rules of the grammar. For each production rule, we define an *attribute rule*. These rules are displayed at the right side of Figure 4. We denote the attribute value for a node `n` by `n`.*ds*. When we use a node resulting in an alphabet symbol of the grammar (in our example `d`) in an attribute rule, that symbol refers its alphabet symbol. Going through the syntax tree from the bottom to the top, we get for node $\mathtt{n}^4$ by attribute rule $\mathtt{n}.ds \leftarrow \mathtt{d}$ the value $\mathtt{n}^4.ds = 9$. The same rule is used to determine $\mathtt{n}^5.ds$ to $\mathtt{n}^7.ds$ whose values are displayed in Figure 5. On the next higher level, the attribute value of each node is composed by taking the sum of the attribute values of the direct descendants. This is done by the rule $\mathtt{n}.ds \leftarrow \mathtt{n}_1.ds + \mathtt{n}_2.ds$. We get $\mathtt{n}^2.ds = 13$ and $\mathtt{n}^3.ds = 15$. The same rule computes one level higher for $\mathtt{n}^1$ the value $\mathtt{n}^1.ds = 28$ which is then passed to the start node `s` by the rule $\mathtt{s}.ds \leftarrow \mathtt{n}.ds$. And 28 is indeed the digit sum of 9487.

[6]Note that universals on the top level like for example in the formula "`?x :p :o.`" do not have a parent$_c$. We assume here, that these formulas are also quantified on the top level. This differs from Cwm which does not support universal quantification on the top level.

[7]Note that there are several options to form a tree for 9487, attribute *ds* also computes the digit sum if one of these is chosen.

4.2.2. Terminology

After our example, we now provide a more formal introduction to attribute grammars and fix the terminology we use. Attribute grammars are defined on top of context free grammars $G = \langle N, \mathcal{A}, P, S\rangle$, with N the set of non-terminal symbols, $\mathcal{A}$ the alphabet or set of terminal symbols, P the set of production rules and $S \in N$ a start symbol. The set P for the context-free grammar above is displayed in Figure 4. This figure also contains all non terminal symbols N. The start symbol is `s`.

For each non-terminal symbol $X \in N$ of the grammar we now define a set of attributes $A(X)$. Each of these attributes either belongs to the set of *inherited* attributes $I(X)$, or to the set of *synthesized* attributes $S(X)$. We assume these two sets of attributes to be disjoint. In our example above we have for each node X: $I(X) = \emptyset$, $S(X) = \{ds\}$, and $A(X) = \emptyset \cup \{ds\} = \{ds\}$.

To take information, the attributes have assigned values which depend on the production rules they occur in.

Definition 11 (Attribute Occurrence). Let $G = \langle N, \mathcal{A}, P, S\rangle$ be a context-free grammar and $A := \bigcup_{X\in N} A(X)$ a set of attributes defined on N. Let $p \in P$ be a production rule of the form

$$X_0 ::= X_1 \cdots X_n \text{ with } n \geq 1,$$

- We say that p has an *attribute occurrence* $X_i.a$ for the attribute a if $a \in A(X_i)$ for some i with $0 \leq i \leq n$.
- We say that p has a *left-side occurrence* $X_0.a$ of the attribute a, if $a \in A(X_0)$.
- We say that p has a *right-side occurrence* $X_k.a$ of the attribute a, if $a \in A(X_k)$ for some k with $1 \leq k \leq n$.

If we consider another attribute for the grammar in Figure 4, which is defined only for the node `d`, the attribute *at*, then the rule

$$\mathtt{n ::= d}$$

has a *right-side occurrence* `d`.*at* of the attribute *at* while the rule

$$\mathtt{n ::= n_1 n_2}$$

does not have any occurrence of *at*.

These occurrences $X_i.a$ now take values, so-called *attribute values*, which are defined by *attribute rules*. These have the form

$$X_i.a \leftarrow f(y_1 \dots y_n)$$

where f is a function and $y_1 \dots y_n$ are other attribute values. An example from above for such an attribute rule is

$$\mathtt{n}.ds \leftarrow \mathtt{n}_1.ds + \mathtt{n}_2.ds$$

which determines the first occurrence `n`.*ds* of the attribute *ds* in the production rule

$$\mathtt{n ::= n_1 n_2}$$

We denote the set of all attribute rules for a production rule $p \in P$ by $R(p)$. Which exact attribute values can be taken into account when calculating a new attribute value depends on the kind of attribute:

Definition 12 (Attribute Grammar). Let $G = \langle N, \mathcal{A}, P, S\rangle$ be a context-free grammar. For every element $X \in N$ let $A(X) = I(X) \cup S(X)$ be a finite set of attributes with $I(X) \cap S(X) = \emptyset$. Let $A = \bigcup_{X\in\mathcal{A}\cup N} A(X)$ be the set of all these attributes. Let $R = \bigcup_{p\in P} R(p)$ be a finite set of attribute rules. We call

$$AG = \langle G, A, R\rangle$$

an *Attribute Grammar* if for each production rule $p \in P$ of the form $X_0 \ ::= \ X_1 \dots X_n$ the following holds:

- for each left-side occurrence $X_0.a$ of a synthesised attribute $a \in S(X_0)$ there exists *exactly one* attribute rule $(X_0.a \leftarrow f(y_1, \dots, y_k)) \in R(p)$ where f is a function and $y_i \in A(X_0) \cup \dots \cup A(X_n)$ for all $1 \leq i \leq k$.
- for each right-side occurrence $X_i.a$, $1 \leq i \leq n$, of an inherited attribute $a \in I(X_i)$ there exists *exactly one* attribute rule $(X_i.a \leftarrow f(y_1, \dots, y_k)) \in R(p)$ where f is a function and $y_j \in A(X_0) \cup \dots \cup A(X_n)$ for all $1 \leq j \leq k$.

In terms of a syntax tree generated by the grammar synthesized and inherited attributes have exactly the properties mentioned above: inherited attributes pass information *down* in the syntax tree, synthesized attributes pass information *upwards*. We assume the values of synthesized attributes defined on terminal symbols to be defined externally. The start symbol cannot take an inherited attribute.[8]

4.3. Definition of an N3 Attribute Grammar

After the definition of attribute grammars in the previous section, we now apply this concept to formalise

[8]The attentive reader might have noticed another difference in the structure of the context-free grammar of the core logic (Figure 1) and the syntax of N3 (Figure 2): the latter has an extra start symbol. The reason for this is rather technical: the attribute grammar we define in the following sections makes use of inherited attributes defined on the symbol `f`. Such attributes cannot be defined on a start symbol.

name	type	defined for	purpose
eq	syn	N	existentials
v_1	syn	N	universals in Cwm
v_2	syn	N	universals in Cwm
s	inh	$\{\mathtt{f}, \mathtt{t}, \mathtt{e}, \mathtt{k}\}$	universals in Cwm
q	inh	$\{\mathtt{f}\}$	universals in Cwm
u	syn	N	universals in EYE
m_c	syn	N	translation Cwm
m_e	syn	N	translation EYE

Table 1: Overview of all attributes defined in the N3-attribute grammar. The type can be *syn* for synthesized or *inh* for inherited.

the different interpretations of implicit quantification for N3Logic. The context-free grammar of N3 is given in Figure 2 which also contains all symbols of N. The alphabet is given in Definition 10. The start symbol is `s`.

On top of this grammar we now define an attribute grammar in order to be able to produce the two different translations of an N3 formula in core logic, one according to Cwm and the team submission and the other according to EYE. As a first step we provide an overview of the attributes we define, state whether they are inherited or synthesized and list for which nodes $X \in N$ they are defined. This information is displayed in Table 1. For each of these attributes we also indicated the purpose or context they are used for. This context can be grouped in four parts: the collection of the variables which are existentially quantified under a (sub-)formula (1), the collection of the universal variables quantified under a (sub-)formula – once for Cwm (2) and once for EYE (3) – and attributes to construct the concrete expression in core logic which is the translation of the N3 formula dependent on the reasoner (4). In the following we will go through these four groups one by one, explain the need for the different attributes and provide their definition. We start with attribute eq, which is used for existential scoping.

4.3.1. Existentials

As we have seen in Section 2.3.1 the scope on an implicitly existentially quantified variable is the *direct* formula it occurs in. The concept of a *direct formula* has been further explained above: the direct formula is either the next formula in curly brackets surrounding the existential variable, or – in case such a formula does not exist – the formula as a whole. To recall the idea, consider the following formula:[9]

$$\texttt{\_:x :s \{\_:y :k :a.\}.} \qquad (15)$$

The direct formula of variable `_:y` is the sub-formula `_:y :k :a.` The direct formula of `_:x` is the formula as a whole. Translated into core logic the formula means:

$$\exists \mathtt{x}.\mathtt{x}\ \mathtt{s}\ \langle\exists \mathtt{y}.\mathtt{y}\ \mathtt{k}\ \mathtt{a}\rangle. \qquad (15')$$

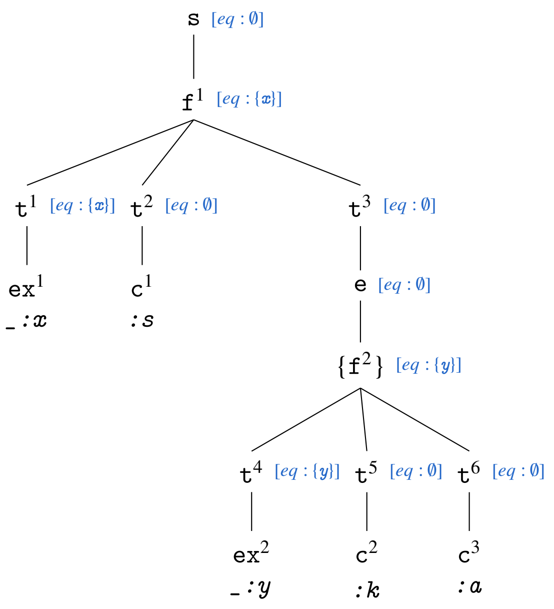

Figure 6: Syntax tree for Formula 15 with values for the attribute eq (in *blue*).

In Section 4.1 we discussed the idea of using the syntax tree to find the direct formula. If we go upwards in the syntax tree from the occurrence of an existential variable to the next higher node `e` or `s`, the child formula `f` of that node is the existential's direct formula which thus carries its quantifier. The syntax tree for our example, Formula 15, is displayed in Figure 6. Here node $\mathtt{f}^2$ is such a node for `_:y` and node $\mathtt{f}^1$ for `_:x`.

Attribute eq is now used in this context to keep track of all existential variables which need to be quantified. If we do a direct translation of the symbols of the alphabet as mentioned at the end of Section 4.1 and add existential quantifiers whenever we encounter a direct formula, the attribute carries for each node the set of existential variables which are free (see Definition 1) un-

[9]Note that structure-wise this formula follows the same pattern as Formula 8. The names of constants are only shortened to make the presentation of the formula easier.

production rule		attribute rule
`s ::=`	`f`	s.eq $\leftarrow \emptyset$
`f ::=`	$\mathtt{t}_1\mathtt{t}_2\mathtt{t}_3$.	f.eq $\leftarrow$ $\mathtt{t}_1$.eq $\cup$ $\mathtt{t}_2$.eq $\cup$ $\mathtt{t}_3$.eq
	$\mathtt{e}_1$`=>`$\mathtt{e}_2$.	f.eq $\leftarrow$ $\mathtt{e}_1$.eq $\cup$ $\mathtt{e}_2$.eq
	$\mathtt{f}_1\mathtt{f}_2$	f.eq $\leftarrow$ $\mathtt{f}_1$.eq $\cup$ $\mathtt{f}_2$.eq
`t ::=`	`uv`	t.eq $\leftarrow \emptyset$
	`ex`	t.eq $\leftarrow$ {ex}
	`c`	t.eq $\leftarrow \emptyset$
	`e`	t.eq $\leftarrow$ e.eq
	`(k)`	t.eq $\leftarrow$ k.eq
	`()`	t.eq $\leftarrow \emptyset$
`k ::=`	`t`	k.eq $\leftarrow$ t.eq
	`t` $\mathtt{k}_1$	k.eq $\leftarrow$ t.eq $\cup$ $\mathtt{k}_1$.eq
`e ::=`	`{f}`	e.eq $\leftarrow \emptyset$
	`{}`	e.eq $\leftarrow \emptyset$
	`false`	e.eq $\leftarrow \emptyset$

Figure 7: Attribute rules for the synthesized attribute eq (right) and their corresponding production rules (left) from the N3 grammar (Figure 2).

der that node and need to be bound when new existential quantifiers are added.

Following that idea, we define the attribute rules for eq. As eq is synthesised we need to state one attribute rule for each left-side occurrence of an attribute on a production rule of Figure 2. As the attribute is defined for all nodes, we need one attribute rule for each production rule. These attribute rules are displayed in Figure 7 (right) next to their corresponding production rules (left). To illustrate how this attribute works, we added the attribute values for each node to the syntax tree in Figure 6. We explain these rules by going through the tree beginning at the bottom.

The production rule `t ::= ex` results in an existential variable and this variable is not quantified under that node. Therefore, the corresponding attribute rule t.eq $\leftarrow$ {ex} collects this variable in a singleton set (values $\mathtt{t}^1$.eq and $\mathtt{t}^4$.eq). The attribute rules for other production rules directly resulting in only symbols of the alphabet do not pass any values since there are no free existential variables occurring under them. For the applications of `t ::= c` the attribute rule t $\leftarrow \emptyset$ assigns the empty set to the nodes $\mathtt{t}^2$, $\mathtt{t}^5$ and $\mathtt{t}^6$. Most other rules now pass the variables from the descendant nodes up to the parents. The attribute value of a formula node `f` is the union of its descendants' values. For $\mathtt{f}^2$ that is the set {`_:y`}. The only exceptions for this behaviour of passing the variables upwards can be found on the attribute rules for the production rules `s ::= f` and `e ::= {f}`. As discussed above, the child formulas `f` of `e` or `s` are *direct formulas*. All free existential variables occurring under such direct formulas get bound on these formulas. The attribute rules for `s ::= f` and `e ::= {f}` thus do not pass any variables upwards. The attribute value for node `e` in our syntax tree is the empty set. This value is again passed upwards via the attribute rule t.eq $\leftarrow$ e.eq and can the be used to determine the attribute value for $\mathtt{f}_1$ which is again the union of all its direct descendants' values ($\mathtt{f}^2$.eq = {`_:x`}). This node is again a direct formula, it is the child node of the node `s`. All existential variables are thus bound on $\mathtt{f}^2$, the attribute rule does not pass existential variables upwards.

The attribute value of eq is always the set of free existential variables occurring under a node. For the formulas $\mathtt{f}^1$ and $\mathtt{f}^2$ these are the sets `{_:x}` and `{_:y}`, respectively. These are exactly the variables existentially quantified on these formulas as can be seen in Formula 15'.

4.3.2. Universals in Cwm

The interpretations of N3's implicit universal quantification differs between reasoners. With a few exceptions,[10] which we also discuss in Appendix A, Cwm implements interpretation *parent*$_c$ of the concept *parent formula* from the W3C team submission while for EYE the *parent formula* is the top formula (*parent*$_e$). In this section we explain the concept of *parent*$_c$ in more detail and define attributes to place universal quantifiers according to this concept. The details on the interpretation following to EYE are discussed in the next section.

