Extending Datatype Support in Web Ontology Reasoning

Jeff Z. Pan and Ian Horrocks

Information Management Group
Department of Computer Science
University of Manchester
Oxford Road, Manchester M13 9PL, UK
{pan,horrocks}@cs.man.ac.uk

Abstract. The Semantic Web is a vision of the next generation Web, in which semantic markup will make Web resources more accessible to automatic processes. Description Logics (DLs) are of crucial importance to the development of the Semantic Web, where their role is to provide formal underpinnings and automated reasoning services for Semantic Web ontology languages such as DAML+OIL. In this paper, we show how the description logic $\mathcal{SHOQ}(\mathbf{D})$, which has been designed to provide such services, can be extended with n-ary datatype predicates and qualified number restrictions with n-ary datatype predicates, to give $\mathcal{SHOQ}(\mathbf{D_n})$, and we present an algorithm for deciding the satisfiability of $\mathcal{SHOQ}(\mathbf{D_n})$ concepts, along with a proof of its soundness and completeness. The work is motivated by the requirement for n-ary datatype predicates and qualified number restrictions with n-ary predicates in relation to "real world" properties in semantic Web ontologies and applications.

1 Introduction

The Semantic Web [2] is a vision of the next generation Web, in which the current rendering markup, which specifies how to display Web resources for human consumption, will be enhanced with so called "semantic" markup, which will specify the meaning of web resources so as to make them more accessible to automatic processes.

Description Logics (DLs) are of crucial importance to the development of the Semantic Web, where their role is to provide formal underpinnings and automated reasoning services for semantic Web ontology languages [3, 11] such as DAML+OIL[1] [7, 12]. Significant effort has already been devoted to the investigation of suitable DLs—in particular, Horrocks and Sattler [8] have presented the $\mathcal{SHOQ}(\mathbf{D})$ DL, along with a sound and complete algorithm for deciding concept satisfiability, a basic reasoning service for DLs and ontologies.

A key feature of $\mathcal{SHOQ}(\mathbf{D})$ is that, like DAML+OIL, it supports *datatypes* [1, 8, 9] (e.g., string, integer) as well as the usual abstract concepts (e.g., animal, plant). $\mathcal{SHOQ}(\mathbf{D})$, however, supports a very restricted form of datatypes, i.e., it can only deal with unary datatype predicates. While this is quite close[2] to the requirements of the *current version* of the DAML+OIL language, it is not enough for (even the current version of) DAML+OIL and some semantic Web ontologies and applications.

An approach of extending DL with datatypes was first introduced by Baader and Hanschke [1], who described a datatype ($\mathcal{D}$) extension of the well known $\mathcal{ALC}$ DL. Baader and Hanschke [1] have shown that although the satisfiability of $\mathcal{ALC}(\mathcal{D})$ is decidable, if $\mathcal{ALC}(\mathcal{D})$ is extended with transitive closure of features, the satisfiability problem is undecidable. Lutz [9] proved that reasoning with $\mathcal{ALC}(\mathcal{D})$ and general TBoxes is undecidable. In order to extend *expressive* DLs

[1] http://www.daml.org/

[2] DAML+OIL supports unary datatype predicates *and* qualified number restrictions with unary datatype predicates.

with concrete domains, Horrocks and Sattler [8] proposed a simplified approach on concrete domain and applied this approach on the $\mathcal{SHOQ}(\mathbf{D})$ DL. Pan [10] investigated the simplifying constraints of $\mathcal{SHOQ}(\mathbf{D})$ w.r.t. datatypes, and showed how these could be relaxed in order to extend $\mathcal{SHOQ}(\mathbf{D})$ with n-ary datatype predicates. We should mention that, similar to Baader and Hanschke [1]'s approach, Haarslev et al. [5] extended the $\mathcal{SHN}$ DL with restricted concrete domain $(\mathcal{D})^-$ and gave the $\mathcal{SHN}(\mathcal{D})^-$ DL, which supports n-ary datatype predicates without qualified number restrictions.

In this paper, we extend our work in [10] and add qualified number restrictions with n-ary predicates to give the $\mathcal{SHOQ}(\mathbf{D_n})$ DL, and we present a sound and complete decision procedure for deciding concept satisfiability and subsumption in this logic. The rest of the paper is organised as follows. In Section 2, we introduce some basic concepts of datatypes and give some concrete examples to demonstrate why n-ary datatype predicates, as well as qualified number restrictions with n-ary datatype predicates, are often necessary in semantic Web ontologies and applications. Section 3 briefly compares the two approaches to extend description logics with datatypes. In Section 4, we introduce $\mathcal{SHOQ}(\mathbf{D_n})$, which supports the n-ary datatype predicates and qualified number restrictions with n-ary datatype predicates. In Section 5, we give a tableau algorithm for $\mathcal{SHOQ}(\mathbf{D_n})$ and its decidability proof. Section 6 is a brief discussion of possible future work.

2 Datatypes in the Semantic Web

Datatypes are important in semantic Web ontologies and applications, because most of which need to represent, in some way, various "real world" properties such as size, weight and duration, and some other complex user defined datatypes. Reasoning and querying over datatype properties are important and necessary if these properties are to be understood by machines.

Before we go further, we introduce the following basic concepts (about datatype), which we will use in this paper:

- *Datatype* is a declaration of the kind of data, i.e. integer and string. The datatype determines which constraints/operations can be performed on such data values.
- *Datatype properties* can be used to represent "real world" properties, i.e. size, weight and duration. They have data values, which belong to certain datatypes.
- *Datatype predicates* represent constraints over a list of datatype properties. Usually we call them "n-ary datatype predicates" if the constraints are defined over n datatype properties.

Currently DAML+OIL supports only unary datatype predicates and qualified number restrictions on unary datatype predicates, e.g. you can define a `less than 21 years old` predicate over datatype property *age*. Many people think it is not enough, while n-ary datatype predicates and qualified number restrictions on n-ary predicates are often necessary. Here are some concrete examples:

Example 1 E-ontologies may need to classify items according to their sizes, and to reason that an item which has *height* `less than 5cm` and `the sum of` *length* and *width* `less than 10cm` is a kind of items for which no shipping costs are charged. Here *height*,*length* and *width* are datatype properties and `the sum of ... less than 10cm` is an n-ary datatype predicate.

Example 2 Let's go back to the *age* example. Suppose in an ontology we define a class Person with a datatype property *age*. If one of its instance is John, and John has age 20, then John's age satisfies the predicate `less than 21 years old`. This may be true *now*, but what about in 10 years—is it still true? Of course not, because our ages change every year. One solution is to define three datatype properties *birth-year*, *current-year* and *real age*

and to define a `minus` predicate over them, which is 3-ary, so that we can calculate the *real age* by *birth-year* and *current-year*.

Example 3 E-ontologies may need to classify customers according to the numbers of their friends' email addresses they provide, and decide that those who provide (1) at least 10 *friends' e-addresses*, (2) among them at least 5 *friends' e-addresses* are `from UK` (e.g. man.ac.uk), (3) and at least 5 *friends' e-addresses* `have the same domain` (e.g. hotmail.com) `as` *their e-addresses* can have 5% cash back during promotion. Here *friends' e-addresses* and *their e-addresses* are datatype properties, and `have the same domain as` is an n-ary datatype predicate. Qualified number restrictions with n-ary predicates are also used in this example. They are very expressive in the sense that they force restrictions on the numbers of the tuples of data values that satisfy the datatype predicate, instead of on the numbers of a certain datatype property. Therefore they are quite different from the qualified number restrictions that we have in the $\mathcal{SHOQ}(\mathbf{D})$ DL and become new challenges to DL reasoners. (See section 5 to find out how we solve this problem.)

From these examples, we can see n-ary datatype predicates and qualified number restrictions with n-ary predicates are very important and useful in semantic Web ontologies and applications. On the one hand, it is very desirable to express business rules [4] or constraints for datatype properties in semantic Web ontologies and applications. On the other hand, too expressive rules and constraints could easily introduce undecidability. In this paper, we prove that adding n-ary datatype predicates as well as qualified number restrictions with n-ary predicates to the $\mathcal{SHOQ}(\mathbf{D})$ DL, which is quite suited to the provision of reasoning support for DAML+OIL, does not change the decidability property of the $\mathcal{SHOQ}(\mathbf{D})$ DL. Thus we suggest that future versions of DAML+OIL, or OWL[3] can include these extended datatype constructs without worrying the decidability problem.