In the previous sections we have seen that in contrast to implicitly existentially quantified variables, universals are – according to the W3C team submission – not quantified on the *direct formula* but on the *parent*. This concept has (at least) two conflicting interpretations which have been further explained in Section 4.1. One of them is parent$_c$: The parent$_c$ is the direct formula of the direct formula or – in terms of the syntax tree – the second descendant `f` of a node `s` or `e` we find when going up the syntax tree from the bottom to the top. In the following formula, for example,

```
{{?x :q ?y.} => {?x :r :c.}.} =>
                        {?x :p :a.}.  (16)
```

[10]The developer of Cwm, Tim Berners-Lee, confirmed that these exceptions were not intended and rather need to be seen as mistakes in the implementation.

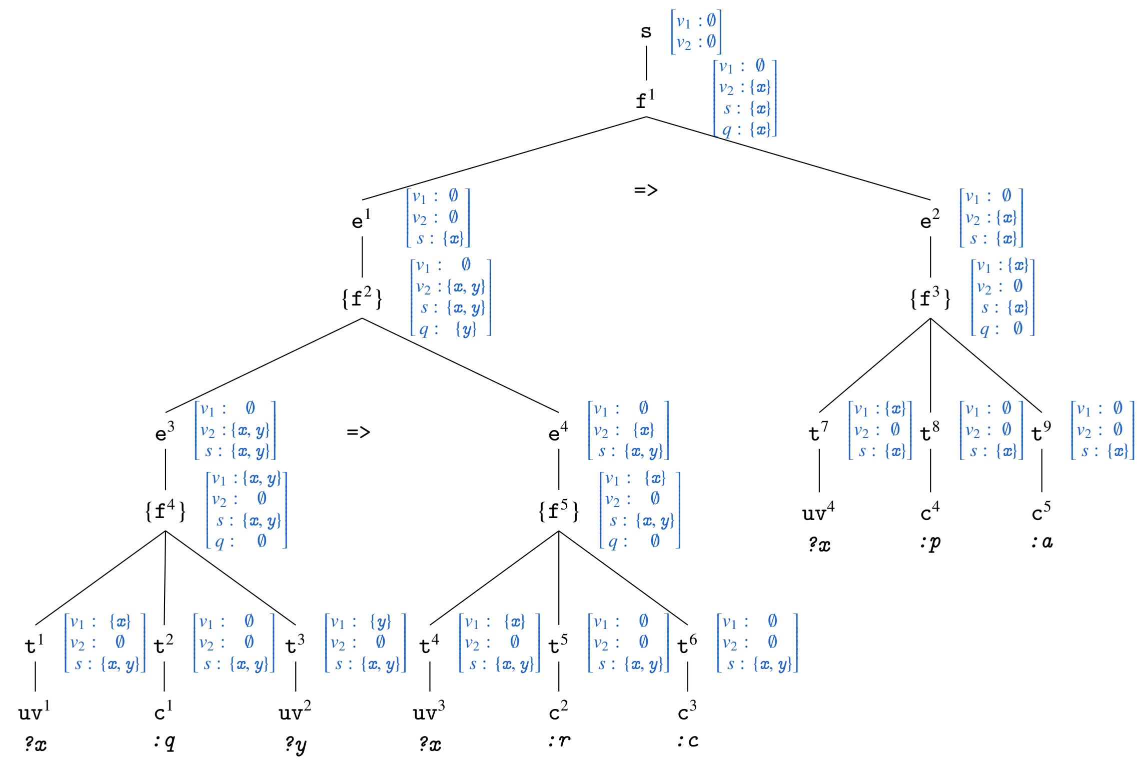
s [v1 : ∅, v2 : ∅]
f1 [v1 : ∅, v2 : {x}, s : {x}, q : {x}]
=>
e1 [v1 : ∅, v2 : ∅, s : {x}]
{f2} [v1 : ∅, v2 : {x, y}, s : {x, y}, q : {y}]
e2 [v1 : ∅, v2 : {x}, s : {x}]
{f3} [v1 : {x}, v2 : ∅, s : {x}, q : ∅]
e3 [v1 : ∅, v2 : {x, y}, s : {x, y}]
=>
e4 [v1 : ∅, v2 : {x}, s : {x, y}]
{f4} [v1 : {x, y}, v2 : ∅, s : {x, y}, q : ∅]
{f5} [v1 : {x}, v2 : ∅, s : {x, y}, q : ∅]
t7 [v1 : {x}, v2 : ∅, s : {x}]
t8 [v1 : ∅, v2 : ∅, s : {x}]
t9 [v1 : ∅, v2 : ∅, s : {x}]
uv4 ?x
c4 :p
c5 :a
t1 [v1 : {x}, v2 : ∅, s : {x, y}]
t2 [v1 : ∅, v2 : ∅, s : {x, y}]
t3 [v1 : {y}, v2 : ∅, s : {x, y}]
t4 [v1 : {x}, v2 : ∅, s : {x, y}]
t5 [v1 : ∅, v2 : ∅, s : {x, y}]
t6 [v1 : ∅, v2 : ∅, s : {x, y}]
uv1 ?x
c1 :q
uv2 ?y
uv3 ?x
c2 :r
c3 :c

Figure 8: Syntax tree of Formula 16 with attribute values for the attributes v_1, v_2, s and q (in *blue*).

CFG		Synthesized attributes		Inherited attributes	
production rules		rules for v_1	rules for v_2	rules for s	rules for q
`s ::=`	`f`	$\texttt{s}.v_1 \leftarrow \emptyset$	$\texttt{s}.v_2 \leftarrow \texttt{f}.v_1$	$\texttt{f}.s \leftarrow \texttt{f}.v_1 \cup \texttt{f}.v_2$	$\texttt{f}.q \leftarrow \texttt{f}.v_1 \cup \texttt{f}.v_2$
`f ::=`	$\texttt{t}_1\texttt{t}_2\texttt{t}_3$.	$\texttt{f}.v_1 \leftarrow \texttt{t}_1.v_1 \cup \texttt{t}_2.v_1 \cup \texttt{t}_3.v_1$	$\texttt{f}.v_2 \leftarrow \texttt{t}_1.v_2 \cup \texttt{t}_2.v_2 \cup \texttt{t}_3.v_2$	$\texttt{t}_i.s \leftarrow \texttt{f}.s$	
	$\texttt{e}_1$=>$\texttt{e}_2$.	$\texttt{f}.v_1 \leftarrow \texttt{e}_1.v_1 \cup \texttt{e}_2.v_1$	$\texttt{f}.v_2 \leftarrow \texttt{e}_1.v_2 \cup \texttt{e}_2.v_2$	$\texttt{e}_i.s \leftarrow \texttt{f}.s$	
	$\texttt{f}_1\texttt{f}_2$	$\texttt{f}.v_1 \leftarrow \texttt{f}_1.v_1 \cup \texttt{f}_2.v_1$	$\texttt{f}.v_2 \leftarrow \texttt{f}_1.v_2 \cup \texttt{f}_2.v_2$	$\texttt{f}_i.s \leftarrow \texttt{f}.s$	$\texttt{f}_i.q \leftarrow \emptyset$
`t ::=`	`uv`	$\texttt{t}.v_1 \leftarrow \{\texttt{uv}\}$	$\texttt{t}.v_2 \leftarrow \emptyset$		
	`ex`	$\texttt{t}.v_1 \leftarrow \emptyset$	$\texttt{t}.v_2 \leftarrow \emptyset$		
	`c`	$\texttt{t}.v_1 \leftarrow \emptyset$	$\texttt{t}.v_2 \leftarrow \emptyset$		
	`e`	$\texttt{t}.v_1 \leftarrow \texttt{e}.v_1$	$\texttt{t}.v_2 \leftarrow \texttt{e}.v_2$	$\texttt{e}.s \leftarrow \texttt{t}.s$	
	`(k)`	$\texttt{t}.v_1 \leftarrow \texttt{k}.v_1$	$\texttt{t}.v_2 \leftarrow \texttt{k}.v_2$	$\texttt{k}.s \leftarrow \texttt{t}.s$	
	`()`	$\texttt{t}.v_1 \leftarrow \emptyset$	$\texttt{t}.v_2 \leftarrow \emptyset$		
`k ::=`	`t`	$\texttt{k}.v_1 \leftarrow \texttt{t}.v_1$	$\texttt{k}.v_2 \leftarrow \texttt{t}.v_2$	$\texttt{t}.s \leftarrow \texttt{k}.s$	
	$\texttt{t k}_1$	$\texttt{k}.v_1 \leftarrow \texttt{t}.v_1 \cup \texttt{k}_1.v_1$	$\texttt{k}.v_2 \leftarrow \texttt{t}.v_2 \cup \texttt{k}_1.v_2$	$\texttt{t}.s \leftarrow \texttt{k}.s$ $\texttt{k}_1.s \leftarrow \texttt{k}.s$	
`e ::=`	`{f}`	$\texttt{e}.v_1 \leftarrow \emptyset$	$\texttt{e}.v_2 \leftarrow \texttt{f}.v_1$	$\texttt{f}.s \leftarrow \texttt{e}.s \cup \texttt{f}.v_2$	$\texttt{f}.q \leftarrow \texttt{f}.v_2 \setminus \texttt{e}.s$
	`{}`	$\texttt{e}.v_1 \leftarrow \emptyset$	$\texttt{e}.v_2 \leftarrow \emptyset$		
	`false`	$\texttt{e}.v_1 \leftarrow \emptyset$	$\texttt{e}.v_2 \leftarrow \emptyset$		

Figure 9: Attribute rules for universal quantification in Cwm defined on the context-free grammar of N3 (Figure 2).

the parent$_c$ of the last occurrence of ?x is the formula as a whole, the parent$_c$ of ?y is the subformula

```
{?x :q ?y.} => {?x :r :c.}.
```

or, if we take a look into the formula's syntax tree displayed in Figure 8, the parent$_c$ of the last ?x (uv^4) is f^1 and the parent$_c$ of ?y (uv^2) is f^2. Therefore formula f^1 carries a universal quantifier for ?x and f^2 carries a universal quantifier for ?y. The first universal quantifier also covers the other occurrences of ?x since the parent$_c$ formula of these other occurrences, namely f^2, is already in scope of this quantifier. Formula 16 means:

$$\forall \texttt{x}.\ \ <\forall \texttt{y}.\ \ <\texttt{x q y}> \rightarrow <\texttt{x r c}>\ > \rightarrow <\texttt{x p a}> \quad (16')$$

Similarly to the example of existential quantification, we define attributes which pass universal variables occurring under a parent$_c$ formula up to that parent$_c$ formula. For this purpose, we use two attributes whose values are as follows:

v_1 the set of universal variables occurring as direct components under a node.

v_2 the set of universal variables occurring as direct components of direct components (parent level).

The first attribute v_1 is used to pass universal attributes up to their direct formulas, the second attribute v_2 is used to further pass these variables from the direct formulas to the parent$_c$ formulas. The attributes are again synthesized and we have one attribute rule for each production rule. These attribute rules are displayed in the second and third column of the table in Figure 9. The first column of the table shows the corresponding production rules. To better explain the attributes, we apply them to the syntax tree of Formula 16. The tree and the corresponding attribute values are displayed in Figure 8.

Passing Variables to their Direct Formula

Attribute v_1 works the exact same way as attribute *eq* with the only difference that at term level the universals get captured in a singleton set (t.$v_1 \leftarrow$ {uv}) instead of the existentials. These universals are then passed upwards to their parent$_c$ formulas. In our example, the formulas f^1 and f^2 do not have universal variables as direct components, f^3 has ?x, f^4 has ?x and ?y, and formula f^5 has ?x.

Passing Variables to their Parent Formula

Attribute v_2 now passes the universal variables which are collected under their direct formula using v_1 to their parent$_c$ formula. This attribute works in a similar way as the previous one: at the level where the universal variables of a direct formula f under a node e are found the rule for the synthesized attribute takes these values gathered by the attribute v_1 and passes them upwards till the next formula being direct descendant of a node s or e is encountered. This formula is then the parent$_c$.

We display the rules for attribute v_2 in the third column of Figure 9. We again explain the rules by going through the syntax tree in Figure 8 from the bottom to the top. The rules only resulting in symbols of the alphabet cannot have direct components. The attribute rule t.$v_2 \leftarrow \emptyset$ for the production rules t ::= uv and t ::= c assign the empty set to the term nodes t^1 to t^9. These values are passed upwards via the attribute rule f $\leftarrow$ t$_1$.v_2 $\cup$ t$_2$.v_2 $\cup$ t$_3$.v_2 on production rule f ::= t$_t$t$_2$t$_3$, the attribute values for f^3, f^4 and f^5 are again the empty set. The attribute rule for the next higher level, the production e ::= {f} is more interesting. The variables which occur as direct components on the formula node f and are captured by the attribute v_1 are passed as children of the next parent$_c$ formula to the node e via the attribute rule e.$v_2 \leftarrow$ f.v_1. We get: e^2.v_2 = {?x}, e^3.v_2 = {?x, ?y} and e^4.v_2 = {?x}. For f^2 these values are passed upwards via the attribute rule f $\leftarrow$ e$_1$ $\cup$ e$_2$. As f^2 is a direct descendant of the node e^1, it is the parent$_c$ formula of the variables mentioned above. These variables are not passed further through. Instead, the attribute value for e^1 is again taken from the attribute v_1 (via e.$v_2 \leftarrow$ f.v_1) which in this case is the empty set. With these values we can determine the attribute value for f^1 (via f $\leftarrow$ e$_1$ $\cup$ e$_2$) which in this case is the singleton set only containing the variable ?x. The attribute value for s is the empty set.

For formulas which are direct descendants of a node s or e the value is now the set of universal variables for which the formula is a parent$_c$: for f^1 that is {?x}, for f^2 it is {?x, ?y}, and for f^3, f^4 and f^5 the value is the empty set since these formulas are not parent$_c$ formulas.

Passing Scoped Variables to the Descendants

With the result from above that formula f^1 is the parent$_c$ of ?x and f^2 the parent$_c$ of ?x and ?y we take again a look to interpretation of Formula 16: knowing that universal variables are quantified on their parent$_c$ formula we could expect one universal quantifier for x on formula f^1 another universal quantifier for x on formula f^2 and a third universal quantifier for y on the same formula. If we go back to the interpretation given in Formula 16' we only count two universal quantifiers: one for x on f^1 and one for y on f^2. The reason for that is that, according to Cwm's interpretation, the first universal quantifier for x already covers all other occurrences of the universal variable ?x in the formula regardless of their level of nesting. As a consequence of that behaviour, adding another conjunct

```
:s :p {:a :b ?y.}.
```

to our formula also changes the meaning of the latter. The formula

```
{{?x :q ?y.} => {?x :r :c.}.}=>
                {?x :p :a.}.
                    :s :p {:a :b ?y.}.   (17)
```

means

$$\forall x.\forall y.(< <x\ q\ y> \rightarrow <x\ r\ c> > \rightarrow <x\ p\ a>.\ s\ p\ <a\ b\ y>.\ ) \quad (17')$$

Note that the quantifier for y which is nested in the interpretation of Formula 16 is on top level in the interpretation of Formula 17. This is the case because the conjunction, ie the formula as a whole, is the parent_c formula of the second occurrence of ?y and thus carries a quantifier which also covers the nested occurrence of ?y.

Attribute s keeps track of this behaviour. The value of s is for each node the set of variables which are universally quantified on the node, either by a quantifier on the node itself, or by a quantifier on a higher level. This kind of information needs to be passed downwards in the syntax tree therefore attribute s is inherited. As the value of s is only relevant for potential parent_c formulas, we only define the attribute for the node f and all nodes which can occur above f in the syntax tree, These are the nodes f, t, e and k. As s is inherited, this time we need to define an attribute rule for each *right-side* occurrence of s on a production rule. These rules are displayed in the fourth column of Figure 9. We explain them by going through the syntax tree in Figure 8.

For the occurrence of f in the rule s ::= f we take all the variables of which f is the parent_c formula since these are quantified on that highest level. Via the attribute rule $\texttt{f}.c \leftarrow \texttt{f}.v_1 \cup \texttt{f}.v_2$[11] we get: $\texttt{f}^1.s = \{\texttt{?x}\}$. For the nodes $\texttt{e}^1$ and $\texttt{e}^2$ this information is passed downwards, via $\texttt{e}_1.s \leftarrow \texttt{f}.s$ and $\texttt{e}_2.s \leftarrow \texttt{f}.s$ we get the value $\{\texttt{?x}\}$ for both nodes. On the production rule e ::= {f} the nodes e can now again be the direct ancestor of a parent_c formula f. The set of variables $\texttt{f}.v_2$ the formula f is parent_c of are either quantified on that same formula or they are already quantified beforehand in which case they are already present in $\texttt{e}.s$. In both cases the union of these values covers all variables quantified at that point, we have as attribute rule $\texttt{f}.s \leftarrow \texttt{e}.s \cup \texttt{f}.v_2$. We get $\texttt{f}^2.s = \{\texttt{?x}, \texttt{?y}\}$ and $\texttt{f}^3.s = \{\texttt{?x}\}$. This information is passed further downwards via the attribute rules $\texttt{t}_{1,2,3}.s \leftarrow \texttt{e}.s$. for the production rule $\texttt{f ::= t}_1\texttt{t}_2\texttt{t}_3$ and via the rules $\texttt{e}_{1,2}.s \leftarrow \texttt{f}.s$ for $\texttt{f ::= e}_1\texttt{e}_2$, we get $\texttt{t}^{7,8}.s = \{\texttt{?x}\}$ and $\texttt{e}^{3,4}.s = \{\texttt{?x}, \texttt{?y}\}$. The descendants of these last two expressions are not parent_c formulas, the attribute thus simply passes their values down to $\texttt{f}^4$ and $\texttt{f}^5$ which then get passed further to the nodes $\texttt{t}^1$ to $\texttt{t}^6$. For all nodes in our syntax tree for which the attribute s is defined we now capture the set of variables which are scoped under that node.

Determining the Universally Quantified Variables for a Formula

In order to use the information captured by the previous attributes one step is missing: we still need to determine the exact set of universal variables quantified on a specific formula. For this we define attribute q: for each formula node f the value of q is the set of universal variables for which f carries a quantifier in the translation. We define q as an inherited attribute on f. The rules for q are listed in the last column of Figure 9.

If f occurs under the starting node, all variables it is parent_c of are quantified on that formula. For the rule s ::= f the value of the attribute is the same as the value of v_2: $\texttt{f}.q = \texttt{f}.v_1 \cup \texttt{f}.v_2$.[12] In our syntax tree in Figure 8 we get $\texttt{f}^1.q = \{\texttt{?x}\}$. If a formula is only a conjunct of a conjunction, it does not carry any universal quantifier (remember for example Formula 17). Therefore the attribute value assigned to $\texttt{f}_1$ and $\texttt{f}_2$ on the production rule $\texttt{f ::= f}_1\texttt{f}_2$ is the empty set: $\texttt{f}_{1,2} \leftarrow \emptyset$. For the third rule with a right-side occurrence of f, e ::= {f}, we take the values of the attributes v_2 and q into account: the value of s on the node e contains all the universal variables which are already quantified on that or a

[11] Note that here attribute s does not only collect all universal variables f is parent_c of but also the universals occurring as direct components. The reason for that is that these variables cannot be passed further upwards. In the original N3 specification, implicitly universally quantified variables on top level are allowed according to the grammar, but their meaning is not covered in the semantics Cwm applies. We handle this problem by adding quantifiers for them to the top level.

[12] As above, this attribute does not only carry the universal variables f is parent_c of but also those which occur directly in the formula.

higher node. The value of v_2 on `f` contains all variables which need to be quantified above or at `f` since `f` is their parent node. The set of universal variables for which `f` needs to carry a quantifier is the difference of these two sets: $\texttt{f}.q \leftarrow \texttt{f}.v_2 \setminus \texttt{e}.s$. For the node $\texttt{f}^2$ in our syntax tree we get $\texttt{f}^2.q = \{\texttt{?x}, \texttt{?y}\} \setminus \{\texttt{?x}\} = \{\texttt{?y}\}$. For the remaining nodes $\texttt{f}^3$, $\texttt{f}^4$ and $\texttt{f}^5$ the attribute value is the empty set. And it is indeed the case that the formula $\texttt{f}^1$ carries a universal quantifier for `?x` and formula $\texttt{f}^3$ for `?y`. The only thing which still needs to be done to obtain Cwm's interpretation of the formula is to construct the concrete translation. The attribute to perform this task is defined below in Section 4.3.4.

4.3.3. Universals in EYE

After having defined several attributes to handle implicit universal quantification according to Cwm in the previous section, we do the same for the reasoner EYE in this section. EYE understands the term *parent formula* from the W3C team submission as the top formula. Universal variables are thus for EYE always quantified on the top level. Formula 16 means according to EYE:

$$\begin{aligned}&\forall \texttt{x}.\forall \texttt{y}.\\ &\quad \texttt{< <x q y>} \rightarrow \texttt{<x r c> >} \rightarrow \texttt{<x p a>}\end{aligned} \tag{16''}$$

To deal with universal quantification we therefore only need one attribute that passes all universal variables occurring in a formula to the top level. We define the synthesized attribute u for all nodes of the grammar. The value of u for a node is always the set of universal variables occurring anywhere under that node. The attribute rules for u are displayed in Figure 10. As the calculation of u's attribute values is rather simple, we do not discuss these rules in detail and only capture for further considerations the value of the top formula $\texttt{f}^1.u = \{\texttt{?x}, \texttt{?y}\}$.

4.3.4. Generation of the translation

In order to obtain the different translations from N3 to core logic, one last step is needed: the core logic formulas need to be generated. We use synthesized attributes to perform this task. To clarify the difference between the signs used in the core formula produced and the logical symbols we use to describe this production, we underline all terminal symbols belonging to the core logic.

We start by defining the following auxiliary functions which add quantifiers to any set of symbols of the alphabet:

Let $ex : 2^{\mathcal{A}} \rightarrow 2^{\mathcal{A}^*}$ be defined as:

$$ex(V) := \{\underline{\exists v_1}.\ldots\underline{\exists v_n}.|v_i \in V; 1 \leq i \leq n = |V|; \quad i \neq j \Rightarrow v_1 \neq v_j\}.$$

production rule		attribute rule
`s ::=`	`f`	$\texttt{s}.u \leftarrow \texttt{f}.u$
`f ::=`	$\texttt{t}_1\texttt{t}_2\texttt{t}_3.$	$\texttt{f}.u \leftarrow \texttt{t}_1.u \cup \texttt{t}_2.u \cup \texttt{t}_3.u$
	$\texttt{e}_1\texttt{=>e}_2.$	$\texttt{f}.u \leftarrow \texttt{e}_1.u \cup \texttt{e}_2.u$
	$\texttt{f}_1\texttt{f}_2$	$\texttt{f}.u \leftarrow \texttt{f}_1.u \cup \texttt{f}_2.u$
`t ::=`	`uv`	$\texttt{t}.u \leftarrow \{\texttt{uv}\}$
	`ex`	$\texttt{t}.u \leftarrow \emptyset$
	`c`	$\texttt{t}.u \leftarrow \emptyset$
	`e`	$\texttt{t}.u \leftarrow \texttt{e}.u$
	`(k)`	$\texttt{t}.u \leftarrow \texttt{k}.u$
	`()`	$\texttt{t}.u \leftarrow \emptyset$
`k ::=`	`t`	$\texttt{k}.u \leftarrow \texttt{t}.u$
	$\texttt{t k}_1$	$\texttt{k}.u \leftarrow \texttt{t}.u \cup \texttt{k}_1.u$
`e ::=`	`{f}`	$\texttt{e}.u \leftarrow \texttt{f}.u$
	`{}`	$\texttt{e}.u \leftarrow \emptyset$
	`false`	$\texttt{e}.u \leftarrow \emptyset$

Figure 10: Attribute rules for the synthesized attribute u (right) and their corresponding production rules (left) from the N3 grammar (Figure 2).

Let $uv : 2^{\mathcal{A}} \rightarrow 2^{\mathcal{A}^*}$ be defined as:

$$uv(V) := \{\underline{\forall v_1}.\ldots\underline{\forall v_n}.|v_i \in V; 1 \leq i \leq n = |V|; \quad i \neq j \Rightarrow v_1 \neq v_j\}.$$

The range of these two functions are sets, for $\{\mathrm{x}, \mathrm{y}\}$ we get for example:

$$ex(\{\mathrm{x}, \mathrm{y}\}) = \{\underline{\exists \mathrm{x}.\exists \mathrm{y}.}, \underline{\exists \mathrm{y}.\exists \mathrm{x}.}\}.$$

To be able to use the functions to construct from a set of variables a sequence of quantified variables we thus need a selection function. Let $select : 2^{T^*} \rightarrow T^*$ be such a function which, given a set, chooses one element of that set. We use the notation $\dot{ex}$ and $\dot{uv}$ to denote $select \circ ex$ and $select \circ uv$, respectively. For the empty set the selection function returns the empty string.

With the help of these functions we can define attributes which take the signs of the N3 formula, replace them by the respective sign of the core logic, and add, where necessary, explicit quantifiers. We use two different attributes:

m_c The value of m_c is the translation of the symbols of the alphabet occurring under a node according to Cwm.

CFG production rules		Cwm rules for m_c	EYE rules for m_e
`s ::=`	`f`	$\texttt{s}.m_c \leftarrow \dot{uv}(\texttt{f}.q)\ \dot{ex}(\texttt{f}.\text{eq})\ \texttt{f}.m_c$	$\texttt{s}.m_e \leftarrow \dot{uv}(\texttt{f}.u)\ \dot{ex}(\texttt{f}.\text{eq})\ \texttt{f}.m_e$
`f ::=`	$\texttt{t}_1\texttt{t}_2\texttt{t}_3.$	$\texttt{f}.m_c \leftarrow \texttt{t}_1.m_c\ \texttt{t}_2.m_c\ \texttt{t}_3.m_c$	$\texttt{f}.m_e \leftarrow \texttt{t}_1.m_e\ \texttt{t}_2.m_e\ \texttt{t}_3.m_e$
	$\texttt{e}_1\texttt{=>}\texttt{e}_2.$	$\texttt{f}.m_c \leftarrow \texttt{e}_1.m_c\ \underline{\rightarrow}\ \texttt{e}_2.m_c$	$\texttt{f}.m_e \leftarrow \texttt{e}_1.m_e\ \underline{\rightarrow}\ \texttt{e}_2.m_e$
	$\texttt{f}_1\texttt{f}_2$	$\texttt{f}.m_c \leftarrow \texttt{f}_1.m_c\ \texttt{f}_2.m_c$	$\texttt{f}.m_e \leftarrow \texttt{f}_1.m_e\ \texttt{f}_2.m_e$
`t ::=`	`uv`	$\texttt{t}.m_c \leftarrow \underline{\texttt{uv}}$	$\texttt{t}.m_e \leftarrow \underline{\texttt{uv}}$
	`ex`	$\texttt{t}.m_c \leftarrow \underline{\texttt{ex}}$	$\texttt{t}.m_e \leftarrow \underline{\texttt{ex}}$
	`c`	$\texttt{t}.m_c \leftarrow \underline{\texttt{c}}$	$\texttt{t}.m_e \leftarrow \underline{\texttt{c}}$
	`e`	$\texttt{t}.m_c \leftarrow \texttt{e}.m_c$	$\texttt{t}.m_e \leftarrow \texttt{e}.m_e$
	`(k)`	$\texttt{t}.m_c \leftarrow \underline{(}\texttt{k}.m_c\underline{)}$	$\texttt{t}.m_e \leftarrow \underline{(}\texttt{k}.m_e\underline{)}$
	`()`	$\texttt{t}.m_c \leftarrow \underline{\texttt{<>}}$	$\texttt{t}.m_e \leftarrow \underline{\texttt{<>}}$
`k ::=`	`t`	$\texttt{k}.m_c \leftarrow \texttt{t}.m_c$	$\texttt{k}.m_e \leftarrow \texttt{t}.m_e$
	$\texttt{t k}_1$	$\texttt{k}.m_c \leftarrow \texttt{t}.m_c\ \texttt{k}_1.m_c$	$\texttt{k}.m_e \leftarrow \texttt{t}.m_e\ \texttt{k}_1.m_e$
`e ::=`	`{f}`	$\texttt{e}.m_c \leftarrow \underline{<}\dot{uv}(\texttt{f}.q)\ \dot{ex}(\texttt{f}.\text{eq})\ \texttt{f}.m_c\underline{>}$	$\texttt{e}.m_e \leftarrow \underline{<}\dot{ex}(\texttt{f}.\text{eq})\ \texttt{f}.m_e\underline{>}$
	`{}`	$\texttt{e}.m_c \leftarrow \underline{\texttt{<>}}$	$\texttt{e}.m_e \leftarrow \underline{\texttt{<>}}$
	`false`	$\texttt{e}.m_c \leftarrow \underline{\texttt{false}}$	$\texttt{e}.m_e \leftarrow \underline{\texttt{false}}$

Figure 11: Attribute rules for constructing the translation of a formula into core logic according to Cwm (m_c) and according to EYE (m_e).

m_e The value of m_e is the translation of the symbols of the alphabet occurring under a node according to EYE.

Both attributes are synthesized and defined for all symbols of the grammar. In Figure 11 we display the corresponding attribute rules. In most cases these rules look very similar: constants, existentials and universals are concatenated, the symbols `{` and `}` are replaced by < and >, and => by →. A different behaviour of the attributes can only be observed at the two places where quantifiers are added, the rules `e ::= {f}` and `s ::= f`.

For the first of these production rules, the rule for the attribute m_c adds quantifiers to the translated subformula. To get all universally quantified variables at that level, the value of the attribute q (universal variables for a formula according to Cwm) is used. For existentially quantified variables we use the value of eq (existential variables for a formula). As explained in Section 2.3 the translation puts first the universal and then the existential quantification. In contrast to that, the rule for the attribute m_e only adds the existential quantifier since according to its understanding, universal quantifiers are only set in front of the top formula.

For the second of these production rules, the starting rule, attribute m_c behaves as before: universal and existential quantifiers are set using the values of q and eq. The rules for attribute m_e also add universal quantifiers at that highest level: the quantifiers for the universal variables collected using u, the attribute gathering all universal variables occurring under a formula.

With these rules the quantifiers for a formula are only added when it is sure that the formula stands on its own and is not part of a conjunction. To understand the reason for that behaviour, recall the example in Formula 17: there, the universal quantifier for `y` caused by the second conjunct also counted for the first and needed thus to stand in front of both formulas. In case `f` is produced using the last rule `s ::= f` we know that there are no more conjuncts for the formula and the quantifiers can be added. For each formula, the translation to core logic generated using the attributes m_c and m_e is the attribute value of $\texttt{s}.m_c$, respectively $\texttt{s}.m_e$.