3 Description Logics and Datatypes

For many knowledge representation application, it is essential to integrate reasoning about datatype properties with reasoning about knowledge represented at the abstract level. One theoretically well-founded approach, first described by Baader and Hanschke [1] is to extend a DL with *concrete domains*. Concrete domains were defined as follows:

Definition 1 A concrete domain is a pair $(\Delta_{\mathcal{D}}, \Phi_{\mathcal{D}})$, where $\Delta_{\mathcal{D}}$ is a set called the domain and $\Phi_{\mathcal{D}}$ is a set of predicate names. Each predicate name P is associated with an arity n, and an n-ary predicate $P^{\mathcal{D}} \subseteq dom(\mathcal{D})^n$. A concrete domain is called *admissible* iff

1. the set of its predicate names is closed under negation and contains a name $\top_{\mathcal{D}}$ for $\Delta_{\mathcal{D}}$, and
2. the satisfiability problem for an expression of the form $P_1^{\mathcal{D}} \cap \ldots \cap P_n^{\mathcal{D}}$ is decidable.

In order to extend *expressive* DLs with concrete domains, Horrocks and Sattler [8] proposed a simplified approach on concrete domain and give the $\mathcal{SHOQ}(\mathbf{D})$ DL. Pan [10] investigated the simplifying constraints introduced in [8] and extend the $\mathcal{SHOQ}(\mathbf{D})$ DL with n-ary predicates. One of the simplifying constraints about concrete datatype is that $\Delta_{\mathbf{D}}$ is defined as the domain of *all* concrete datatypes, instead of any *specific* concrete domain $\Delta_{\mathcal{D}}$.

Here we give a formal definition of (some of) the concepts we introduced in Section 2.

Definition 2 $\Delta_{\mathbf{D}}$ is the *datatype domain* covering all concrete datatypes. A set of datatypes $d_1 \ldots, d_m$ can be defined in $\Delta_{\mathbf{D}}$, whose domain are subsets of the datatype domain, $\forall i : 1 \leq$

[3] `http://www.w3.org/2001/sw/WebOnt/`

$i \leq m, d_i^{\mathbf{D}} \subseteq \Delta_{\mathbf{D}}$. Datatype predicates $P_1, \ldots, P_r$ can be defined over these datatypes, and each predicate P_j is associated with an arity n_j, where $\forall j : 1 \leq j \leq r, P_j^{\mathbf{D}} \subseteq d_{j1}^{\mathbf{D}} \times \ldots \times d_{jn_j}^{\mathbf{D}} \subseteq \Delta_{\mathbf{D}}^{n_j}$.

Each such datatype predicate $P_j(v_1, \ldots, v_{n_j})$ is a boolean function, where $v_1, \ldots, v_{n_j}$ are its input variables. It returns true iff there exists a tuple of data values $t_1, \ldots, t_{n_j}$ in $\Delta_{\mathbf{D}}$, s.t. $\forall i : 1 \leq i \leq n_j, v_i^{\mathbf{D}} = t_i$ and $\langle t_1, \ldots, t_{n_j} \rangle \in P_j^{\mathbf{D}}$.

The negation of datatype predicate $P_j(v_1, \ldots, v_{n_j})$, i.e. $\neg P_j(v_1, \ldots, v_{n_j})$, is also a boolean function. It returns true iff. $P_j(v_1, \ldots, v_{n_j})$ returns false.

An example of datatype is INT+, which has the set of nonnegative integers as its domain. A binary predicate $\geq$ and a unary predicate $\geq_n$ are defined over INT+.

Definition 3 A set of datatypes $d_1 \ldots, d_m$ is *conforming* if

1. the datatype domain $\Delta_{\mathbf{D}}$ is disjoint with the object domain $\Delta^{\mathcal{I}}$, and
2. a required binary inequality-checking predicate $\neq_i \in \{P_1, \ldots, P_r\}$, is defined for each datatype d_i, and
3. there exists a sound and complete decision procedure for checking if a set of tuples of data values satisfy a set of related datatype predicates $P_1, \ldots, P_r$.

The datatype INT+ is not conforming, since it doesn't have a built-in binary inequality-checking predicate $\neq$. Thus any set of datatypes which has INT+ as its member is not conforming due to the second condition in Definition 3. Please note that according to Definition 2, datatype predicates can be defined over different datatypes, and Definition 3 requires that each conforming datatype have a built-in binary inequality-checking predicate $\neq$.

In the simplified approach, *type systems* are introduced to define datatypes and datatype predicates, including the built-in binary inequality-checking predicates for all supported datatypes. Type systems are used with DL reasoners, and can answer boolean questions, e.g. to check if a set of tuples of data values satisfy a set of predicates simultaneously. With type systems, we can deal with an arbitrary conforming set of datatypes and predicates without compromising the compactness of the concept language or the soundness and completeness of our decision procedure [8].

Another main difference between the two approaches, besides the definition of the datatype domain, is that in the new approach, datatype properties are roles instead of features (functional roles), which makes it possible to have qualified number restrictions on n-ary predicates—while with features, the number restriction is already set as less than or equal to 1.

In next section, we will give the definition of the $\mathcal{SHOQ}(\mathbf{D_n})$ DL. Please note that in DLs, "classes" are usually called "concepts", "properties" are often called "roles", and "datatype properties" are also called "concrete roles".

4 $\mathcal{SHOQ}(\mathbf{D_n})$

Definition 4 Let $\mathbf{C}$, $\mathbf{R} = \mathbf{R}_A \uplus \mathbf{R_D}$, $\mathbf{I}$ be disjoint sets of concept, abstract and concrete role and individual names. For R and S roles, a *role axiom* is either a role inclusion, which is of the form $R \sqsubseteq S$ for $R, S \in \mathbf{R}_A$ or $R, S \in \mathbf{R_D}$, or a transitivity axiom, which is of the form $\mathsf{Trans}(R)$ for $R \in \mathbf{R}_A$. A *role box* $\mathcal{R}$ is a finite set of role axioms. A role R is called *simple* if, for $\sqsubseteq^*$ the transitive reflexive closure of $\sqsubseteq$ on $\mathcal{R}$ and for each role S, $S \sqsubseteq^* R$ implies $\mathsf{Trans}(S) \notin \mathcal{R}$.

The set of concept terms of $\mathcal{SHOQ}(\mathbf{D_n})$ is inductively defined. As a starting point of the induction, any element of $\mathbf{C}$ is a concept term (atomic terms). Now let C and D be concept terms, o be an individual, R be a abstract role name, $T_1, \ldots, T_n$ be concrete role names, P be an n-ary datatype predicate name. Then the following expressions are also concept terms:

1. $\top$ (universal concept) and $\top_{\mathtt{D}}$ (universal datatype),

Construct Name	Syntax	Semantics
universal datatype	$\top_{\mathtt{D}}$	$\top_{\mathtt{D}}^{\mathbf{D}} = \Delta_{\mathbf{D}}$
datatype exists	$\exists T_1, \cdots, T_n.P_n$	$(\exists T_1, \cdots, T_n.P_n)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \exists y_1 \cdots y_n.$ $\langle x, y_1\rangle \in T_1^{\mathcal{I}} \wedge \cdots \wedge \langle x, y_n\rangle \in T_n^{\mathcal{I}} \wedge \langle y_1, \cdots y_n\rangle \in P_n^{\mathbf{D}}\}$
datatype value	$\forall T_1, \cdots, T_n.P_n$	$(\forall T_1, \cdots, T_n.P_n)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \forall y_1 \cdots y_n.$ $\langle x, y_1\rangle \in T_1^{\mathcal{I}} \wedge \cdots \wedge \langle x, y_n\rangle \in T_n^{\mathcal{I}} \rightarrow \langle y_1, \cdots y_n\rangle \in P_n^{\mathbf{D}}\}$
datatype atleast	$\geqslant m T_1, \ldots, T_n.P_n$	$(\geqslant m T_1, \ldots, T_n.P_n)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \sharp\{\langle y_1 \cdots y_n\rangle \mid$ $\langle x, y_1\rangle \in T_1^{\mathcal{I}} \wedge \cdots \wedge \langle x, y_n\rangle \in T_n^{\mathcal{I}} \wedge \langle y_1, \cdots y_n\rangle \in P_n^{\mathbf{D}}\} \geq m\}$
datatype atmost	$\leqslant m T_1, \ldots, T_n.P_n$	$(\leqslant m T_1, \ldots, T_n.P_n)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \sharp\{\langle y_1 \cdots y_n\rangle \mid$ $\langle x, y_1\rangle \in T_1^{\mathcal{I}} \wedge \cdots \wedge \langle x, y_n\rangle \in T_n^{\mathcal{I}} \wedge \langle y_1, \cdots y_n\rangle \in P_n^{\mathbf{D}}\} \leq m\}$
concrete role atleast	$\geqslant m T$	$(\geqslant m T)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \sharp\{y \in \Delta_{\mathbf{D}} \mid \langle x, y\rangle \in T\} \geq m\}$
concrete role atmost	$\leqslant m T$	$(\leqslant m T)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \sharp\{y \in \Delta_{\mathbf{D}} \mid \langle x, y\rangle \in T\} \leq m\}$

Fig. 1. Datatype constructs in $\mathcal{SHOQ}(\mathbf{D_n})$

2. $C \sqcup D$ (disjunction), $C \sqcap D$ (conjunction), $\neg C$ (negation), and $\{o\}$ (nominals),
3. $\exists R.C$ (exists-in restriction) and $\forall R.C$ (value restriction),
4. $\geqslant m R.C$ (atleast restriction) and $\leqslant m R.C$ (atmost restriction),
5. $\exists T_1, \cdots, T_n.P_n$ (datatype exists) and $\forall T_1, \cdots, T_n.P_n$ (datatype value),
6. $\geqslant m T_1, \ldots, T_n.P_n$ (datatype atleast) and $\leqslant m T_1, \ldots, T_n.P_n$ (datatype atmost),
7. $\geqslant m T$ (concrete role atleast), $\leqslant m T$ (concrete role atmost).