We finish this section by an example: we again consider the syntax tree of Formula 16 displayed in Figure 8, this time to construct the translations using the attributes m_c and m_e. To improve readability, we omit the prefixes for the constants (“`:`”) and the introducing question-marks (“?”) for universal variables. Going from the bottom to the top of the tree, the first steps are the same for m_c and m_e and very easy to understand: the symbols are just captured, translated where needed and then concatenated. For $\texttt{f}^4$ we get for example:

$$\texttt{f}^4.m_{c,e} \leftarrow \underline{\texttt{x q y}}$$

On the next higher level, both attributes do not add quantifiers:

$$\begin{aligned} \texttt{e}^3.m_c \leftarrow \quad & \underline{<}\dot{u}v(\texttt{f}^4.q)\dot{e}x(\texttt{f}^4.\text{eq})\underline{\texttt{x q y>}} \\ & =\underline{<}\dot{u}v(\emptyset)\dot{e}x(\emptyset)\underline{\texttt{x q y>}} = \underline{\texttt{<x q y>}} \end{aligned}$$

and

$$\texttt{e}^3.m_e \leftarrow \quad \underline{<}\dot{e}x(\texttt{f}^4.\text{eq})\underline{\texttt{x q y>}} = \underline{\texttt{<x q y>}}$$

This is also the case for $\texttt{e}^2$ and $\texttt{e}^4$. As the attributes work analogously for $\texttt{e}^1$ with the only difference that at that level the attribute rule for m_c adds a universal quantifier as the value $\texttt{f}^2.q$ is not empty, we also omit these values and take a closer look to the attribute values at s. For $\texttt{f}^1$ we have

$$\texttt{f}^1.m_c \leftarrow \underline{\texttt{<}\forall\texttt{y.<x q y>} \rightarrow \texttt{<x r c>>} \rightarrow \texttt{<x p a>}}$$

And get

$$\begin{aligned} \texttt{s}.m_c \leftarrow & \ \dot{u}v(\texttt{f}^1.q)\dot{e}x(\texttt{f}^1.\text{eq})\texttt{f}^1.m_c \\ & = \dot{u}v(\{\text{x}\})\dot{e}x(\emptyset)\texttt{f}^1.m_c \\ & = \underline{\forall\texttt{x.<}\forall\texttt{y.<x q y>} \rightarrow \texttt{<x r c>>} \rightarrow \texttt{<x p a>}} \\ & = \text{Formula 16'} \end{aligned}$$

And with

$$\texttt{f}^1.m_e \leftarrow \underline{\texttt{<<x q y>} \rightarrow \texttt{<x r c>>} \rightarrow \texttt{<x p a>}}$$

we get:

$$\begin{aligned} \texttt{s}.m_e \leftarrow & \ \dot{u}v(\texttt{f}^1.u)\dot{e}x(\texttt{f}^1.\text{eq})\texttt{f}^1.m_e \\ & = \dot{u}v(\{\text{x}, \text{y}\})\dot{e}x(\emptyset)\texttt{f}^1.m_e \\ & = \underline{\forall\texttt{x.}\forall\texttt{y.<<x q y>} \rightarrow \texttt{<x r c>>} \rightarrow \texttt{<x p a>}} \\ & = \text{Formula 16''} \end{aligned}$$

The attributes deliver the expected result.

5. Evaluation

In the previous sections we specified how existing interpretations of formulas in N3Logic differ in their handling of implicit quantification. Here, we take a closer look at these differences. How do they impact practical cases? In which kinds of applications are they relevant? In order to answer these questions, we implemented the attribute grammar introduced above and tested for several formulas whether the two interpretations we formalised differ on these examples. An explanation of the implementation, the datasets used and the results of our tests – quantitative and qualitative – is given below.

5.1. Implementation

For our evaluation we have implemented the attribute grammar as specified above. In order to stay close to our format, we used the Utrecht University Attribute Grammar (UUAG) and its compiler the Utrecht University Attribute Grammar Compiler (UUAGC) [19]. This framework enables the user to specify attribute grammars which then get translated to Haskell code and can be used in all kinds of applications. All additional applications and functions were written in Haskell.

For every N3 formula, our program produces the syntax tree of the translated core logic formula as well as its string representation in our two interpretations. We furthermore implemented a function which compares these translations. Note that in N3, especially in Cwm's interpretation, every file needs to be treated as one formula. This is because the conjunction is done by putting triples after each other. It makes a difference whether

```
{:a :b {:c :d ?x}} => {:e :f :g}.
```

is followed by

`:s :p {?x :p :o}.` or `:s :p {?y :p :o}.`

In the first case Cwm's interpretation puts the quantifier for the variable `?x` on the top formula; in the second case it is inside the premise of the rule. We therefore cannot give the meaning of the first implication without taking its context into account. Our function thus always compares the meaning of an entire file and then displays the concrete differences between interpretations. All code can be accessed at `https://github.com/IDLabResearch/N3CoreLogic`.

5.2. Datasets used

To test whether the differences described above can be observed in practical applications, we used two kinds of datasets: a test dataset of the reasoner EYE and several datasets used in previous research projects.

The *EYE dataset*[13] is a collection of test cases for the EYE reasoner. Some of the tests were created to challenge the reasoner (e.g. parsing of nested expressions, scoping of blank nodes and universals) and are therefore rather artificial, but the majority of test files was either sent by users of the reasoner to explain problems they had or are minimal examples of practical use cases from different parties. In that sense the content of the

[13]Accessible at `https://github.com/josd/eye/tree/master/reasoning/`. Our tests are based on the version of October 15, 2017.

dataset reflects a big variety of applications created by different users. At the moment we tested, the dataset contained 359 N3 files in 50 folders of which 303 contained implicitly quantified universal variables. As the latest version of EYE does not support that feature, the test cases do not include explicit quantification.

The *Project datasets* contain rules we specified in previous projects, in particular: the projects *ORCA (Ontology based Reasoning for nurse Call)* [20, 21], *Facts4Workers (Factories for Workers)* [22] and *DiSSeCt (Distributed Semantic software solutions for complex Service Composition)* [23]. The rules of the *ORCA dataset*[14] are written to perform an optimized version of OWL-RL reasoning and to follow a complex decision tree which, depending on the set-up and the current situation of a hospital, assign the best staff member to answer a patient call. The aim of the rules from the *Facts4Workers* use-case[15] is to deal with the diverse infrastructure of modern factories in which different machines are able to perform a huge variety of tasks. These tasks are described via rules which can be combined to fulfil a desired goal. The last set of rules[16], used in the project *DiSSeCt*, is designed to check RDF datasets for user-specified constraints. We chose these datasets because they were produced to be used in practical rather complex applications and not to merely test the reasoner. To ensure compatibility with EYE the datasets do not make use of explicit quantification.

For our tests, we selected only the files which contain universal variables. For the ORCA dataset this selection contained 130 files, for Facts4Workers 25 files, and for DiSSeCt 84 files.

5.3. Results

For the datasets introduced above, we tested whether the reasoners interpret every file in the exact same way, ie whether the two core logic translations produced by our software were the same (accepting differences in the naming of variables and in the order of universal and of existential quantifiers on the same level of a formula), or whether we can identify differences. The results are displayed in Table 2. We see that every dataset contains files for which the interpretation differs between Cwm and EYE. Nevertheless, the portion of affected files varies per dataset and depends on the nature of the data. We have identified three kinds of constructs which can be related with disagreements in the interpretation:

dataset	#files	#differences	percentage
EYE	303	81	27%
F4W	25	12	48%
ORCA	130	27	21%
DiSSeCt	84	50	60%
Total	542	179	31%

Table 2: Results of tests for differences in the interpretations of N3 files. By #differences we mean the number of files which are interpreted differently by the two reasoners.

Proofs formulas which represent a *proof*, a special kind of N3 formulas explaining the derivation steps of the reasoner (see also [24]);

Built-ins formulas which contain built-in functions which have a special meaning for one or both reasoners;

Nesting formulas which, without using built-in functions, either act on graph patterns or perform reasoning about rules.

Before taking a closer look at the distribution of these three cases in the different datasets, we explain what we mean by *proofs*, *built-ins* and *nesting* in more detail.

5.3.1. Proofs

When drawing conclusions from data, both reasoners are able to provide an explanation for their derivations. For that they both use the N3 proof vocabulary[17] created in the context of the Semantic Web Application Platform (SWAP) [25]. The vocabulary makes it possible to express proof steps performed by the reasoners. While Cwm proofs only contain explicit universal quantification and are therefore out of scope for our current evaluation, EYE proofs employ implicit universal quantification. An example proof step is given in Listing 1.

We see the proof step of an `r:Extraction`. This corresponds to conjunction elimination from common first-order calculi: if a bigger conjunction is known to be correct, so are its conjuncts. Here, this example step is based on another step, a `r:Parsing`, ie reading information from a file. The proof steps yield the formula

```
{:s :p ?x1} => {:s :pp ?x1}.        (18)
```

[14]This project was in cooperation with an industry partner and the results might be used in a product. The rules are therefore not public.

[15]Available at `https://github.com/IDLabResearch/Facts4Workers/tree/master/n3`.

[16]Available at `https://github.com/IDLabResearch/data-validation`.

[17]`https://www.w3.org/2000/10/swap/reason#`

```
@prefix  :  <http://example.org/ex#>.
@prefix r:  <http://www.w3.org/2000/10/swap/reason#>.

 <#lemma1> a r:Extraction;
  r:gives {
    {:s :p ?x1} => {:s :pp ?x1}.
  };
  r:because [ a r:Parsing ].
```

Listing 1: Representation of a proof step.

indicated by the predicate `r:gives`. Interesting from a structural point of view is that the formula appears in a formula expression. For Cwm the variable `?x1` is therefore universally quantified in this expression. We get the interpretation:

$$\texttt{<L1> gives <}\forall\texttt{x. <s p x>} \rightarrow \texttt{<s pp x> >.}$$

Since the file from which the formula stems contains Formula 18 this interpretation is correct in this context. For EYE all variables are quantified on top level, we get:

$$\forall\texttt{x. <L1> gives < <s p x>} \rightarrow \texttt{<s pp x> >.}$$

Yet, if we consider the meaning of the predicate `r:gives` which indicates the consequences we can draw from the reasoning steps, then this interpretation is also right: from the fact that Formula 18 appears in our data we can for every `x` conclude that `< <s p x>` → `<s pp x> >`. Hence in this case the differences in the interpretations do not have practical consequences even if the proofs are used for further reasoning.

N3 reasoning mostly depends on rules containing universal variables. Like in the example, most proofs list the parsing and selection of such rules and are therefore often subject to the problem described. In our datasets, this is the case for all proofs.

5.3.2. Built-ins

Another big group of differences in the interpretations of a formula can be observed in connection with built-in functions. Both reasoners, Cwm and EYE, provide a set of predicates with predefined meanings[18] which can be used to, for example, deal with lists (`rdf:first`), to compare terms (`log:equalTo`) or to do calculations (`math:product`). Some built-in predicates are reasoner-specific. Using these leads to different reasoning results. But even if a predicate is supported by different reasoners, we often get differences in connection with their usage. This is because many built-ins deal with graphs or graph patterns.

As an example consider the built-in predicate `log:includes`[19] which is supported by both reasoners and compares formula expressions. A triple `A log:includes B.` is correct iff the terms `A` and `B` are formula expressions and the triples occuring in `B` also occur in `A`. `{:a :b :c} log:includes {:a :b :c}.` is correct while `{:a :b :c} log:includes {:a :b :o}.` is not. The following rule contains a triple using `log:includes` in its antecedent:

```
{{:a :b :c} log:includes {:a :b ?x}}
                              => {:d :e :f}.
```

The shape of this triple is similar to the examples given before with the difference that instead of `:o` or `:a`, the object of the triple in the right-hand side formula expression is the universal `?x`. Cwm interprets this formula as

$$\texttt{<}\forall\texttt{x. <a b c> includes <a b x>>} \rightarrow \texttt{<d e f>.}$$

Since it is not true that for every `x` the expression `<a b x>` is included in `<a b c>` – think for example in the case above, `x` = `o` – the consequent of the formula is not derived in Cwm. Opposed to that, EYE understands

$$\forall\texttt{x. < <a b c> includes <a b x> >} \rightarrow \texttt{<d e f>.}$$

Here, the antecedent of the implication is fulfilled for `x` = `c`. EYE derives the new triple `d e f.` The different interpretation of universals changes the reasoning result.

5.3.3. Nesting

Our examples contain a third kind of differences: rules which either operate on other rules or on graph structures. Many examples for this can be found in the ORCA dataset where rule-producing rules are applied (see [21]) but also in the Facts4Workers dataset, where formula expressions refer to events. A (shortened) example of such a reference in a rule is shown in Listing 2.

In this example we can perform actions on machines. Such actions can for example be to start or stop the machine, but also to simply do an observation of a problem. Each action performed gets an event id. The rule from the example expresses that, if a problem is observed, the very next action to be performed is to stop the machine

[18]Available at `https://www.w3.org/2000/10/swap/doc/CwmBuiltins` for Cwm, and `http://eulersharp.sourceforge.net/2003/03swap/eye-builtins.html` for EYE.

[19]Prefix `log:<http://www.w3.org/2000/10/swap/log#>`.

```
@prefix http:  <http://www.w3.org/2011/http#>.
@prefix math:  <http://www.w3.org/2000/10/swap/math#>.
@prefix     :  <http://example.org/ex#>.

{
 {?machine :hasProblem ?problem.}
                            :eventId ?eid.

 (?eid 1) math:sum ?nid
}
=>
{
 _:request http:methodName "POST";
        http:requestURI
               "https://f4w/actions";
        http:body ("stop" ?machine).

 {?machine :stoppedBy ?problem.}
                          :eventId ?nid.
}.
```

Listing 2: Example rule using a nested graph (taken from the project Facts4Workers).

```
∀ m. ∀ i. ∀ i2.
<
 ∀ p.
 <m hasProblem p> hasId i.
 (i 1) sum i2
>
→
<
 ∀ p1. ∃ r.
 r methodBame 'POST'.
 r requestURI 'https://f4w/actions'.
 r body ('stop' m).
 <m stoppedBy p1> eventid i2.
>
```

Listing 3: Interpretation of Listing 2 according to Cwm. The occurrences of variable `?problem` are understood as two different variables.

(here via an http-call). Note, that the different actions or events in this example are expressed by formula expressions which contain universal variables. While in EYE the two occurrences of the variable `?problem` corefer – here, the universal quantification of implicit universals is always on top level – this is not the case for Cwm whose interpretation is displayed in Listing 3. We clearly see that the two occurrences of `?problem` are understood as two different variables. The meaning of the rule differs between reasoners and the implementation of this use case does not work with Cwm.

5.3.4. *Distribution of Cases*

Having seen examples for common cases causing differences in the interpretation of N3 formulas, we return to the numbers of Table 2. These numbers depend on the nature of the datasets and the constructs they contain: If a dataset does not contain proofs at all our implementation will also not find any problems related to proof constructs there. The same holds for the two other kinds of potentially problematic constructs we describe above. We therefore show in Figure 12 how many files contain proofs, built-ins and nesting of any kind[20] (dark blue). Next to that information, we also display, how many of these constructs actually lead to conflicting interpretations (light blue). Note that the cases we consider are overlapping: proofs can contain built-ins, files can have deeply nested formula expressions at one place and use built-ins at other places. Due to the nature of proofs, every proof file is also subject to nesting.

We already mentioned earlier that proofs containing universal variables cause conflicting interpretations and that in our examples all proofs contain such variables. As a consequence, the dataset containing most proofs, the DiSSeCt data set, is also the one having the highest share of diverging interpretations. For the other kinds of constructs we discussed, built-in functions and nesting, we take a closer look at the numbers appearing in Figure 12 below where we also explain how the values were calculated

Disagreements per Formula Type

In order to understand how many of the built-ins present in our datasets are causing problems, we extended the attribute grammar by additional attributes. We give the definition of these attributes in Appendix B.1. Here, we only want to briefly discuss the idea: If a triple has a built-in function in predicate position we test whether subject and object of that triple contains universal variables. If this is the case, we next need to know where exactly the interpretation according to Cwm quantifies these variables. If all these universal variables are quantified on a higher level than the parent_c level of the built-in then the built-in construct itself is not causing conflicting interpretations.

In Figure 12 where the numbers calculated by that method are displayed in the middle we already see that in many cases the presence of a built-in construct in a data set does not cause problems. To better understand how likely this presence of a built-in constructs

[20]In that context we define nesting as containing at least one formula expression occurring in another one as in the example `:a :b {:c :d {:e :f :g.}.}.` where `{:e :f :g.}` is nested.

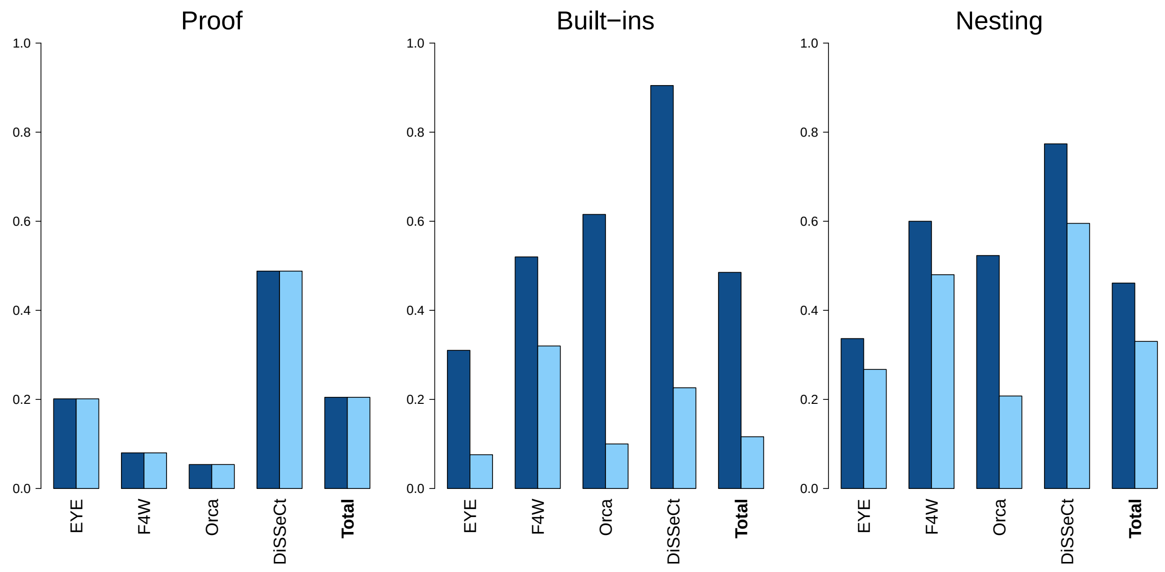

Figure 12: Distributions of proofs, built-ins and nesting in datasets. The share in dark blue always represents the files containing the respective feature while light blue is used to represent the cases where the feature leads to different interpretations.

causes contradicting interpretations we display the numbers from above in another way. Figure 13 shows in how many of the files containing built-in functions at least one of the built-in functions occurs in a construct which causes contradictory interpretations.

We see that for the whole dataset and most subsets approximately a quarter of all files with built-ins contains at least one critical construct. Only in the Facts4Workers dataset this share is higher. This has to do with the fact that in that project one specific built-in is used very often: The built-in `e:whenGround`[21] tests whether its subject contains variables or is ground and calls the object if the latter is true. This built-in is implemented in EYE for a few very specific use cases. We expect that users employing such special predicates are aware of the fact that their rules only work with one specific reasoner. The cases counted here are therefore less critical for the problem that files written for use cases of the Semantic Web which rely on interoperability are interpreted contradictory by different reasoners.

A similar figure as the one above can be produced for nested formulas. We display the share of formulas being subject of a nesting problem in all formulas containing nesting in Figure 14. In Figures 12 and 14 we understand a formula as nested if it contains a graph construct in a graph construct (ie a formula in nested brackets `{...{...}...}`). As a nesting construction leading to the problems described in this section we understand any nested construct containing a universal variable for which the quantifier according to Cwm is on any other level than the top level. This rather broad definition qualifies all differences listed in Table 2 as subject to a nesting problem and these numbers are also used for the two figures. We observe that, when deep nesting is already present in a file, we suffer from conflicting interpretations in 72% of the cases. This is not very surprising since most nested graphs occur in rules which most likely also contain universals, but this figure shows once again, that when using nested graphs, users need to be careful with universal variables.

Categorisation of Errors

As a last point in this subsection we display how the different problem types are distributed over the files counted in Table 2. The reasons for conflicting interpretations of a formula can be overlapping. To be able to show a distribution we separate them in disjoint groups as follows:

Built-in Group Every formula containing a built-in construct which leads to contradictory interpretations is counted as such even if this construct occurs in a proof or additionally contains nested graph constructs without built-ins.

Proof Group Every proof formula for which the reason of the contradictory interpretations is only the proof structure itself. That means that the formula

[21]See `http://eulersharp.sourceforge.net/2003/03swap/log-rules.html#whenGround`.

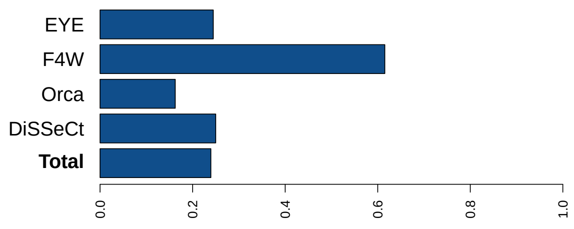

Figure 13: Share of files containing built-ins causing conflicting interpretations in files containing built-ins. Only in a quarter of all files containing built-ins these occur with deeply nested universals.

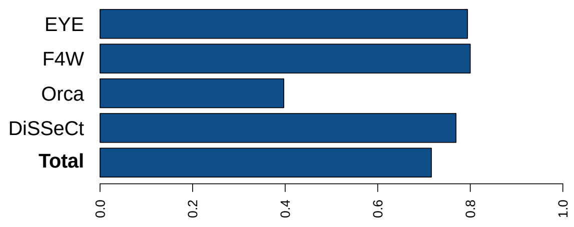

Figure 14: Share of files causing conflicting interpretations in files which have nested expressions. 72% of the files containing nesting are interpreted differently by both reasoners.

does not belong to the *Built-in Group* and that the results of the proof steps – ie the *result* part in the triples `<#lemma> r:gives` { *result* } – do not contain any universal variables which Cwm quantifies on a level which is nested inside these results.

Nesting Group Every formula not belonging to the two groups above is counted as a nested formula.

To test whether conflicting interpretations are caused by built-ins, we used the attributes introduced above. The method to determine whether a formula representing a proof belongs to the *Nesting Class* or the *Proof Class* is discussed in Appendix B.2.

Following this classification, the results of our tests are displayed in Figure 15. For the overall dataset (last line) half (51%) of the conflicts between reasoning results occur only because of the different interpretations of proofs. As discussed these can be considered as rather harmless. The next bigger group of differences occurs in connections with built-in functions (31%). Here, not all, but some of the conflicts are unavoidable, since some built-in functions are not supported by all

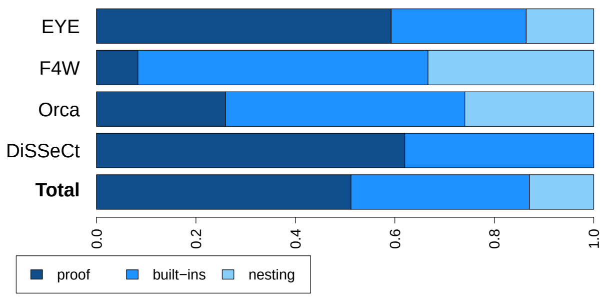

Figure 15: Distribution of different cases causing conflicting interpretations by the reasoners Cwm and EYE.

reasoners. The last group of conflicts (13%), caused by the simple use of nested graphs or rules without direct involvement of built-in functions or proof predicates, is the most dangerous: while users employing built-in functions are often aware that the support of these could be limited to one reasoner and therefore also check carefully when they want to switch to a different one, this is not the case here. If nested rules and graphs are used without any special predicates, it is normally expected that the reasoning results from formulas containing these constructions do not differ between reasoners. The user has no reason to do extra compatibility checks.

Whether these cases occur depends on the use case: In the DiSSeCt dataset, such cases do not occur. This has to do with the fact that the use case here is the test for constraints on RDF data, ie data without formula expressions or rules. The constraints themselves and the results are plain RDF and there is always only one reasoning run applied in order to find constraint violations. In contrast, the Facts4Workers dataset contains many constructs similar to the one displayed in Listing 2 and does therefore have a rather high occurrence of cases belonging to the last category (33%).

6. Possible solutions

In the previous section we examined the impact of having different interpretations for nested implicit universal quantification on practical cases and discovered that 31% of our test files contained at least one critical construct. This means that the problem is not only of theoretical nature and needs to be addressed. In this section, we discuss possible solutions from two perspectives: First, we take the perspective of a practitioner who – given the current situation – needs to make sure that his rules lead to the same result in both reasoners. Secondly, we take a broader perspective and clarify the different positions a standardisation could take.

6.1. Avoiding Conflicts in Practical Cases

Having seen which kinds of constructs lead to conflicts between the different interpretations of N3 we now discuss how to deal with those conflicts. One option is to avoid them from the very beginning by not using nested formula expressions. The drawback of this solution is that this also means to not use the full power of N3 since constructs such as rule-producing rules [21] would not be available any more. If we want to support N3 as it is, we need a way to translate from one reasoner to the other.

6.1.1. EYE formulas interpreted by Cwm

In order to make N3 formulas written for the reasoner EYE be interpreted in the exact same way by Cwm, we can use a simple trick: Since we know that the interpretation of implicitly universally quantified variables also depends on their contexts, ie on their occurrence in the different conjuncts of a formula, we can add a dummy formula which lifts the scope of deeply nested variables to the top level without changing EYE's interpretation of the whole expression. To illustrate that idea we use an example we have seen earlier: remember that in Cwm's interpretation (Interpretation 16') of Formula 16 the quantifier for the universal variable `?y` was nested while for EYE it was on top level, ie in front of the overall formula (Interpretation 16"). If we now add the implication `{?y ?y ?y}=>{?y ?y ?y}.` to the formula this difference disappears. The formula

```
{?y ?y ?y.}=>{?y ?y ?y.}.                    (16a)
{{?x :q ?y.} => {?x :r :c.}.}
                        =>{?x :p :a.}.
```

has in both reasoners the same interpretation, namely:

```
∀x.∀y.                                       (16a')
  <y y y> → <y y y>.
  < < x q y> → < x r c> > → < x p a>.
```

Since the antecedent and the consequent of the added rule are exactly the same, its addition does not change the meaning of the whole formula according to EYE. For Cwm, the meaning does change, the quantifier for `?y` is lifted to the top level and is no longer nested.

While here, the change of meaning has been performed on purpose – we wanted the formula to have the same interpretation by both reasoners – phenomena as the one above can also occur rather randomly: In our datasets there are several rules embedded in a bigger context which make use of nested universals but whose interpretation does not differ between Cwm and EYE.[22] The reason is that they make use of rather arbitrary variable names such as `?x` and `?y` which are also used in other conjuncts of the same very long formulas. Having that in mind, users of Cwm who want to use nested implicit universal quantification need to be careful with the variable names they are using to avoid unwanted changes of scope.

6.1.2. Cwm formulas interpreted by EYE

Performing the other direction – making sure that EYE interprets a formula containing universal variables in nested graphs the same way Cwm does – is more difficult: The only way to do so is to use explicit universal quantification as one can easily see recalling Cwm's interpretation of Formula 16:

```
∀x.<∀y.<x q y> → <x r c>> → <x p a>  (16')
```

Here the universal quantifier for `y` is nested, but EYE interprets all implicitly universally quantified variables as quantified on top level.

Due to the open issues mentioned in Section 2.4, the current version of EYE does not support nested explicit quantification which makes the desired task impossible. This was different in earlier versions.[23] A possible way to make sure that formulas written for Cwm are understood equally by EYE could be to use such an older version of the reasoner. Then our implementation can be used to generate the representation of an N3 formula in core logic according to Cwm's interpretation which we could then translate to explicit quantification in N3. But even when doing that, it cannot be guaranteed that this explicit quantification in N3 works exactly the same way explicit quantification in core logic does since a formal specification of the former is missing. Without a clearly defined explicit quantification in EYE, it is not possible for each nested formula to translate Cwm's interpretation to an N3 formula EYE interprets equally.

6.2. Definition of a Standard

As discussed above, in the current situation the user writing rules needs to either know beforehand with

[22]A concrete example is the file `https://github.com/josd/eye/blob/master/reasoning/n3p/sample.n3` in the eye dataset.

[23]In all versions before EYE-2014-12 nested explicit quantification is allowed.

which reasoning engine he wants to use his rules, or he needs to apply the strategy discussed above of adding dummy rules which lift the quantifier of deeply nested implicitly universally quantified variables to the top level. Given that N3Logic was created for the Semantic Web where interoperability is a very crucial feature, this situation is not acceptable and the community needs to come to an agreement. For such an agreement, we see three options which we discuss below.

6.2.1. Semantics with Nested Universal Quantifiers

One possible solution is to follow Cwm. In that case we could use the specification provided in this paper as the official semantics of N3.

One problem with that solution is that it is rather difficult to formalise. We needed to define an attribute grammar with four attributes to be able to handle Cwm's implicit universal quantification. Following the same approach, scoping on top level only required the definition of one attribute. Being a direct realisation of this grammar our implementation is equally complex and so far we did not encounter an easier way to implement the scoping as intended by Cwm. Even the Cwm reasoner itself has difficulties with implicit universal quantification in some cases (see Appendix A). To the best of our knowledge, there is no other logical framework supporting implicit universal quantification which interprets this feature the way Cwm does (we also discuss other frameworks supporting implicit quantification in Section 7.2). We therefore suspect that the users' intuition about implicit universal quantification could be opposed to Cwm's interpretation. We furthermore see the the difficulties when formalising the semantics and implementing a parser or reasoner following it as an indication that end users writing N3 rules could also have problems to understand and apply the formalisation of implicit universal quantification according to Cwm.

All these reasons make us to rather opt against the possibility to handle implicit universal quantification the way Cwm does.

6.2.2. Semantics with Quantification on Top Level

Another possible remedy for the problem at hand would be to strictly follow the interpretation which understands implicitly universally quantified variables as quantified on top level such as the reasoner EYE does.

One argument to do so is, that this is easier to formalise and also to implement. The attribute we used for the quantification of EYE was rather simple and just passed all implicitly universally quantified variables upwards in the syntax tree. A formalisation could also be made without employing attribute grammars by simply using a universal closure for all universal variables. As assuming the universal closure for variables freely occurring in formulas is common practice and done in many frameworks like for example Prolog [26] we expect that users writing N3 rules can easily understand this behaviour and write their rules accordingly.