$\mathcal{SHOQ}(\mathbf{D_n})$ extends $\mathcal{SHOQ}(\mathbf{D})$ by supporting n-ary datatype predicates P_n. The interpretations of datatype constructs are listed in Figure 1. Note that concrete role atleast (atmost) is only a special form of datatype atleast (atmost, respectively) where $n = 1$ and $P_n = \top_{\mathtt{D}}$.

To illustrate the use of $\mathcal{SHOQ}(\mathbf{D_n})$-concept, let's go back to the examples we used in Section 2:

1. Items with height less than 5cm, and the sum of their length and width less that 10cm can be defined as a $\mathcal{SHOQ}(\mathbf{D_n})$-concept

$$Item \sqcap = 1 height.<_{\mathtt{5cm}} \sqcap = 1 length \sqcap = 1 width \ \sqcap \forall length, width.\mathtt{sum} <_{\mathtt{10cm}}$$

   where "=1" is a shortcut for "$\leqslant 1 \sqcap \geqslant 1$", and *height*, *length* and *width* are concrete roles, $<_{\mathtt{5cm}}$ is a unary datatype predicate and $\mathtt{sum} <_{\mathtt{10cm}}$ is a binary predicate. Note that $<_{\mathtt{5cm}}$ and $\mathtt{sum} <_{\mathtt{10cm}}$ are datatype predicates, rather than datatype atmost.
2. Persons who are younger than 21 years old can be defined as a $\mathcal{SHOQ}(\mathbf{D_n})$-concept

$$Person \sqcap = 1 birth{-}year \sqcap = 1 current{-}year \sqcap = 1 age.<_{21} \sqcap \forall birth{-}year, current{-}date, age.\mathtt{minus}$$

   where `minus` is a trinary datatype predicate, which can be defined in the type system as lambda expression $\lambda(x, y, z)(z = y - x)$.
3. Customers in example 3 can be defined as a $\mathcal{SHOQ}(\mathbf{D_n})$-concept

$$Customer \sqcap \geqslant 10 friends - eaddress \sqcap \geqslant 5 friends - eaddress.\mathtt{from} - \mathtt{UK} \ \sqcap$$
$$\geqslant 5 friends - eaddress, eaddress.\mathtt{same} - \mathtt{domain}$$

   where $\geqslant 10 friends - eaddress$ is a concrete role atleast, and $\geqslant 5 friends - eaddress.\mathtt{from} - \mathtt{UK}$ and $\geqslant 5 friends - eaddress, eaddress.\mathtt{same} - \mathtt{domain}$ are datatype atleast.

Now we define a tableau for $\mathcal{SHOQ}(\mathbf{D_n})$. For ease of presentation, we assume all concepts to be in *negation normal form* (NNF). We use $\sim C$ to denote the NNF of $\neg C$. Moreover, for a concept D, we use $\mathsf{cl}(D)$ to denote the set of all sub-concepts of D, the NNF of these sub-concepts, and the (possibly negated) datatypes occurring in these (NNF of) sub-concepts.

Definition 5 If D is a $\mathcal{SHOQ}(\mathbf{D_n})$-concept in NNF, $\mathcal{R}$ a role box, and $\mathbf{R}_A^D$, $\mathbf{R}_\mathbf{D}^D$ are the sets of abstract and concrete roles occurring in D or $\mathcal{R}$, a tableau $\mathcal{T}$ for D w.r.t. $\mathcal{R}$ is defined as a quadruple $(\mathbf{S}, \mathcal{L}, \mathcal{E}_A, \mathcal{E}_\mathbf{D})$ such that: $\mathbf{S}$ is a set of individuals, $\mathcal{L} : \mathbf{S} \to 2^{\mathsf{cl}(D)}$ maps each individual to a set of concepts which is a subset of $\mathsf{cl}(D)$, $\mathcal{E}_A : \mathbf{R}_A^D \to 2^{\mathbf{S}\times\mathbf{S}}$ maps each abstract role in $\mathbf{R}_A^D$ to a set of pairs of individuals, $\mathcal{E}_\mathbf{D} : \mathbf{R}_\mathbf{D}^D \to 2^{\mathbf{S}\times\Delta_\mathbf{D}}$ maps each concrete role in $\mathbf{R}_\mathbf{D}^D$ to a set of pairs of individuals and concrete values, and there is some individual $s \in \mathbf{S}$ such that $D \in \mathcal{L}(s)$. For all $s, t \in \mathbf{S}$, $C, C_1, C_2 \in cl(D)$, $R, S \in \mathbf{R}_A^D$, $T, T', T_1, \ldots, T_n \in \mathbf{R}_\mathbf{D}^D$, n-ary predicate P_n and

$$S^\mathcal{T}(s, C) := \{t \in \mathbf{S} \mid \langle s, t\rangle \in \mathcal{E}_A(S) \text{ and } C \in \mathcal{L}(t)\},$$
$$T_1T_2 \ldots T_n^\mathcal{T}(s, P_n) := \{\langle y_1, \ldots, y_n\rangle \in P_n^\mathbf{D} \mid \langle s, y_1\rangle \in \mathcal{E}_\mathbf{D}(T_1), \ldots, \langle s, y_n\rangle \in \mathcal{E}_\mathbf{D}(T_n)\},$$
$$DC^\mathcal{T}(s, T_1, \ldots, T_n, y_1, \ldots, y_n, P_n) := \begin{cases} true & \text{if } \langle s, y_i\rangle \in \mathcal{E}_\mathbf{D}(T_n)(1 \le i \le n) \text{ and} \\ & \langle y_1, \ldots, y_n\rangle \in P_n^\mathbf{D} \\ false & \text{otherwise} \end{cases}$$

it holds that:

(P1) if $C \in \mathcal{L}(s)$, then $\neg C \notin \mathcal{L}(s)$,
(P2) if $C_1 \sqcap C_2 \in \mathcal{L}(s)$, then $C_1 \in \mathcal{L}(s)$ and $C_2 \in \mathcal{L}(s)$,
(P3) if $C_1 \sqcup C_2 \in \mathcal{L}(s)$, then $C_1 \in \mathcal{L}(s)$ or $C_2 \in \mathcal{L}(s)$,
(P4) if $\langle s, t\rangle \in \mathcal{E}_A(R)$ and $R \sqsubseteq^* S$, then $\langle s, t\rangle \in \mathcal{E}_A(S)$,
if $\langle s, t\rangle \in \mathcal{E}_\mathbf{D}(T)$ and $T \sqsubseteq^* T'$, then $\langle s, t\rangle \in \mathcal{E}_\mathbf{D}(T')$,
(P5) if $\forall R.C \in \mathcal{L}(s)$ and $\langle s, t\rangle \in \mathcal{E}_A(R)$, then $C \in \mathcal{L}(t)$,
(P6) if $\exists R.C \in \mathcal{L}(s)$, then there is some $t \in \mathbf{S}$ such that $\langle s, t\rangle \in \mathcal{E}_A(R)$ and $C \in \mathcal{L}(t)$,
(P7) if $\forall S.C \in \mathcal{L}(s)$ and $\langle s, t\rangle \in \mathcal{E}_A(R)$ for some $R \sqsubseteq^* S$ with $\mathsf{Trans}(R)$, then $\forall R.C \in \mathcal{L}(t)$,
(P8) if $\geqslant nS.C \in \mathcal{L}(s)$, then $\sharp S^T(s, C) \geqslant n$,
(P9) if $\leqslant nS.C \in \mathcal{L}(s)$, then $\sharp S^T(s, C) \leqslant n$,
(P10) if $\{\leqslant nS.C, \geqslant nS.C\} \cap \mathcal{L}(s) \neq \emptyset$ and $\langle s, t\rangle \in \mathcal{E}_A(S)$, then $\{C, \sim C\} \cap \mathcal{L}(t) \neq \emptyset$,
(P11) if $\{o\} \in \mathcal{L}(s) \cap \mathcal{L}(t)$, then $s = t$,
(P12) if $\forall T_1, \cdots, T_n.P_n \in \mathcal{L}(s)$ and $\langle s, t_1\rangle \in \mathcal{E}_\mathbf{D}(T_1), \cdots, \langle s, t_n\rangle \in \mathcal{E}_\mathbf{D}(T_n)$, then $DC^\mathcal{T}(s, T_1, \ldots, T_n, t_1, \ldots, t_n, P_n) = true$,
(P13) if $\exists T_1, \cdots, T_n.P_n \in \mathcal{L}(s)$, then there is some $t_1, \cdots, t_n \in \Delta_\mathbf{D}$ such that $\langle s, t_1\rangle \in \mathcal{E}_\mathbf{D}(T_1), \cdots, \langle s, t_n\rangle \in \mathcal{E}_\mathbf{D}(T_n)$, $DC^\mathcal{T}(s, T_1, \ldots, T_n, t_1, \ldots, t_n, P_n) = true$,
(P14) if $\geqslant mT_1, \ldots, T_n.P_n \in \mathcal{L}(s)$, then $\sharp T_1T_2 \ldots T_n^\mathcal{T}(s, P_n) \geqslant m$,
(P15) if $\leqslant mT_1, \ldots, T_n.P_n \in \mathcal{L}(s)$, then $\sharp T_1T_2 \ldots T_n^\mathcal{T}(s, P_n) \leqslant m$,
(P16) if $\{\leqslant mT_1, \ldots, T_n.P_n, \geqslant mT_1, \ldots, T_n.P_n\} \cap \mathcal{L}(s) \neq \emptyset$ and $\langle s, t_1\rangle \in \mathcal{E}_\mathbf{D}(T_1), \ldots, \langle s, t_n\rangle \in \mathcal{E}_\mathbf{D}(T_n)$, then for $1 \le i \le n$, we have either $DC^\mathcal{T}(s, T_1, \ldots, T_n, t_1, \ldots, t_n, P_n) = true$, or $DC^\mathcal{T}(s, T_1, \ldots, T_n, t_1, \ldots, t_n, \neg P_n) = true$.

Lemma 1. *A $\mathcal{SHOQ}(\mathbf{D_n})$-concept D in NNF is satisfiable w.r.t. a role box $\mathcal{R}$ iff D has a tableau w.r.t. $\mathcal{R}$.*

Proof: For the *if* direction, if $\mathcal{T} = (\mathbf{S}, \mathcal{L}, \mathcal{E}_A, \mathcal{E}_\mathbf{D})$ is a tableau for D, a model $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ of D can be defined as: $\Delta^\mathcal{I} = \mathbf{S}$, $\mathsf{CN}^\mathcal{I} = \{s \mid \mathsf{CN} \in \mathcal{L}(s)\}$ for all concept names CN in $\mathtt{cl}(D)$, if $R \in \mathbf{R}_+$, $\mathbf{R}_A^\mathcal{I} = \mathcal{E}_A(R)^+$, otherwise $\mathbf{R}_A^\mathcal{I} = \mathcal{E}_A(R) \cup \bigcup_{P \sqsubseteq^* R, P \neq R} P^\mathcal{I}$, $\mathbf{R}_\mathbf{D}^\mathcal{I} = \mathcal{E}_D(R)$, where $\mathcal{E}_A(R)^+$ denotes the transitive closure of $\mathcal{E}_A(R)$. $D^\mathcal{I} \neq \emptyset$ because $s_0 \in D^\mathcal{I}$. Here we only concentrate on ($\mathsf{P14}$) to ($\mathsf{P15}$); the remainder is similar to the proofs found in [10][4].

1. $E = \geqslant mT_1, \ldots, T_n.P_n$. According to ($\mathsf{P14}$), $E \in \mathcal{L}(s)$ implies that $\sharp T_1T_2 \ldots T_n^\mathcal{T}(s, P_n) \geqslant m$. By the definition of $T_1T_2 \ldots T_n^\mathcal{T}(s, P_n)$, we have $s \in \{x \in \Delta^\mathcal{I} \mid \sharp\{\langle t_1, \ldots, t_n\rangle \mid \langle x, t_1\rangle \in \mathcal{E}_\mathbf{D}(T_1) \wedge \ldots \wedge \langle x, t_n\rangle \in \mathcal{E}_\mathbf{D}(T_n) \wedge \langle t_1, \ldots, t_n\rangle \in P_n^\mathbf{D}\} \geq m\}$, Since $\mathcal{E}_\mathbf{D}(T_i) = T_i^\mathcal{I}$, we have $s \in (\geqslant mT_1, \ldots, T_n.P_n)^\mathcal{I}$. Similarly, if $E = \leqslant mT_1, \ldots, T_n.P_n$, we have $s \in (\leqslant mT_1, \ldots, T_n.P_n)^\mathcal{I}$.

[4] Note that in this paper, we mainly focus on the proof of the number restriction on concrete roles, the remainder is similar to the proofs found in [10].

For the converse, if $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is a model of D, then a tableau $\mathcal{T} = (\mathbf{S}, \mathcal{L}, \mathcal{E}_A, \mathcal{E}_{\mathbf{D}})$ for D can be defined as: $S = \Delta^{\mathcal{I}}$, $\mathcal{E}_A(R) = R_A^{\mathcal{I}}$, $\mathcal{E}_{\mathbf{D}}(R) = R_D^{\mathcal{I}}$, $\mathcal{L}(s) = \{C \in \mathtt{cl}(D) \mid s \in C^{\mathcal{I}}\}$. It only remains to demonstrate that T is a tableau for D: $\mathcal{T}$ satisfies (P14) to (P16) as a direct consequence of the semantics of datatype constructs. □

5 A Tableau Algorithm for $\mathcal{SHOQ}(\mathbf{D_n})$

Form Lemma 1, an algorithm which constructs a tableau for a $\mathcal{SHOQ}(\mathbf{D_n})$-concept D can be used as a decision procedure for the satisfiability of D with respect to a role box $\mathcal{R}$.

Definition 6 Let $\mathcal{R}$ be a role box, D a $\mathcal{SHOQ}(\mathbf{D_n})$-concept in NNF, $\mathbf{R}_A^D$ the set of abstract roles occurring in D or $\mathcal{R}$, and $\mathbf{I}^D$ the set of nominal occurring in D. A tableaux algorithm works on a *completion forest* for D w.r.t. $\mathcal{R}$, which is a set of trees $\mathtt{F}$. Each node x of the forest is labelled with a set

$$\mathcal{L}(x) \subseteq \mathtt{cl}(D) \cup \{\uparrow (R, \{o\}) \mid R \in \mathbf{R}_A^D \text{ and } \{o\} \in \mathbf{I}^D\},$$

and each edge $\langle x, y\rangle$ is labelled with a set of role names $\mathcal{L}(\langle x, y\rangle)$ containing roles occurring in $\mathrm{cl}(D)$ or $\mathcal{R}$. A node x is either an abstract node or a concrete node, the latter one represents a data value, and is always a leaf of $\mathtt{F}$. Additionally, we keep track of inequalities between nodes of the tree with a symmetric binary relation $\neq$, which can be used between two abstract nodes, or between two tuples of concrete nodes of $\mathtt{F}$. Inequality relations between tuples of concrete nodes can be mapped to inequality-checking predicates provided by the type systems[5]. For each $\{o\} \in \mathbf{I}^D$ there is a *distinguished* node $x_{\{o\}}$ in $\mathtt{F}$ such that $\{o\} \in \mathcal{L}(x)$. The algorithm expands the forest either by extending $\mathcal{L}(x)$ for some node x or by adding new leaf nodes.