We therefore favour the interpretation putting the quantifier of implicitly universally quantified variables on the top level.

6.2.3. Exclude Implicit Universal Quantification

The third solution for the problem explained in this paper would be to either not allow implicit universal quantification at all or to at least only allow it in non-nested structures such that the scoping of every implicitly universally quantified variable is clearly defined in both interpretations. Instead, explicit quantification could be employed. The problem here is that, as explained in Section 2.4, the meaning of explicit quantification is not clearly defined in the W3C team submission either. Problems especially arise if explicit universal quantification is used in combination with explicit existential quantification – to be consistent with the implicit case universal quantification always dominates existential quantification if both occur on the same level – and if constants and quantified variables are not clearly distinguished. In order at least make this last opportunity an applicable option, these uncertainties need to be clarified. One possible way to do that would be to extend the attribute grammar discussed in this paper.

7. Related Work

Our related work section consists of three parts: In the first part we discuss sources that we used to determine the semantics of N3 logic and logical frameworks which are related to N3 in general. Next, we have a broader look into logics that support implicit quantification. The third part discusses our methodology which is very similar to the techniques employed to define the semantics of programming languages.

7.1. N3 Semantics

7.1.1. Determining the meaning of N3 formulas

To determine the intended meaning of N3 and formalise it in this paper we made use of several sources: In 2008 a paper about Notation3 Logic was published [1]. The paper discusses in an informal way the basic concepts of N3 such as citations of formulas, rules, and the use of variables. Further informal specifications for N3 are given by the W3C team submission [2] and a Web page to explain design issues of the Web [3]. Both Web pages provide insight in the semantics of N3 by explaining examples, a formal definition is not given. While the W3C team submission is consistent with the paper, the oldest source, the Web page about design issues, sometimes conflicts with the more recent sources.[24] Therefore it has only been taken into account when further explanation was needed. Despite the differences described above, the implementations of the reasoners Cwm [8] and EYE [9] give an indication how N3 can be understood. Tests as for example described in our previous paper [11] were performed on these reasoners. In case of doubt how an interpretation needs to be or at least is intended to be in a reasoner, questions were sent to the Cwm mailing list[25] or asked in personal communication.[26,27]

7.1.2. RDF

The semantics of the core logic is influenced by the specification of RDF semantics [6]. As Notation3 is meant to be an extension of RDF the direct semantics described in our previous paper [11] as well as the elaboration semantics in the present paper are designed to be compatible with RDF. Similarly to N3, RDF does not have a strict distinction between resources and properties. The same URI can be used in subject and in predicate position. To interpret a constant the interpretation functions (IS and IL) first map the constant to a corresponding element in the domain of discourse, consisting of the non disjoint sets of resources (IR) and properties (IP). A second mapping ($\text{IEXT} : \text{IP} \rightarrow 2^{\text{IR}\times\text{IR}}$) from the set of properties to the power-set of pairs of resources is then used to determine the meaning of the predicates. This approach is similar to our interpretation mechanism for core logic presented in Section 3 where the object mapping $\mathfrak{a}$ maps constants to the domain of discourse. For this domain of discourse we defined a subset of properties to which the predicate function $\mathfrak{p}$ assigns the sets of valid interpretations just as the mapping IEXT does in RDF semantics. A difference between both formalisations can be observed in the handling of implicitly quantified variables: In RDF semantics a mapping (A) from the set of blank nodes to the universe of discourse and not to the set of ground terms is used to model the implicit existential quantification expressed by a blank node. Another difference between the two specifications can be found in the handling of lists: Lists denoted by the `()`-notation are considered to be first-class citizens of N3 [2]. In RDF this notation is a short cut of the first-rest notation using blank nodes. Therefore, our core logic specifically deals with lists while in RDF the meaning of the first-rest predicates is covered instead. To support the definitions of RDF semantics in core logic, rules can be used [1, p.6f]. The predicates `rdf:first` and `rdf:rest` are handled as built-in functions in N3.

7.1.3. Rules for RDF

Next to N3Logic there are other proposals to add rules to RDF. The Rule Interchange Format (RIF) [27] is a W3C standard which was designed to enable the exchange of rules in the Web. Following that purpose, RIF consists of multiple dialects with sometimes conflicting model-theoretic semantics (eg well-founded vs stable semantics) or even with no model-theoretic semantics at all: for the RIF Production Rule Dialect (RIF-PRD) [28] only operational semantics is defined. To put a logic into the context of RIF and enable exchange, RIF provides the RIF Framework for Logic Dialects (RIF-FLD) [29] with some basic semantic definitions which can be extended to fully define the semantics of a rule-based logic. The interaction of RIF with RDF is described in an extra document [30]. RIF rules act on top of RDF and incorporate triples by using so-called frames. These frames only act on the structure of the triples, the semantics of the existing RIF dialects is separated from RDF semantics but can be aligned. This separation between the two semantics is also the reason why we did not choose RIF as a formalism to define the semantics of N3Logic. The idea of N3Logic is to naturally extend RDF by rules and citations. A separation between predicates and constructs of the rule language on the one hand and the well-defined semantics of RDF on the other hand contradicts that goal.

Another framework to express rules over RDF triples is the SPARQL Inference Notation (SPIN) [31]. As the name indicates, SPIN is defined on top of the Semantic Web query language SPARQL and uses the SPARQL query option CONSTRUCT to express rules. The difference between SPARQL and SPIN is that the latter employs RDF to express rules. The semantics of SPIN is inherited from SPARQL and thereby – as in RIF – separated from the semantics of RDF. In the case of SPIN,

[24] As an example take the concept of equality: in the beginning N3 followed the unique name assumption which was later discarded and only remained in N3's the built-in `log:equalTo`. General equality follows the idea of the OWL-concept `owl:sameAs`.

[25] `https://lists.w3.org/Archives/Public/public-cwm-talk/`

[26] The co-author Ruben Verborgh spent the time from July till September 2017 at the Massachusetts Institute of Technology where he worked with Tim Berners-Lee who developed the Cwm reasoner.

[27] Co-author Jos De Roo develops the EYE reasoner.

this separation is rather problematic: As RDF does not support the use of SPARQL's query variables, SPIN employs blank nodes instead. In RDF these blank nodes are understood as implicitly existentially quantified on top level while the query variable of a SPARQL CONSTRUCT query rather works as a universally quantified variable with its quantifier on top level. Therefore the semantics of SPIN and RDF documents is not fully compatible while both follow the same structure.

The Semantic Web Rule Language (SWRL) [32] was developed as an extension for the subset OWL DL of the Web Ontology Language (OWL) [33] and is also compatible with its successor OWL 2 [34] when following its direct semantics [35]. Even though this flavour of OWL can be expressed using an RDF notation it is not fully compatible with RDF. The direct semantics of OWL ignores, for example, RDF's blank nodes and does not understand them as existentially quantified. Furthermore, the expressivity of SWRL is limited to a DL-safe variant of Datalog and does therefore not support nested rules. For these reasons SWRL was not taken into account when the semantics of our core logic was developed.

7.1.4. *Grounding function*

The idea of grounding quantified variables instead of directly mapping them to the domain of discourse using a validation function is inspired by Herbrand semantics [36]. In Herbrand semantics the set of all ground terms forms the domain of discourse. This is different to our approach where we still have a separated domain of discourse. The relation between first order logic with its classical Tarskian semantics and Herbrand semantics is further discussed in a companion paper to the source mentioned above [37]. Every entailment under the classical Tarskian semantics of first order logic is also true under Herbrand semantics. But there are also differences: since the existential quantifier only refers to ground terms of the language and cannot be assigned to a nameless element of the domain of discourse, it is easy to use negation to construct a counter example for the compactness theorem in Herbrand semantics. How the results of Herbrand semantics can be applied to our core logic is subject to further research.

7.1.5. *Cited formulas*

A similar construct as the citation of RDF formulas in an RDF graph as present in N3Logic has recently been proposed in another RDF extension, RDF* [38, 39]. RDF* allows users to annotate triples. The symbols '<<' and '>>' are used to refer to single triples just as N3 uses '{' and '}' to refer to graphs. To search for such patterns the query language SPARQL can be extended to SPARQL*. The semantics of RDF* and SPARQL* is not fixed yet. We could therefore not base or align the semantics of our core logic with RDF* in its current form. Nevertheless, we expect that further research on RDF* will address that problem and enable N3Logic and RDF* to mutually benefit from each other.

The development of N3Logic has furthermore been influenced by the Knowledge Interchange Format (KIF) [40] and the related ISO standard Common Logic (CL) [41] which are both influenced by McCarthy's Logic of context [42]. Both support, such as N3, quantified variables in predicate position and the citation of formulas. Their mechanisms to handle the former is similar to RDF: variables get mapped to resources of the domain of discourse. If resources are used as properties a second mapping interprets these properties. Both formalisms support universal and existential quantification. A difference between them is that in KIF the set of variables is disjoint from the set of constants while CL, similar to N3 for explicit quantification (see Section 2.4), does not distinguish between variable names and constants. As a consequence, variables cannot occur freely in CL formulas. Free variables in KIF are universally bound on top level. Similar as it is done in core logic, cited formulas in KIF and in CL are interpreted as single elements of the universe of discourse. For KIF, this is done by mapping them directly to their string representation which in KIF needs to be included in the universe of discourse. In CL, a citation is written as a pair of a name and a text (the cited logical formula(s)). This name can then be used in different contexts. Both logics also provide a mechanism to relate the quotation to the actual formula they quote. Being out of scope for this paper which focusses on implicit quantification, it is also planned to add such a mechanism to our core logic. KIF, CL, but also their predeceasing logic of contexts are the most promising sources for this extension of our formalisation.

7.2. *Implicit Quantification*

Implicit quantification is widely used in different contexts. Most frameworks around RDF support the use of *blank nodes*. Hogan et al [43] provide an overview of their meaning and use in these different contexts. In RDF and RDFS, blank nodes stand, just as in N3, for implicitly existentially quantified variables. The quantification is local, ie the same blank nodes cannot be shared between different documents and the existential quantification always counts for a graph. Traditionally, RDF does not support nested graph constructions. This means that all blank nodes occurring in N3 triples which are also valid in RDF have the same meaning in both frameworks. In the newer addition TriG [44] different graph constructs indicated by the `{}`-construct can have common blank nodes.[28] Where these blank nodes are quantified depends on the semantics chosen. Hogan et al furthermore point out that, despite the clear definition, users do not always understand and use blank nodes as existentially quantified variables in RDF: often blank nodes are rather used to refer a concrete object whose IRI is unknown. A possible reason for that is that SPARQL [45] interprets blank nodes occurring in the queried RDF graph exactly in that manner. Also in N3 or at least in the N3 implementation provided by the EYE reasoner we can find some built-in functions which treat blank nodes that way, an example is the predicate `e:findall`[29] from EYE. Built-in functions are excluded from this paper but certainly form a major challenge in our future work.

Implicit universal quantification can be found in different programming languages and RDF related frameworks: in Prolog [26] variables are understood to be universally quantified. The scope of this quantification is the clause in which the variable occurs. This is similar to the universal quantification on top level as implemented in EYE. But as Prolog does not allow the construction of nested rules, which form the biggest challenge for the determination of scoping in N3, Prolog's quantification is only partly comparable to N3's. The RDF-query language SPARQL [45] supports *query variables* which, if a query is understood as some kind of filter rule, can be seen as universally quantified. SPARQL allows nesting of graphs and queries. A SPARQL query consists of two parts, an outer part starting with one of the keywords SELECT, DESCRIBE, ASK, or CONSTRUCT which can contain search variables and a WHERE-part which specifies the search pattern. If a query is nested in another one, ie a new query occurs in the WHERE-part of another query, only the universal variables which occur in the SELECT-part of the sub-query share their variables with the WHERE-part of the top query.[30] The aspect that variables in nested queries are clearly separated in these cases is slightly similar to the separation of different deeply nested graphs in N3. However, when we translate such nested queries to core logic rules, we only need to rename identical variables occurring separately on different levels, the universal quantifier for all variables is still on top level.

[28]`https://www.w3.org/TR/trig/#terms-blanks-nodes`

[29]`http://eulersharp.sourceforge.net/2003/03swap/eye-builtins.html`

[30]`https://www.w3.org/TR/2013/REC-sparql11-query-20130321/#subqueries`

7.3. *Mappings from Representations to Core Logics*

The approach of translating a logical representation into a well defined core logic to explain its semantics has been inspired by the formal description of programming languages where this practice is quite common. In programming languages, normally the lambda calculus and its extensions are used as a core logic. The general idea is, for example, explained by Pierce [46] and has been implemented for several programming languages: Sulzmann et al [47] define System F_C, an extension of the polymorphic lambda calculus System F, to provide a way to express even rather complicated constructs contained in Haskell and other functional languages, as for example generalised algebraic data types (GADTs) and associated types, in terms of a well defined logic. Next to the definition and discussion of the basic properties of that logic, the authors also show how to translate the above mentioned examples from a source language into System F_C. Igarashi et al [48] follow a similar approach for the programming language Java. To have a logical representation of the core features of Java, they define Featherweight Java. This logic can be used to describe and prove essential properties of the programming language. They furthermore discuss how the formalism can be extended by adding generic types and methods.

In the Semantic Web context a similar approach to the one represented in this paper can be found for the definition of SPARQL's semantics: instead of defining the semantics directly on the language itself, expressions of SPARQL are first mapped to the SPARQL algebra for which an evaluation semantics is defined.[31] Another similar, although slightly different, approach can be found for RDF. De Bruijn et al [49] embed RDF into F-Logic and first order logic. The difference to our approach is that this embedding is not done to define the semantics of RDF – this is defined directly – but to research its relation to other logics. The authors show that RDF can be represented in the two frameworks.

8. Conclusion

We conclude this paper in two parts: First, we get back to the research questions which we raised in the

[31]`https://www.w3.org/TR/2013/REC-sparql11-query-20130321/#sparqlDefinition`

introduction and summarize how they have been addressed in this paper. In a second part we give an outlook to future work and discuss open challenges for the formalisation of N3Logic.

8.1. Review of the Research Questions

We discuss each research question separately:

(i) *How can we formally express the difference between two interpretations of the same N3 formula?*

To express the interpretations of implicitly quantified variables in N3Logic, we defined a core logic (Section 3). The syntax of this core logic is very close to the syntax of N3 and – apart from some differences in the symbols – only differs from it in the fact that it only supports explicit quantification. That way, it is easy to express at which position an interpretation sets the quantifier when being faced with implicit quantification and to compare different interpretations. For the syntax we also provided a semantics which respects the characteristics of N3 – like the fact that lists need to be treated as *first-class citizens* – and allows for further refinement of concepts which have not been treated in detail in this paper like the interpretation of cited formulas and the definition of N3's built-in functions.

To connect an N3 formula to its core logic counterpart, we chose the concept of attribute grammars (Section 4.2) and can – using this concept – formally define how an interpretation maps N3 to core logic.

(ii) *How do existing interpretations of N3Logic conceptually differ in their way of handling implicit quantification?*

To answer this research question we first explained that universal quantification is closely related to the concept of a *parent formula* (Section 2.3.2) which is underspecified in the W3C team submission. We specified attribute grammars for two different interpretations (Section 4): one seeing the top formula as the *parent formula* of all implicitly universally quantified variables occurring in it regardless of their level of nesting – this is the interpretation the reasoner EYE applies – one for which this *parent formula* of an implicitly universally quantified variable is not the direct formula either written in curly brackets {} or being the top formula containing the variable but the next higher formula fulfilling these conditions – this approach is implemented in Cwm.

(iii) *How often does this conceptual difference lead to conflicting interpretations of formulas used in practical applications?*

To know whether the conceptual difference in the treatment of implicit universal quantification is not only of theoretical nature but does have impact on practical cases we implemented the previously defined attribute grammars in Haskell (Section 5.1). We applied the implementation to different datasets which were written for practical applications and discovered that in 31% of all our investigated files at least one construct which is understood differently by the different interpretations can be found (Sections 5.2 and 5.3).

(iv) *Which kinds of constructs cause these conflicting interpretations in practice and how likely is it that a file containing these constructs is actually subject to the problem?*

For the answer of this last question, we looked deeper into the datasets and identified three different categories of constructs which caused conflicting interpretations, namely proofs, built-ins, and deeply nested formulas without the occurrence of proof-constructs or built-ins (Section 5.3). For all these constructs we provided examples and discussed whether the differences in the interpretations are problematic which was especially the case for nested constructs without built-ins which are not part of a proof. We then tested, given one of these constructs is present in a file, how likely it is that this file is treated differently by the two different interpretations. While this was always the case if proofs are present, disagreements could also be found for a quarter of the cases containing built-ins and for 72% of the files containing nested constructs of any kind. As a last step, we divided the files showing differences in the interpretations into disjoint groups depending on the main reason causing that difference. We identified that 13% of our disagreements were caused by simple nesting without built-ins or proof-constructs. These are the cases we consider as most dangerous since files are mostly manually written – opposed to computer generated proofs – by users which do not have a specific reasoner in mind.

Especially these cases lead us to the conclusion that the problem of having different interpretations for deeply nested implicit universal quantification needs to be addressed by a formalised standard. In our opinion, this standard should be easy to understand for users and easy to implement which is why we favour to the interpretation which understands the top formula as the parent for all universals. By providing a way to explicitly express the differences between existing interpretations as we did by defining the core logic and showing how attribute grammars can be used to map from N3 formulas to that logic, we provided tools which ease the discussion. Having these tools at hand enables anyone involved to clearly define how he understands implicit universal quantification and to test where this interpretation differs from others. We thereby set a step forward towards the standardisation of N3.

8.2. *Open Challenges and Future Directions*

While this paper shows how the uncertainties about implicit quantification in N3 can be clarified there are further challenges lying ahead which need to be solved in order to standardise N3Logic.

Next to the clarification of the different ways to interpret implicit quantification in N3 and a study whether the concrete meaning of existing N3 files changes when applying a different interpretation, the expressivity of formulas using implicit quantification is an important factor when making a choice how this should be handled. Understanding the *parent formula* as the top formula and not as a nested formula makes N3 with only implicit quantification less expressive. Whether this limited expressivity is strong enough to support all the tasks N3 is meant for and how it compares to other standards needs to be carefully studied in order to come to an agreement.

The expressivity of implicit quantification becomes less crucial if N3, as intended by the W3C team submission, also supports explicit quantification. Section 2.4 briefly discusses the problems and uncertainties in the current informal specification. The most important factors are how the exact position of a quantifier should be taken into account (ie does it make sense that universal quantification always dominates existential quantification) and whether or not variables should be separated from constants and have for example a designated name space. These topics need to be discussed in the community. We plan to use our core logic to formalise the different options which then, hopefully, leads to an agreement.

Another important topic we excluded from this paper is the formalisation of built-in functions. The W3C team submission discusses different predicates which are considered as part of N3 like the list functions `rdf:first` and `rdf:last` which N3 inherits from RDF but also, for example, the predicate `log:equalTo` to state ground equality. We designed the core logic in such way that it can be easily extended to support these and other predicates and plan to provide this formalisation as future work.

The last, but probably most critical point, which needs to be tackled in order to standardise N3 is a clear definition how cited formulas need to be handled. The need for such constructs but also the difficulties which come with this kind of standardisation can be observed in RDF: RDF reification is excluded from the definition of RDF semantics [6] and for the TriG syntax to express named graphs, there is a disagreement about its meaning [13]. We plan to base our own proposal for this part of N3Logic on KIF [40] and ISO Common Logic [41].

With the definition of an extendible core logic we provided a framework which can and will help us to clarify all properties of N3Logic and thereby leverage this logic to become a standard for the Semantic Web.

Acknowledgements

The described research activities were funded by Ghent University, imec, Flanders Innovation & Entrepreneurship (VLAIO), the Research Foundation – Flanders (FWO) and the European Union. Ruben Verborgh is a postdoctoral fellow of the Research Foundation – Flanders (FWO).

Appendix A. Problems in Cwm

In this section we explain where the Cwm's interpretations of N3 formulas as specified in this paper differ from the actual implementation. We start with a simple example. Consider the formula:

```
{?x :p :o}=>{?x :q :o}.
  :a :b {:c :d {:e :f {?x :g :h}}}.   (A.1)
```

According to the formalisation provided in this paper, the occurrences of the variable `?x` in the implication are universally quantified on its parent$_c$ formula which is the overall formula. This quantification then also counts for the third occurrence of `?x` which is more deeply nested. In core logic this formula can be represented as:

```
∀x.  <x p o> → <x q o>.
           <a b <c d <e f <x g h>>>.   (A.1')
```

In this case this is also the result Cwm gives. In Listing 4 we display the reasoning result of Cwm[32] when provided with Formula A.1. The output is produced by using the option `--think` which makes the reasoner derive the deductive closure of its input dataset. During this process Cwm also translates all implicit universal

[32]Version: v 1.197 2007/12/13 15:38:39 syosi

```
@prefix :  <http://example.org/ex#>
@prefix c:  <#>

@forAll c:x .
:a :b {
  :c :d {:e :f {c:x :g :h.}.}.
}.
{c:x :p :o.} => {c:x :q :o.}.
```

Listing 4: Output of Cwm when provided with Formula A.1.

```
 1 @prefix :  <http://example.org/ex#>
 2 @prefix c:  <#>
 3
 4 @forAll c:x .
 5 :a :b {
 6   :c :d {
 7     @forAll c:x.
 8     :e :f {c:x :g :h.}.
 9   }.
10 }.
11 {c:x :p :o.} => {c:x :q :o.}.
```

Listing 5: Output of Cwm when provided with Formula A.2.

quantification into its explicit counterpart. But if we now change the order of the conjunction to

```
:a :b {:c :d {:e :f {?x :g :h}}}.
              {?x :p :o}=>{?x :q :o}.   (A.2)
```

we get a different result. According to the formalisation this formula has the same meaning as the previous one namely Interpretation A.1'. In Listing 5 we display the reasoning output of Cwm for this formula. We clearly see that in this interpretation an extra universal quantifier for the deeply nested occurrence of ?x is added (line 7). For Cwm the interpretation of the conjunction depends on the order of its conjuncts. Cwm does its translation from implicit to explicit quantification while parsing. Whenever a new universal variable is encountered which is not quantified yet a universal quantifier is added on its parent$_c$ level. This mechanism does not take the later encounter of universal variables on higher levels into account. Such a mechanism would have a negative impact on the performance of Cwm. Similar examples can be easily constructed.

Appendix B. Attributes and Methods for Evaluation

For the evaluation in Section 5.3.4 we defined several attributes. In this section, we display the definition of these attributes and explain how they have been used for the categorisation of different cases.

Appendix B.1. Critical Built-in Constructs

To identify the built-in constructs in our datasets which are causing conflicting interpretations we extend the attribute grammar by the two synthesized attributes b and bi. We display the rules for these attributes in Figure B.16.

Attribute b is used to pass the set of universal variables occurring in the subject or object position of a built-in predicate upwards in the syntax tree to the parent$_c$ formula of that built-in. Here, it is most interesting how the attribute behaves for the production rule of a simple triple, ie the rule `f := t`$_1$`t`$_2$`t`$_3$. In that case we make use of the attributes m_c and u which have been defined in Section 4.3.2 and apply function $f_1 : \mathcal{T} \times 2^U \times 2^U \rightarrow 2^U$ which is defined as follows:

$$f_1(t, s_1, s_2) := \begin{cases} s_2 & \texttt{if } \; t \text{ is no built-in symbol} \\ s_1 & \text{else} \end{cases}$$

The first argument of the function is the actual value of the predicate which can either be a built-in or not. In the case it is not a built-in, the function simply passes the information collected by the attribute so far upwards. If the predicate is a built-in, the values of the attribute u for subject and object get collected, ie all universal variables which occur in these two positions. For the other production rules the attribute value is either the empty set – in case we have a rule resulting in a terminal node of the tree – or it gets passed upwards through the tree. The only exception is the production rule `e ::= {f}` for which the attribute value of `e` is the empty set since `f` is a parent$_c$ formula. For this formula the attribute value `e`.s with s defined as in Section 4.3.2 gives information which variables are already quantified on a higher level. This is used in the definition of attribute bi.

Attribute bi carries the information whether or not a built-in predicate has been found whose subject or object contains one or more universal variable which Cwm's interpretation quantifies either on the same level the built-in function occurs or any lower level. If this is the case, the value of the attribute is 1, else it is 0. If we look at Figure B.16 we see that the rules for this attribute mostly simply pass information upwards with one exception: For the production rule `e ::= {f}` we apply the function $f_2 : \mathbb{N} \times 2^{U \times U} \rightarrow \mathbb{N}$ which is defined as follows:

$$f_2(n, s) := \begin{cases} 0 & \text{if } n = 0 \text{ and } s = \emptyset \\ 1 & \text{else} \end{cases}$$

To better understand this definition, we take a closer look to the arguments the function is used with in the

production rule		rules for b	rules for bi
`s ::=`	`f`	$\texttt{s}.b \leftarrow \emptyset$	$\texttt{s}.\mathrm{bi} \leftarrow \texttt{f}.\mathrm{bi}$
`f ::=`	$\texttt{t}_1\texttt{t}_2\texttt{t}_3.$	$\texttt{f}.\mathrm{b} \leftarrow f_1(\texttt{t}_2.m_c, (\texttt{t}_1.u \cup \texttt{t}_3.u), (\texttt{t}_1.\mathrm{b} \cup \texttt{t}_2.\mathrm{b} \cup \texttt{t}_3.\mathrm{b}))$	$\texttt{f}.\mathrm{bi} \leftarrow \max(\texttt{t}_1.\mathrm{bi}, \texttt{t}_2.\mathrm{bi}, \texttt{t}_3.\mathrm{bi})$
	$\texttt{e}_1\texttt{=>}\texttt{e}_2.$	$\texttt{f}.\mathrm{b} \leftarrow \texttt{e}_1.\mathrm{b} \cup \texttt{e}_2.\mathrm{b}$	$\texttt{f}.\mathrm{bi} \leftarrow \max(\texttt{e}_1.\mathrm{bi}, \texttt{e}_2.\mathrm{bi})$
	$\texttt{f}_1\texttt{f}_2$	$\texttt{f}.\mathrm{b} \leftarrow \texttt{f}_1.\mathrm{b} \cup \texttt{f}_2.\mathrm{b}$	$\texttt{f}.\mathrm{bi} \leftarrow \max(\texttt{f}_1.\mathrm{bi}, \texttt{f}_2.\mathrm{bi})$
`t ::=`	`uv`	$\texttt{t}.\mathrm{b} \leftarrow \emptyset$	$\texttt{t}.\mathrm{bi} \leftarrow 0$
	`ex`	$\texttt{t}.\mathrm{b} \leftarrow \emptyset$	$\texttt{t}.\mathrm{bi} \leftarrow 0$
	`c`	$\texttt{t}.\mathrm{b} \leftarrow \emptyset$	$\texttt{t}.\mathrm{bi} \leftarrow 0$
	`e`	$\texttt{t}.\mathrm{b} \leftarrow \texttt{e}.\mathrm{b}$	$\texttt{t}.\mathrm{bi} \leftarrow \texttt{e}.\mathrm{bi}$
	`(k)`	$\texttt{t}.\mathrm{b} \leftarrow \texttt{k}.\mathrm{b}$	$\texttt{t}.\mathrm{bi} \leftarrow \texttt{k}.\mathrm{bi}$
	`()`	$\texttt{t}.\mathrm{b} \leftarrow \emptyset$	$\texttt{t}.\mathrm{bi} \leftarrow 0$
`k ::=`	`t`	$\texttt{k}.\mathrm{b} \leftarrow \texttt{t}.\mathrm{b}$	$\texttt{k}.\mathrm{bi} \leftarrow \texttt{t}.\mathrm{bi}$
	$\texttt{t k}_1$	$\texttt{k}.\mathrm{b} \leftarrow \texttt{t}.\mathrm{b} \cup \texttt{k}_1.\mathrm{b}$	$\texttt{k}.\mathrm{bi} \leftarrow \max(\texttt{t}.\mathrm{bi}, \texttt{k}_1.\mathrm{bi})$
`e ::=`	`{f}`	$\texttt{e}.\mathrm{b} \leftarrow \emptyset$	$\texttt{e}.\mathrm{bi} \leftarrow f_2(\texttt{f}.\mathrm{bi}, (\texttt{f}.\mathrm{b} \setminus \texttt{e}.s))$
	`{}`	$\texttt{e}.\mathrm{b} \leftarrow \emptyset$	$\texttt{e}.\mathrm{bi} \leftarrow 0$
	`false`	$\texttt{e}.\mathrm{b} \leftarrow \emptyset$	$\texttt{e}.\mathrm{bi} \leftarrow 0$

Figure B.16: Attribute rules for the synthesized attributes b (middle) and bi (right), and their corresponding production rules (left) from the N3 grammar (Figure 2). The attributes test whether a formula contains built-ins whose subject or object has universals which are not quantified on the parent or any higher level of the built-in.

attribute definition: The first argument is simply used to pass the information that a problematic built-in construct has been found upwards in the syntax tree. So, if its value is 1, the application of function f_2 should also result in 1 (second case). As the second argument, we have the difference of $\texttt{f}.b$ – the variables we collected using attribute b – and $\texttt{e}.s$ – the variables which are quantified on the parent$_c$ level of the formula. If this difference is not empty, this means that we found a variable which occurs in the subject or object position of a built-in (collected by b) which Cwm quantifies either on the same level as the built-in or on a lower level. These are the constructs which are problematic in our consideration. In these cases function f_2 will result in 1, else its value is 0.

Appendix B.2. Nested Universals

The reasons for conflicting interpretations of a formula can be overlapping. Especially for proofs we want to know whether conflicts are only caused by the proof vocabulary which normally uses graphs – these cases are rather harmless – or if either a built-in construct in combination with a formula expression quantified within this expression or another occurrence of such a formula expressions is involved.

For built-ins we use the attributes defined in Appendix B.1. To find deeply nested universals we introduce the concept of depth: we measure the depth of a universal variable as one plus the number of formula expressions its quantifier is enclosed in when translating the N3 formula to its core logic translation according to Cwm. If a variable does not occur in a file at all, it has depth 0. For example, Formula 16 from Section 4.3.2 the variable `?y` has depth 2 while `?x` has depth 1.