Given a completion forest, a node y is called an *R-successor* of a node x if, for some $R^{'}$ with $R^{'} \sqsubseteq^{*} R$, either y is a successor of x and $R^{'} \in \mathcal{L}(\langle x, y\rangle)$, or $\uparrow (R^{'}, \{o\}) \in \mathcal{L}(x)$ and $y = x_{\{o\}}$. Ancestors and roots are defined as usual. For an abstract role S and a node x in $\mathtt{F}$ we define $S^{\mathtt{F}}(x, C)$ by

$$S^{\mathtt{F}}(x, C) := \{y \mid y \text{ is an } S\text{-successor of x and } C \in \mathcal{L}(y)\}.$$

Given a completion forest, concrete nodes $t_1, \ldots, t_n$ are called $T_1T_2 \ldots T_n$-*successors* of a node x if, for some concrete roles $T_1^{'}, \ldots, T_n^{'}$ with $T_i^{'} \sqsubseteq^{*} T_i$, $t_1, \ldots, t_n$ are successors of x and $T_i^{'} \in \mathcal{L}(\langle x, t_i\rangle), 1 \leq i \leq n$. For a node x, its $T_1T_2 \ldots T_n$-successors $\langle t_1, \ldots, t_n\rangle$, n-ary datatype predicate P_n, we define a set $DC^{\mathtt{F}}$ by

$$DC^{F} = \{< DCelement >\}$$

where $DC^{\mathtt{F}}$ is a set of $DCelements$, which have the form

$$< DCelement >= \{x, \langle T_1, \ldots, T_n\rangle, \langle t_1, \ldots, t_n\rangle, P_n\}$$

$DC^{\mathtt{F}}$ is initialised as an empty set. $DC^{\mathtt{F}}$is is *satisfied* iff. (i) there exists $value : N_C \rightarrow \Delta_{\mathbf{D}}$, where N_C is the set of all concrete nodes, s.t. for all $\{x, \langle T_1, \ldots, T_n\rangle, \langle t_1, \ldots, t_n\rangle, P_n\} \in DC^{\mathtt{F}}$, $\langle value(t_1), \ldots, value(t_n)\rangle \in P_n^{\mathbf{D}}$ are true; (ii) the inequality relations between tuples of concrete nodes are satisfied. In order to retrieve the set of all the $T_1T_2 \ldots T_n$-successors of x, which satisfy a certain predicate P_n, we define $DCSuccessors^{\mathtt{F}}(x, P_n)$ by

$$DCSuccessors^{\mathtt{F}}(x, T_1, \ldots, T_n, P_n) := \{\langle t_1, \ldots, t_n\rangle \mid \{x, \langle T_1, \ldots, T_n\rangle, \langle t_1, \ldots, t_n\rangle, P_n\} \in DC^{\mathtt{F}}\}$$

In order to retrieve the set of datatype predicates, which are satisfied by $T_1T_2 \ldots T_n$-successors $t_1, \ldots, t_n$ of x, we define $DCPredicates^{\mathtt{F}}(x, T_1, \ldots, T_n, t_1, \ldots, t_n)$by

$$DCPredicates^{\mathtt{F}}(x, T_1, \ldots, T_n, t_1, \ldots, t_n) := \{P_n \mid \{x, \langle T_1, \ldots, T_n\rangle, \langle t_1, \ldots, t_n\rangle, P_n\} \in DC^{\mathtt{F}}\}$$

[5] See the definition of conforming datatypes in section 3

$\forall_P$-rule:	if 1.$\forall T_1, \cdots, T_n.P_n \in \mathcal{L}(x)$, x is not blocked, and 2.there are $T_1T_2 \ldots T_n$-successors $\langle t_1, \ldots, t_n \rangle$ of x with $P_n \notin DCPredicates^{\mathtt{F}}(x, T_1, \ldots, T_n, t_1, \ldots, t_n)$, then $DC^{\mathtt{F}} \longrightarrow DC^{\mathtt{F}} \cup \{x, \langle T_1, \ldots, T_n \rangle, \langle t_1, \ldots, t_n \rangle, P_n\}$.
$\exists_P$-rule:	if 1.$\exists T_1, \cdots, T_n.P_n \in \mathcal{L}(x)$, x is not blocked, and 2.there are no $T_1T_2 \ldots T_n$-successors $\langle t_1, \ldots, t_n \rangle$ of x, with $P_n \in DCPredicates^{\mathtt{F}}(x, T_1, \ldots, T_n, t_1, \ldots, t_n)$, then 1. create $T_1T_2 \ldots T_n$-successors $\langle t_1, \cdots, t_n \rangle$ with $\mathcal{L}(\langle x, t_i \rangle) = \{Ti\}$ for $1 \leq i \leq n$ and 2. $DC^{\mathtt{F}} \longrightarrow DC^{\mathtt{F}} \cup \{x, \langle T_1, \ldots, T_n \rangle, \langle t_1, \ldots, t_n \rangle, P_n\}$.
$\geqslant_P$-rule:	if 1.$\geqslant m T_1, \cdots, T_n.P_n \in \mathcal{L}(x)$, x is not blocked, and 2.there is no m $T_1T_2 \ldots T_n$-successors $\langle t_{11}, \ldots, t_{1n} \rangle, \cdots, \langle t_{m1}, \ldots, t_{mn} \rangle$, such that $\{x, \langle T_1, \ldots, T_n \rangle, \langle t_{j1}, \ldots, t_{jn} \rangle, P_n\} \in DC^{\mathtt{F}}$, and $\langle t_{j1}, \ldots, t_{jn} \rangle \dot{\neq} \langle t_{k1}, \ldots, t_{kn} \rangle$, for all $1 \leq j < k \leq m$ then 1. create m $T_1T_2 \ldots T_n$-successors $\langle t_{11}, \ldots, t_{1n} \rangle, \cdots, \langle t_{m1}, \ldots, t_{mn} \rangle$, with $\mathcal{L}(\langle x, t_{ji} \rangle) \longrightarrow \{Ti\}$, and 2. $DC^{\mathtt{F}} \longrightarrow DC^{\mathtt{F}} \cup \{x, \langle T_1, \ldots, T_n \rangle, \langle t_{j1}, \ldots, t_{jn} \rangle, P_n\}$ and 3. set $\langle t_{j1}, \ldots, t_{jn} \rangle \dot{\neq} \langle t_{k1}, \ldots, t_{kn} \rangle$, for all $1 \leq i \leq n$, $1 \leq j < k \leq m$.
$\leqslant_P$-rule:	if 1.$\leqslant m T_1, \cdots, T_n.P_n \in \mathcal{L}(x)$, x is not blocked, and 2.$\sharp DCSuccessors^{\mathtt{F}}(x, T_1, \ldots, T_n, P_n) > m$ and 3.there exist $j \neq k$, s.t. $\langle t_{j1}, \ldots, t_{jn} \rangle, \langle t_{k1}, \ldots, t_{kn} \rangle \in DCSuccessors^{\mathtt{F}}(x,$ $T_1, \ldots, T_n, P_n)$ but not $\langle t_{j1}, \ldots, t_{jn} \rangle \dot{\neq} \langle t_{k1}, \ldots, t_{kn} \rangle, 1 \leq j < k \leq m+1$, then 1. $\mathcal{L}(\langle x, t_{ki} \rangle) \longrightarrow \mathcal{L}(\langle x, t_{ki} \rangle) \cup \mathcal{L}(\langle x, t_{ji} \rangle)$, and 2. $DC^{\mathtt{F}} \longrightarrow DC^{\mathtt{F}}[\langle t_{j1}, \ldots, t_{jn} \rangle / \langle t_{k1}, \ldots, t_{kn} \rangle] \mid_{x, T_1, \ldots, T_n, P_n}$, and 3. add $u \dot{\neq} \langle t_{k1}, \ldots, t_{kn} \rangle$ for each tuple u with $u \dot{\neq} \langle t_{j1}, \ldots, t_{jn} \rangle$, and 4. remove all t_{ji} where t_{ji} isn't in any tuples of$DCSuccessors^{\mathtt{F}}(x, *, *)$ and remove all edges leading to these t_{ji} from $\mathtt{F}$.
$choose_P$-rule:	if 1.$\{\leqslant m T_1, \cdots, T_n.P_n, \geqslant m T_1, \cdots, T_n.P_n\} \cap \mathcal{L}(x) \neq \emptyset$, x is not blocked, and 2.$\langle t_1, \ldots, t_n \rangle$ are $T_1T_2 \ldots T_n$-successors of x, and then either $DC^{\mathtt{F}} \longrightarrow DC^{\mathtt{F}} \cup \{x, \langle T_1, \ldots, T_n \rangle, \langle t_1, \ldots, t_n \rangle, P_n\}$, or $DC^{\mathtt{F}} \longrightarrow DC^{\mathtt{F}} \cup \{x, \langle T_1, \ldots, T_n \rangle, \langle t_1, \ldots, t_n \rangle, \neg P_n\}$.

Fig. 2. The Tableaux Expansion Rules for $\mathcal{SHOQ}(\mathbf{D_n})$ (I)

Note that we can use * as parameter in $DCSuccessors^{\mathtt{F}}$ and $DCPredicates^{\mathtt{F}}$, e.g. $DCSuccessors^{\mathtt{F}}(x, *, *)$ means all the concrete successors of node x.