In order to determine the depth of the deepest nested variable in a formula we define two attributes: the inherited attribute c and the synthesized attribute d. The first attribute c simply goes downwards in the syntax tree and counts every formula expression it encounters on the way down. To understand attribute d we first take a closer look to the formula involved, $f_3 : 2^{U\times U} \times \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is defined as follows:

$$f_3(s, n, m) := \begin{cases} m & \text{if } s = \emptyset \\ \max(n, m) & \text{else} \end{cases}$$

Attribute d now takes this formula to check for every formula expression and for the top formula whether according to Cwm's translation there is at least one quantifier for universal variables at that level (remember that the set of universals quantified at each level is captured

production rule		rules for d	rules for c
`s::=`	`f`	$\mathtt{s}.d \leftarrow f_3(\mathtt{f}.q, 1, \mathtt{f}.d)$	$\mathtt{f}.c \leftarrow 1$
`f::=`	$\mathtt{t}_1\mathtt{t}_2\mathtt{t}_3.$	$\mathtt{f}.d \leftarrow \max_i \mathtt{t}_i.d$	$\mathtt{t}_i.c \leftarrow \mathtt{f}.c$
	$\mathtt{e}_1\mathtt{=>}\mathtt{e}_2.$	$\mathtt{f}.d \leftarrow \max_i \mathtt{e}_i.d$	$\mathtt{e}_i.c \leftarrow \mathtt{f}.c$
	$\mathtt{f}_1\mathtt{f}_2$	$\mathtt{f}.d \leftarrow \max_i \mathtt{f}_i.d$	$\mathtt{f}_i.c \leftarrow \mathtt{f}.c$
`t::=`	`uv`	$\mathtt{t}.d \leftarrow 0$	
	`ex`	$\mathtt{t}.d \leftarrow 0$	
	`c`	$\mathtt{t}.d \leftarrow 0$	
	`e`	$\mathtt{t}.d \leftarrow \mathtt{e}.d$	$\mathtt{e}.c \leftarrow \mathtt{t}.c$
	`(k)`	$\mathtt{t}.d \leftarrow \mathtt{k}.d$	$\mathtt{k}.c \leftarrow \mathtt{t}.c$
	`()`	$\mathtt{t}.d \leftarrow 0$	
`k::=`	`t`	$\mathtt{k}.d \leftarrow \mathtt{t}.d$	$\mathtt{t}.c \leftarrow \mathtt{k}.c$
	$\mathtt{t}\ \mathtt{k}_1$	$\mathtt{k}.d \leftarrow \max(\mathtt{t}.d, \mathtt{k}_1.d)$	$\mathtt{t}.c, \mathtt{k}.c \leftarrow \mathtt{k}.c$
`e::=`	`{f}`	$\mathtt{e}.d \leftarrow f_3(\mathtt{f}.q, \mathtt{f}.c, \mathtt{f}.d)$	$\mathtt{f}.c \leftarrow \mathtt{e}.c + 1$
	`{}`	$\mathtt{e}.d \leftarrow 0$	
	`false`	$\mathtt{e}.d \leftarrow 0$	

Figure B.17: Attribute rules for the synthesized attribute d (middle) and the inherited c (right), and their corresponding production rules (left) from the N3 grammar (Figure 2). The attributes keep track of the deepest nested universal in a formula.

by attribute q). If so, it takes the depth of the current formula which is captured by attribute c and compares it with the values for d already found in the depending nodes. From these two values it takes the maximum. If there are no universal quantifiers at a node, the attribute only passes the maximum value for d of the depending nodes upwards. The value $\mathtt{s}.d$ is now the depth of the deepest nested universal in the formula.

Because of the nature of the proof we know that if the maximum depth of the universal quantifiers is two or lower, then the only reason for conflicting interpretations is the use of the predicate `r:gives` which has a graph as object. If the maximum depth of universal variables is bigger than two, we know that the conflict found in the proof is more serious and caused by a deeper nesting in one of the formulas occurring in the object of `r:gives`.

References

[1] T. Berners-Lee, D. Connolly, L. Kagal, Y. Scharf, J. Hendler, N3Logic: A logical framework for the World Wide Web, Theory and Practice of Logic Programming 8 (3) (2008) 249–269. doi:http://dx.doi.org/10.1017/S1471068407003213.

[2] T. Berners-Lee, D. Connolly, Notation3 (N3): A readable RDF syntax, W3C Team Submission, `http://www.w3.org/TeamSubmission/n3/` (Mar. 2011).

[3] T. Berners-Lee, Notation 3 logic, `http://www.w3.org/DesignIssues/N3Logic` (2005).

[4] F. Gandon, G. Schreiber, RDF 1.1 XML Syntax, W3C Recommendation, `https://www.w3.org/TR/rdf-syntax-grammar/` (Feb. 2014).

[5] R. Cyganiak, D. Wood, M. Lanthaler, RDF 1.1: Concepts and Abstract Syntax, W3C Recommendation, `http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/` (Feb. 2014).

[6] P. J. Hayes, P. F. Patel-Schneider, RDF 1.1 Semantics, W3C Recommendation, `http://www.w3.org/TR/2014/REC-rdf11-mt-20140225/` (Feb. 2014).

[7] D. Beckett, T. Berners-Lee, E. Prud'hommeaux, G. Carothers, Turtle - Terse RDF Triple Language, W3C Recommendation, `http://www.w3.org/TR/turtle/` (Feb. 2014).

[8] T. Berners-Lee, *cwm*, `http://www.w3.org/2000/10/swap/doc/cwm.html` (2000–2009).

[9] J. De Roo, Euler yet another proof engine, `http://eulersharp.sourceforge.net/` (1999–2019).

[10] FuXi 1.4: A Python-based, bi-directional logical reasoning system for the semantic web, `http://code.google.com/p/fuxi/`.

[11] D. Arndt, R. Verborgh, J. De Roo, H. Sun, E. Mannens, R. Van de Walle, Semantics of Notation3 logic: A solution for implicit quantification, in: N. Bassiliades, G. Gottlob, F. Sadri, A. Paschke, D. Roman (Eds.), Rule Technologies: Foundations, Tools, and Applications, Vol. 9202 of Lecture Notes in Computer Science, Springer, 2015, pp. 127–143.
URL `http://link.springer.com/chapter/10.1007/978-3-319-21542-6_9`

[12] M. Duerst, M. Suignard, Internationalized Resource Identifiers (IRIs), `http://www.ietf.org/rfc/rfc3987.txt` (Jan. 2005).

[13] A. Zimmermann, RDF 1.1: On Semantics of RDF Datasets, W3C Working Group Note, `http://www.w3.org/TR/2014/NOTE-rdf11-datasets-20140225/` (Feb. 2014).

[14] R. Verborgh, J. De Roo, Drawing conclusions from Linked Data on the Web, IEEE Software 32 (5) (2015) 23–27.
URL `http://online.qmags.com/ISW0515?cid=3244717&eid=19361&pg=25`

[15] L. Henkin, Completeness in the theory of types, Journal of Symbolic Logic 15 (2) (1950) 81–91. doi:10.2307/2266967.

[16] D. E. Knuth, Semantics of context-free languages, Mathematical systems theory 2 (2) (1968) 127–145. doi:10.1007/BF01692511.
URL `http://dx.doi.org/10.1007/BF01692511`

[17] D. E. Knuth, Semantics of context-free languages: Correction, Mathematical systems theory 5 (2) (1971) 95–96. doi:10.1007/BF01702865.
URL `http://dx.doi.org/10.1007/BF01702865`

[18] J. Paakki, Attribute grammar paradigms—a high-level methodology in language implementation, ACM Comput. Surv. 27 (2) (1995) 196–255. doi:10.1145/210376.197409.
URL `http://doi.acm.org/10.1145/210376.197409`

[19] S. D. Swierstra, P. R. A. Alcocer, J. Saraiva, Designing and implementing combinator languages, in: International School on Advanced Functional Programming, Springer, 1998, pp. 150–206.

[20] D. Arndt, B. De Meester, P. Bonte, J. Schaballie, J. Bhatti, W. Dereuddre, R. Verborgh, F. Ongenae, F. De Turck, R. Van de Walle, E. Mannens, Ontology reasoning using rules in an eHealth context, in: N. Bassiliades, G. Gottlob, F. Sadri, A. Paschke, D. Roman (Eds.), Rule Technologies: Foundations, Tools, and Applications, Vol. 9202 of Lecture Notes in Computer Science, Springer, 2015, pp. 465–472.
URL `http://link.springer.com/chapter/10.1007/978-3-319-21542-6_31`

[21] D. Arndt, B. De Meester, P. Bonte, J. Schaballie, J. Bhatti,

W. Dereuddre, R. Verborgh, F. Ongenae, F. De Turck, R. Van de Walle, E. Mannens, Improving OWL RL reasoning in N3 by using specialized rules, in: V. Tamma, M. Dragoni, R. Gonçalves, A. Ławrynowicz (Eds.), Ontology Engineering: 12th International Experiences and Directions Workshop on OWL, Vol. 9557 of Lecture Notes in Computer Science, Springer, 2016, pp. 93–104. doi:10.1007/978-3-319-33245-1_10.
URL http://dx.doi.org/10.1007/978-3-319-33245-1_10

[22] D. Arndt, J. Van Herwegen, R. Verborgh, E. Mannens, R. Van de Walle, Using rules to generate and execute workflows in smart factories, in: Proceedings of the RuleML 2016 Challenge, Doctoral Consortium and Industry Track hosted by the 10th International Web Rule Symposium, Vol. 1620 of CEUR Workshop Proceedings, 2016.
URL http://ceur-ws.org/Vol-1620/paper12.pdf

[23] D. Arndt, B. De Meester, A. Dimou, R. Verborgh, E. Mannens, Using rule-based reasoning for RDF validation, in: S. Costantini, E. Franconi, W. Van Woensel, R. Kontchakov, F. Sadri, D. Roman (Eds.), Proceedings of the International Joint Conference on Rules and Reasoning, Vol. 10364 of Lecture Notes in Computer Science, Springer, 2017, pp. 22–36. doi:10.1007/978-3-319-61252-2_3.

[24] R. Verborgh, D. Arndt, S. Van Hoecke, J. De Roo, G. Mels, T. Steiner, J. Gabarró Vallés, The pragmatic proof: Hypermedia API composition and execution, Theory and Practice of Logic Programming 17 (1) (2017) 1–48. doi:10.1017/S1471068416000016.
URL http://arxiv.org/pdf/1512.07780v1.pdf

[25] T. Berners-Lee, Semantic Web Application Platform, http://www.w3.org/2000/10/swap/ (2000).

[26] W. F. Clocksin, C. S. Mellish, Programming in PROLOG, Springer, 1994.

[27] A. Kifer, H. Boley, RIF Overview (Second Edition), w3c Working Group Note, https://www.w3.org/TR/rif-overview/ (Feb. 2013).

[28] C. de Sainte Marie, G. Hallmark, A. Paschke, RIF Production Rule Dialect (Second Edition), w3c Recommendation, https://www.w3.org/TR/rif-prd/ (Feb. 2013).

[29] H. Boley, M. Kifer, RIF Framework for Logic Dialects (Second Edition), w3c Recommendation, https://www.w3.org/TR/rif-fld/ (Feb. 2013).

[30] J. de Bruijn, C. Welty, RIF RDF and OWL Compatibility (Second Edition), w3c Recommendation, https://www.w3.org/TR/rif-rdf-owl/ (Feb. 2013).

[31] H. Knublauch, J. Hendler, K. Idehen, SPIN - Overview and Motivation, w3c Member Submission, https://www.w3.org/Submission/spin-overview/ (Feb. 2011).

[32] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean, SWRL: A Semantic Web Rule Language Combining OWL and RuleML, w3c Member Submission, https://www.w3.org/Submission/SWRL/ (May 2004).

[33] M. D. , G. Schreiber, S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, L. A. Stein, owl Web Ontology Language, w3c Recommendation, https://www.w3.org/TR/owl-ref/ (Feb. 2004).

[34] W3C OWL Working Group, owl 2 Web Ontology Language, w3c Recommendation, https://www.w3.org/TR/owl2-overview/ (Dec. 2012).

[35] B. Motik, P. F. Patel-Schneider, B. C. Grau, I. Horrocks, B. Parsia, U. Sattler, owl 2 Web Ontology Language Direct Semantics, w3c Recommendation, https://www.w3.org/TR/owl2-direct-semantics/ (Dec. 2012).

[36] M. Genesereth, E. Kao, The herbrand manifesto, in: N. Bassiliades, G. Gottlob, F. Sadri, A. Paschke, D. Roman (Eds.), Rule Technologies: Foundations, Tools, and Applications, Springer International Publishing, Cham, 2015, pp. 3–12.

[37] M. Genesereth, E. Kao, Herbrand Semantics, http://logic.stanford.edu/herbrand/herbrand.html (2015).

[38] O. Hartig, RDF* and SPARQL*: An alternative approach to annotate statements in RDF, in: Proceedings of the ISWC 2017 Posters & Demonstrations and Industry Tracks co-located with 16th International Semantic Web Conference (ISWC 2017), Vienna, Austria, October 23rd - to - 25th, 2017., 2017.
URL http://ceur-ws.org/Vol-1963/paper593.pdf

[39] O. Hartig, B. Thompson, Foundations of an alternative approach to reification in RDF, CoRR abs/1406.3399. arXiv:1406.3399.
URL http://arxiv.org/abs/1406.3399

[40] M. R. Genesereth, R. E. Fikes, et al., Knowledge interchange format-version 3.0: reference manual (1992) 1–68.

[41] ISO/IEC 24707:2007 Information technology – Common Logic (CL), standards.iso.org/ittf/PubliclyAvailableStandards/c039175_ISO_IEC_24707_2007%28E%29.zip (2007).

[42] J. McCarthy, Notes on formalizing context, in: Proceedings of the 13th International Joint Conference on Artifical Intelligence - Volume 1, IJCAI'93, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993, pp. 555–560.
URL http://dl.acm.org/citation.cfm?id=1624025.1624103

[43] A. Hogan, M. Arenas, A. Mallea, A. Polleres, Everything you always wanted to know about blank nodes, Web Semantics: Science, Services and Agents on the World Wide Web 27 (2014) 42–69.

[44] C. Bizer, R. Cygniak, RDF 1.1 TriG, w3c Recommendation, http://www.w3.org/TR/trig/ (Feb. 2014).

[45] S. Harris, A. Seaborne, SPARQL 1.1 Query Language, w3c Recommendation, http://www.w3.org/TR/sparql11-query/ (Mar. 2013).

[46] B. C. Pierce, Types and Programming Languages, 1st Edition, The MIT Press, 2002.

[47] M. Sulzmann, M. M. T. Chakravarty, S. P. Jones, K. Donnelly, System f with type equality coercions, in: Proceedings of the 2007 ACM SIGPLAN International Workshop on Types in Languages Design and Implementation, TLDI '07, ACM, New York, NY, USA, 2007, pp. 53–66. doi:10.1145/1190315.1190324.
URL http://doi.acm.org/10.1145/1190315.1190324

[48] A. Igarashi, B. C. Pierce, P. Wadler, Featherweight java: A minimal core calculus for java and gj, ACM Trans. Program. Lang. Syst. 23 (3) (2001) 396–450. doi:10.1145/503502.503505.
URL http://doi.acm.org/10.1145/503502.503505

[49] J. de Bruijn, S. Heymans, Logical foundations of (e)rdf(s): Complexity and reasoning, in: K. Aberer, K.-S. Choi, N. Noy, D. Allemang, K.-I. Lee, L. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, P. Cudré-Mauroux (Eds.), The Semantic Web: 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007. Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 86–99. doi:10.1007/978-3-540-76298-0_7.
URL https://doi.org/10.1007/978-3-540-76298-0_7