A node x is *directly blocked* if none of its ancestors are blocked, and it has an ancestor $x^{'}$ that is not distinguished such that $\mathcal{L}(x) \subseteq \mathcal{L}(x^{'})$. We call $x^{'}$ blocks x. A node is *blocked* if it is directly blocks or if its predecessor is blocked.

If $\{o_1\}, \cdots, \{o_l\}$ are all individuals occurring in D, the algorithm initialises the completion forest $\mathtt{F}$ to contain $l+1$ root nodes $x_0, x_{\{o_1\}}, \cdots, x_{\{o_l\}}$ with $\mathcal{L}(x_0) = \{D\}$ and $\mathcal{L}(x_{\{o_i\}}) = \{\{o_i\}\}$. The inequality relation $\dot{\neq}$ is initialised with the empty relation. $\mathtt{F}$ is then expended by repeatedly applying the *expansion rules*, listed in Figure 2[6], stopping if a *clash* occurs in one of its nodes.

For a node x, $\mathcal{L}(x)$ is said to contain a *clash* if:

1. for some concept name $A \in N_C$, $\{A, \neg A\} \subseteq \mathcal{L}(x)$, or
2. for some role S, $\leqslant S.C \in \mathcal{L}(x)$ and there are $n+1$ S-successors $y_0, \cdots, y_n$ of x with $C \in \mathcal{L}(y_i)$ for each $0 \leq i \leq n$ and $y_i \neq y_j$ for each $0 \leq i < j \leq n$, or
3. $DC^{\mathtt{F}}$ isn't satisfied;
4. for some concrete roles $T_1, \ldots, T_n$, n-ary datatype predicate P_n, $\leqslant m T_1, \ldots, T_n.P_n \in \mathcal{L}(x)$, we have $\sharp DCSuccessors^{\mathtt{F}}(x, T_1, \ldots, T_n, P_n) \geqslant m+1$, or

[6] Figure 2 only lists the rules about datatypes, other rules can be found in [10].

5. for some $\{o\} \in \mathcal{L}(x), x \dot{\neq} x_{\{o\}}$.

The completion forest is *complete* when, for some node x, $\mathcal{L}(x)$ contains a clash, or when none of the expansion rules is applicable. If the expansion rules can be applied in such a way that they yield a complete, clash-free completion forest, then the algorithm returns "D is *satisfiable* w.r.t. $\mathcal{R}$", and "D is *unsatisfiable* w.r.t. $\mathcal{R}$" otherwise.

Lemma 2. (Termination) *When started with a $\mathcal{SHOQ}(\mathbf{D_n})$-concept D in NNF, the tableau algorithm terminates.*

Proof: Let $d = |cl(D)|$, $k = |\mathbf{R}_A^D|$, n_{max} the maximal number in atleast number restrictions as well as datatype atleast, and $\ell = |\mathcal{I}^D|$. Here we mainly concentrate on rules about number restriction on concrete roles. Note that the set of datatypes used is conforming, so we don't have to worry about the checking on concrete nodes. Termination is a consequence of the following properties of the expansion rules:

1. Each rule but the $\leqslant$-, $\leqslant_P$- or the **O**-rule strictly extends the completion forest, by extending node labels or adding nodes, while removing neither nodes nor elements from node.
2. New nodes are only generated by the $\exists$-, $\exists_P$-, $\geqslant$-rule or the $\geqslant_P$-rule as successors of a node x for concepts of the form $\exists R.C$, $\exists T_1, \cdots, T_n.P_n$, $\geqslant nS.C$ and $\geqslant mT_1, \cdots, T_n.P_n$ in $\mathcal{L}(x)$. For a node x, each of these concepts can trigger the generation of successors at most once—even though the node(s) generated was later removed by either the $\leqslant$-, $\leqslant_P$- or the **O**-rule. For the $\geqslant_P$-rule: If $T_1T_2 \ldots T_n$-successors $\langle t_{11}, \cdots, t_{1n}\rangle, \cdots, \langle t_{m1}, \cdots, t_{mn}\rangle$ were generated by an application of the $\geqslant_P$-rule for a concept $(\geqslant mT_1, \cdots, T_n.P_n)$, then $\langle t_{j1}, \cdots, t_{jn}\rangle \neq \langle t_{k1}, \cdots, t_{kn}\rangle$ holds for all $1 \leq i \leq n$ and $1 \leq j < k \leq m$. This implies there will always be m $T_1T_2 \ldots T_n$-successors $\langle t_{11}, \cdots, t_{1n}\rangle, \cdots, \langle t_{m1}, \cdots, t_{mn}\rangle$ of x with $P_n(i) \in \mathcal{L}(t_i)$ and $\langle t_{j1}, \cdots, t_{jn}\rangle \neq \langle t_{k1}, \cdots, t_{kn}\rangle$ holds for all $1 \leq i \leq n$ and $1 \leq j < k \leq m$, since the $\leqslant$-, **O**- and $\leqslant_P$-rule can never merge them, and, whenever an application of the $\leqslant_P$-rule sets some $\mathcal{L}(t_{ji})$ to $\emptyset$, then there will be some $T_1T_2 \ldots T_n$-successors $\langle t_{k1}, \cdots, t_{kn}\rangle$ of x with $P_n(i) \in \mathcal{L}(t_{ki})$ and $\langle t_{k1}, \cdots, t_{kn}\rangle$ "inherits" all inequalities from $\langle t_{j1}, \cdots, t_{jn}\rangle$. Hence the out-degree of the forest is bounded by $d \cdot n_{max}$.
3. Nodes are labelled with subsets of $cl(D) \cup \{\uparrow(R, \{o\}) \mid R \in \mathbf{R}_A^D \text{ and } \{o\} \in \mathcal{I}^D\}$, and the concrete value nodes are always leaves, so there are at most $2^{d+k\ell}$ different node labellings. Therefore, if a path p is of length at least $2^{d+k\ell}$, then, from the blocking condition above, there are two nodes x, y on p such that x is directly blocked by y. Hence paths are of length at most $2^{d+k\ell}$. □

Lemma 3. (Soundness) *If the expansion rules can be applied to a $\mathcal{SHOQ}(\mathbf{D_n})$-concept D in NNF and a role box $\mathcal{R}$ such that they yield a complete and clash-free completion forest, then D has a tableau w.r.t. $\mathcal{R}$.*

Proof: Let $\mathtt{F}$ be the complete and clash-free completion forest constructed by the tableaux algorithm for D. To cope with cycle, an individual in $\mathbf{S}$ corresponds to a *path* in $\mathtt{F}$. Due to qualifying number restrictions, we must distinguish different nodes that are blocked by the same node. We refer the readers to [10] for the definitions of path and related concepts. We can define a tableau $\mathcal{T} = (\mathbf{S}, \mathcal{L}, \mathcal{E}_A, \mathcal{E}_\mathbf{D})$ with: $\mathbf{S} = \mathtt{Paths(F)}$, $\mathcal{L}(p) = \mathcal{L}(\mathtt{Tail}(p))$, $\mathcal{E}_A(R_A) = \{\langle p, q\rangle \in \mathbf{S} \times \mathbf{S} \mid q = [p|(x, x')]$ *and* x' is an R_A-successor of $\mathtt{Tail}(p), \}$ $\mathcal{E}_\mathbf{D}(R_\mathbf{D}) = \{\langle p, value(t)\rangle \in \mathbf{S} \times \Delta_\mathbf{D} \mid t$ is an $R_\mathbf{D}$-successor of $\mathtt{Tail}(p)\}$.

We have to show that $\mathcal{T}$ satisfies (P14) to (P16) from Definition 5.

– (P14): Assume $\geqslant mT_1, \ldots, T_n.P_n \in \mathcal{L}(p)$. This implies that in $\mathtt{F}$ there exist m $T_1T_2 \ldots T_n$-successors $\langle t_{11}, \ldots, t_{1n}\rangle, \ldots, \langle t_{m1}, \ldots, t_{mn}\rangle$ of $\mathtt{Tail}(p)$, s.t. $\langle t_{j1}, \cdots, t_{jn}\rangle \neq \langle t_{k1}, \cdots, t_{kn}\rangle$ and $\{p, \langle T_1, \ldots, T_n\rangle, \langle t_{j1}, \ldots, t_{jn}\rangle, P_n\} \in DC^\mathtt{F}$ for all $1 \leq j < k \leq m$ (otherwise, $\geqslant_P$-rule was still applicable). We claim that, for each of these concrete nodes, according to the construction of $\mathcal{E}_\mathbf{D}$ above, we have $\langle p, value(t_{ji})\rangle \in \mathcal{E}_D(T_i)$ and $\langle value(t_{j1}), \ldots, value(t_{jn})\rangle \in P_n^\mathbf{D}$ for all $1 \leq i \leq n$ and $1 \leq j \leq m$. According to the satisfiability of $DC^\mathtt{F}$ and definition of $T_1T_2 \ldots T_n^\mathcal{T}(p, P_n)$, this implies $\sharp T_1T_2 \ldots T_n^\mathcal{T}(p, P_n) \geqslant m$.

– (P15): Assume (P15) doesn't hold. Hence there is some $p \in \mathbf{S}$ with $(\leqslant m T_1, \ldots, T_n.P_n) \in \mathcal{L}(p)$ and $\sharp T_1 T_2 \ldots T_n^{\mathcal{T}}(p, P_n) > m$. According to the definition of $T_1 T_2 \ldots T_n^{\mathcal{T}}(p, P_n)$, this implies that there exist $\langle t_{11}, \ldots, t_{1n}\rangle$, $\ldots$, $\langle t_{m+1,1}, \ldots, t_{m+1,n}\rangle$ such that $\langle p, value(t_{ji})\rangle \in \mathcal{E}_D(T_i)$, and $\{p, \langle T_1, \ldots, T_n\rangle, \langle t_{j1}, \ldots, t_{jn}\rangle, P_n\} \in DC^{\mathtt{F}}$, for all $1 \leq i \leq n$ and $1 \leq j < k \leq m+1$. Since the $m+1$ tuples of data values aren't equal to each other, so the $m+1$ tuples of concrete nodes aren't equal to each other. Thus $DCSuccessors^{\mathtt{F}}(x, T_1, \ldots, T_n, P_n) \geqslant m+1$, which fires a clash. Thus the assumption $\sharp T_1 T_2 \ldots T_n^{\mathcal{T}}\ (p, P_n) > m$ is false. So we have $\sharp T_1 T_2 \ldots T_n^{\mathcal{T}}(p, P_n) \leqslant m$.
– (P16): Assume $\{\leqslant m T_1, \cdots, T_n.P_n, \geqslant m T_1, \cdots, T_n.P_n\} \cap \mathcal{L}(p) \neq \emptyset$, $\langle p, t_i\rangle \in \mathcal{E}_{\mathbf{D}}(T_i), 1 \leq i \leq n$, thus $\langle t_1, \ldots, t_n\rangle$ is a $T_1 T_2 \ldots T_n$-successors of $\mathtt{Tail}(p)$. Let $value(t_i)$ be the value of t_i: (1) if $\{p, \langle T_1, \ldots, T_n\rangle, \langle t_1, \ldots, t_n\rangle, P_n\} \in DC^{\mathtt{F}}$, we have $DC^{\mathcal{T}}(p, T_1, \ldots, T_n, value(t_1), \ldots, value(t_n), P_n) = true$; (2) if $\{p, \langle T_1, \ldots, T_n\rangle, \langle t_1, \ldots, t_n\rangle, \neg P_n\} \in DC^{\mathtt{F}}$, we have $DC^{\mathcal{T}}(p, T_1, \ldots, T_n, value(t_1), \ldots, value(t_n), \neg P_n) = true$.

□

Lemma 4. (Completeness) *If a $\mathcal{SHOQ}(\mathbf{D_n})$-concept D in NNF has a tableau w.r.t. $\mathcal{R}$, then the expansion rules can be applied to D and $\mathcal{R}$ such that they yield a complete, clash-free completion forest.*

Proof: Let $\mathcal{T} = (\mathbf{S}, \mathcal{L}, \mathcal{E}_A, \mathcal{E}_{\mathbf{D}})$ be a tableau for D w.r.t. a role box $\mathcal{R}$. We use $\mathcal{T}$ to guide the application of the non-deterministic rules. We define a function π, mapping the nodes of the forest $\mathtt{F}$ to $\mathbf{S} \cup \Delta_{\mathbf{D}}$ such that $\mathcal{L}(x) \subseteq \mathcal{L}(\pi(x))$; $\langle \pi(x), \pi(y)\rangle \in \mathcal{E}_A$ if: 1. $\pi(y) \in \mathbf{S}$ and y is an R_A-successor of x, or 2. $\uparrow (R, \{o\}) \in \mathcal{L}(x)$ and $y = x_{\{o\}}$; $\langle \pi(x), \pi(y)\rangle \in \mathcal{E}_{\mathbf{D}}$ if $\pi(y) \in \Delta_{\mathbf{D}}$ and y is an $R_{\mathbf{D}}$-successor of x; abstract nodes $x \neq y$ implies $\pi(x) \neq \pi(y)$; while tuples of concrete nodes $\langle y_{j1}, \ldots, y_{jn}\rangle \neq \langle y_{k1}, \ldots, y_{kn}\rangle$ implies $\langle \pi(y_{j1}), \ldots, \pi(y_{jn})\rangle \neq \langle \pi(y_{k1}), \ldots, \pi(y_{kn})\rangle$. $(*)$

We only have to consider the various rules about number restriction on datatypes.

– The $\geqslant_P$-rule: If $\geqslant m T_1, \ldots, T_n.P_n \in \mathcal{L}(x)$, then $\geqslant m T_1, \ldots, T_n.P_n \in \mathcal{L}(\pi(x))$. Since $\mathcal{T}$ is a tableau, (P14) of Definition 5 implies that $\sharp T_1 T_2 \ldots T_n^{\mathcal{T}}(\pi(x), P_n) \geqslant m$. Hence there are m tuples of data values $\langle t_{11}, \ldots, t_{1n}\rangle, \ldots, \langle t_{m1}, \ldots, t_{mn}\rangle$, such that $\langle \pi(x), t_{ji}\rangle \in \mathcal{E}_{\mathbf{D}}$, $\langle t_{j1}, \ldots, t_{jn}\rangle \neq \langle t_{k1}, \ldots, t_{kn}\rangle$, and $DC^{\mathcal{T}}(\pi(x), T_1, \ldots, T_n, t_{j1}, \ldots, t_{jn}, P_n) = true$, for $1 \leq i \leq n$ and $1 \leq j < k \leq m$. The $\geqslant_P$-rule generates m new $T_1 T_2 \ldots T_n$-successors $\langle y_{11}, \ldots, y_{1n}\rangle, \ldots, \langle y_{m1}, \ldots, y_{mn}\rangle$. By setting $\pi' := \pi[y_{ji} \mapsto t_{ji}](1 \leq i \leq n, 1 \leq j < k \leq m)$, one obtains a function π' that satisfies $(*)$ for the modified forest.
– The $\leqslant_P$-rule: If $\leqslant m T_1, \ldots, T_n.P_n \in \mathcal{L}(x)$, then $\leqslant m T_1, \ldots, T_n.P_n \in \mathcal{L}(\pi(x))$. Since $\mathcal{T}$ is a tableau, (P15) of Definition 5 implies $\sharp T_1 T_2 \ldots T_n^{\mathcal{T}}(\pi(x), P_n) \leqslant m$. If the $\leqslant_P$-rule is applicable, we have $\sharp DCSuccessors^{\mathtt{F}}(x, T_1, \ldots, T_n, P_n) > m$, which implies that there are at least $m+1$ $T_1 T_2 \ldots T_n$-successors $\langle y_{11}, \ldots, y_{1n}\rangle, \ldots, \langle y_{m+1,1}, \ldots, y_{m+1,n}\rangle$ such that $\{x, \langle T_1, \ldots, T_n\rangle, \langle y_{j1}, \ldots, y_{jn}\rangle, P_n\} \in DC^{\mathtt{F}}$, for $1 \leq j \leq m+1$. Thus, there must be two $\langle y_{j1}, \ldots, y_{jn}\rangle$ and $\langle y_{k1}, \ldots, y_{kn}\rangle$ among the $m+1$ $T_1 T_2 \ldots T_n$-successors such that $\langle \pi(y_{j1}), \ldots, \pi(y_{jn})\rangle = \langle \pi(y_{k1}, \ldots, \pi(y_{kn})\rangle$ (otherwise $\sharp T_1 T_2 \ldots T_n^{\mathcal{T}}(\pi(x), P_n) > m$ would hold). This implies $\langle y_{j1}, \ldots, y_{jn}\rangle \neq \langle y_{k1}, \ldots, y_{kn}\rangle$ cannot hold because of $(*)$. Hence the $\leqslant_P$-rule can be applied without violating $(*)$.
– The $choose_P$-rule: If $\{\leqslant m T_1, \cdots, T_n.P_n, \geqslant m T_1, \cdots, T_n.P_n\} \cap \mathcal{L}(x) \neq \emptyset$, we have $\{\leqslant m T_1, \cdots, T_n.P_n, \geqslant m T_1, \cdots, T_n.P_n\} \cap \mathcal{L}(\pi(x)) \neq \emptyset$, and if there are $T_1 T_2 \ldots T_n$-successors $\langle y_1, \ldots, y_n\rangle$ of x, then $\langle \pi(x), \pi(y_i)\rangle \in \mathcal{E}_{\mathbf{D}}, 1 \leq i \leq n$, due to $(*)$. Since $\mathcal{T}$ is a tableau, (P16) of Definition 5 implies either $DC^{\mathcal{T}}(\pi(x), T_1, \ldots, T_n, \pi(y_1), \ldots, \pi(y_n), P_n) = true$, or $DC^{\mathcal{T}}(\pi(x), T_1, \ldots, T_n, \pi(y_1), \ldots, \pi(y_n), \neg P_n) = true$. Hence the $choose_P$-rule can accordingly either set $DC^{\mathtt{F}} \longrightarrow DC^{\mathtt{F}} \cup \{x, \langle T_1, \ldots T_n\rangle, \langle y_1, \ldots, y_n\rangle, P_n\}$, or set $DC^{\mathtt{F}} \longrightarrow DC^{\mathtt{F}} \cup \{x, \langle T_1, \ldots, T_n\rangle, \langle y_1, \ldots, y_n\rangle, \neg P_n\}$.

Whenever a rule is applicable to $\mathtt{F}$, it can be applied in a way that maintains $(*)$, and, from Lemma 2, we have that any sequence of rule applications must terminate. Since $(*)$ holds, any forest generated by these rule-applications must be clash-free. This can be seen from the condition described in [8] plus the following:

– If $\mathtt{F}$ does not satisfy $DC^{\mathtt{F}}$, there must be some concrete nodes from which no values mapping satisfies all the relevant predicates, and therefore there can be no values satisfying all of properties (P12) to (P16).

– $\mathtt{F}$ cannot contain a node x with $\leqslant m T_1, \ldots, T_n.P_n \in \mathcal{L}(x)$, and $m+1$ $T_1 T_2 \ldots T_n$-successors $\langle t_{11}, \ldots, t_{1n}\rangle, \ldots, \langle t_{m+1,1}, \ldots, t_{m+1,n}\rangle$ of x with $\langle t_{j1}, \ldots, t_{jn}\rangle \neq \langle t_{k1}, \ldots, t_{kn}\rangle$ and $\{x, \langle T_1, \ldots, T_n\rangle, \langle t_{j1}, \ldots, t_{jn}\rangle, P_n\} \in DC^{\mathtt{F}}$, for all $1 \leq i \leq n, 1 \leq j < k \leq m+1$, because $\leqslant m T_1, \ldots, T_n.P_n \in \mathcal{L}(\pi(x))$, and, since $\langle t_{j1}, \ldots, t_{jn}\rangle \neq \langle t_{k1}, \ldots, t_{kn}\rangle$ implies $\langle \pi(t_{j1}), \ldots, \pi(t_{jn})\rangle \neq \langle \pi(t_{k1}), \ldots, \pi(t_{kn})\rangle$, $\sharp T_1 T_2 \ldots T_n^{\mathcal{T}}(\pi(x), P_n) > m$ would hold which contradicts (P15) of Definition 5. □

As an immediate consequence of Lemmas 2,4,5 and 6, the completion algorithm always terminates, and answers with "D is satisfiable w.r.t. $\mathcal{R}$" iff. D has a tableau T. Next, subsumption can be reduced to (un)satisfiability. Finally, $\mathcal{SHOQ}(\mathbf{D_n})$ can internalise general concept inclusion axions [6]. However, in the presence of nominals, we must also add $\exists O.o_1 \cap \cdots \cap \exists O.o_l$ to the concept internalising the general concept inclusion axioms to make sure that the universal role O indeed reaches all nominals O_i occurring in the input concept and terminology. Thus, we can decide these inference problems also w.r.t. terminologies.

Theorem 7 The tableau algorithm presented in Definition 6 is a decision procedure for satisfiability and subsumption of $\mathcal{SHOQ}(\mathbf{D_n})$-concepts w.r.t. terminologies.

6 Discussion

As we have seen, unary datatype predicates are usually not enough, while n-ary datatype predicates as well as qualified number restrictions with n-ary datatype predicates are often necessary when modelling the "concrete properties" of real world entities in semantic Web ontologies and applications. Therefore, we have extended $\mathcal{SHOQ}(\mathbf{D})$ with n-ary datatype predicates and qualified number restrictions with n-ary predicates to give the $\mathcal{SHOQ}(\mathbf{D_n})$ DL. We have shown that the decision procedure for concept satisfiability and subsumption is still decidable in $\mathcal{SHOQ}(\mathbf{D_n})$. An implementation based on the FaCT system is planned, and will be used to test empirical performance.

With its support for nominals, n-ary datatype predicates and qualified number restrictions with n-ary datatype predicates, $\mathcal{SHOQ}(\mathbf{D_n})$ is well suited to provide reasoning support for ontology languages in general, and semantic Web ontology languages in particular. Therefore, we think that future versions of Web ontology languages, e.g. DAML+OIL and OWL, can include n-ary datatype predicates and qualified number restrictions with n-ary datatype predicates. As future work, it is interesting to study how the extended datatype support introduced in the paper can help to express various business rules required in semantic Web ontologies and applications. Secondly, it is also important to extend current optimisation techniques to cope with nominals used in the $\mathcal{SHOQ}(\mathbf{D_n})$ DL.

Acknowledgements

We would like to thank Ulrike Sattler, since the work presented here extends the original work on $\mathcal{SHOQ}(\mathbf{D})$. Thanks are also due to Carsten Lutz and Volker Haarslev for their helpful discussion on concrete datatypes.

Bibliography

[1] F. Baader and Philipp Hanschke. A Scheme for Integrating Concrete Domains into Concept Languages. In *IJCAI-91*, pages 452–457, 1991.

[2] Tim Berners-lee. Semantic Web Road Map. W3C Design Issues. `URLhttp://www.w3.org/DesignIssues/Semantic.html`, Oct. 1998.

[3] J. Broekstra, M. Klein, S. Decker, D. Fensel, F. can Harmelen, and I. Horrocks. Enabling knowledge representation on the Web by extending RDF Schema, Nov. 2000.

[4] Benjamin N. Grosof and Terrence C. Poon. Representing Agent Contracts with Exceptions using XML Rules, Ontologies, and Process Descriptions. In *Proceedings of International Workshop on Rule Markup Languages for Business Rules on the Semantic Web*, Jun 2002.

[5] Volker Haarslev, Ralf Mller, and Michael Wessel. The Description Logic ALCNHR+ Extended with Concrete Domains: A Practically Motivated Approach. In *Proceedings of International Joint Conference on Automated Reasoning, IJCAR'2001, R. Gor, A. Leitsch, T. Nipkow (Eds.),Siena, Italy, Springer-Verlag, Berlin.*, pages 29–44, Jun. 2001.

[6] I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Expressive Description Logics. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes in Artificial Intelligence, pages 161–180. Springer-Verlag, 1999.

[7] Ian Horrocks and Peter F. Patel-Schneider. The Generation of DAML+OIL. Aug. 2001. The 2001 International Workshop on Description Logics.

[8] Ian Horrocks and U. Sattler. Ontology Reasoning for the Semantic Web. In *In B. Nebel, editor, Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI'01), Morgan Kaufmann*, pages 199–204, 2001.

[9] C. Lutz. NExpTime-complete Description Logics with Concrete Domains . LuFG Theoretical Computer Science, RWTH Aachen, Germany, 2000.

[10] Jeff Z. Pan. Web Ontology Reasoning in the SHOQ(Dn) Description Logic. In *Proceedings of the Methods for Modalities 2 (M4M-2)*, Nov 2001. ILLC, University of Amsterdam, URL `http://www.cs.man.ac.uk/~panz/Zhilin/download/Paper/Pan-shoqdn-2001.pdf`.

[11] Jeff Z. Pan and Ian Horrocks. Metamodeling Architecture of Web Ontology Languages. In *Proceeding of the Semantic Web Working Symposium (SWWS)*, July 2001. URL `http://www.cs.man.ac.uk/~panz/Zhilin/download/Paper/Pan-Horrocks-rdfsfa-2001.pdf`.

[12] Frank van Harmelen, Peter F. Patel-Schneider, and Ian Horrocks. A Model-Theortis Semantics of DAML+OIL(March 2001). Mar. 2001.