A Tableaux Decision Procedure for $\mathcal{SHOIQ}$

Ian Horrocks and **Ulrike Sattler**

Department of Computer Science
University of Manchester, UK
{horrocks|sattler}@cs.man.ac.uk

Abstract

OWL DL, a new W3C ontology language recommendation, is based on the expressive description logic $\mathcal{SHOIN}$. Although the ontology consistency problem for $\mathcal{SHOIN}$ is known to be decidable, up to now there has been no known "practical" decision procedure, i.e., a goal directed procedure that is likely to perform well with realistic ontology derived problems. We present such a decision procedure (for $\mathcal{SHOIQ}$, a slightly more expressive logic than $\mathcal{SHOIN}$), extending the well known algorithm for $\mathcal{SHIQ}$, which is the basis for several highly successful implementations.

A short version of this report appears in the *Proceedings of Nineteenth International Joint Conference on Artificial Intelligence* (IJCAI 2005).

Contents

1 Introduction

Description Logics (DLs) are a family of logic based knowledge representation formalisms. Although they have a range of applications (e.g., configuration [14], and information integration [4]), they are perhaps best known as the basis for widely used ontology languages such as OIL, DAML+OIL and OWL [10], the last of which is now a World Wide Web Consortium (W3C) recommendation [2].

The OWL specification describes three language "species", OWL Lite, OWL DL and OWL Full, two of which (OWL Lite and OWL DL) are based on expressive description logics.[1] The decision to base these languages on DLs was motivated by a requirement that key inference problems (such as ontology consistency) be decidable, and hence that it should be possible to provide reasoning services to support ontology design and deployment [10].

OWL Lite and OWL DL are based on the DLs $\mathcal{SHIF}$ and $\mathcal{SHOIN}$ respectively—in fact OWL Lite is just a syntactic subset of OWL DL [10].[2] Although the ontology consistency problem for $\mathcal{SHOIN}$ is known to be decidable,[3] to the best of our knowledge no "practical" decision procedure is known for it, i.e., no goal directed procedure that is likely to perform well with realistic ontology derived problems [20, 11]. In this report we present such a decision procedure for $\mathcal{SHOIQ}$, i.e., $\mathcal{SHOIN}$ extended with *qualified* number restrictions [7]. The algorithm extends the well-known tableaux algorithm for $\mathcal{SHIQ}$ [12], which is the basis for several highly successful implementations [9, 5, 17].

The $\mathcal{O}$ in $\mathcal{SHOIQ}$ denotes *nominals*, i.e., classes with a singleton extension. Nominals are a prominent feature of hybrid logics [3], and can also be viewed as a powerful generalisation of *ABox individuals* [18, 11]. They occur naturally in ontologies, e.g., when describing a class such as EUCountries by enumerating its members, i.e., $\{\mathsf{Austria}, \ldots, \mathsf{UnitedKingdom}\}$ (such an enumeration is equivalent to a disjunction of nominals). This allows applications to infer, e.g., that persons who only visit EUCountries can visit at most 15 countries.

One reason why DLs (and propositional modal and dynamic logics) enjoy good computational properties, such as being robustly decidable, is that they have some form of tree model property [22], i.e., if an ontology is consistent, then it has a (form of) tree model. This feature is crucial in the design of tableaux

[1] OWL Full uses the same language vocabulary as OWL DL, but does not restrict its use to "well formed formulae".

[2] OWL also includes *datatypes*, a simple form of *concrete domain* [1]. These can, however, be treated exactly as in $\mathcal{SHOQ}(\mathbf{D})$/$\mathcal{SHOQ}(\mathbf{D}_n)$ [11, 16], so we will not complicate our presentation by considering them here.

[3] This is an immediate consequence of a reduction of DLs with transitive roles to DLs without such roles [20] and the fact that applying this reduction to $\mathcal{SHOIN}$ yields a fragment of the two variable fragment of first order logic with counting quantifiers [15].

algorithms, allowing them to search only for tree like models. More precisely, DL tableaux algorithms decide consistency of an ontology by trying to construct an abstraction of a model for it, a so-called "completion graph". For logics with the tree model property, we can restrict our search/construction to tree-shaped completion graphs.

Tableaux algorithms for expressive DLs employ a cycle detection technique called *blocking* to ensure termination. This is of special interest for $\mathcal{SHIQ}$, where the interaction between inverse roles and number restrictions results in the loss of the *finite model property*, i.e., there are consistent ontologies that only admit infinite models. On such an input, the $\mathcal{SHIQ}$ tableaux algorithm generates a finite tree-shaped completion graph that can be *unravelled* into an infinite tree model, and where a node in the completion graph may stand for infinitely many elements of the model. Even when the language includes nominals, but *excludes* one of number restrictions or inverse roles [11, 6], or if nominals are restricted to ABox individuals [13], we can work on forest-shaped completion graphs, with each nominal (individual) being the root of a tree like section; this causes no inherent difficulty as the size of the non-tree part of the graph is restricted by the number of individuals/nominals in the input.

The difficulty in extending the $\mathcal{SHOQ}$ or $\mathcal{SHIQ}$ algorithms to $\mathcal{SHOIQ}$ is due to the interaction between nominals, number restrictions, and inverse roles, which leads to the *almost* complete loss of the tree model property, and causes the complexity of the ontology consistency problem to jump from ExpTime to NExpTime [19]. To see this, consider an ontology containing the following two axioms that use a nominal o to impose an upper bound of n on the number of instances of the concept F:

$$\begin{aligned} \top &\dot{\sqsubseteq} \exists u^{-}.o \\ o &\dot{\sqsubseteq} (\leqslant n\, u.F) \end{aligned}$$

The first statement requires that, in a model of this ontology, every element has an incoming u-edge from o; the second statement restricts the number of u-edges going from o to instances of F to at most n. In this case, we might need to consider arbitrarily complex relational structures amongst instances of F, and thus cannot restrict our attention to completion trees or forests. Let us assume that our ontology also forces the existence of an infinite number of instances of another concept, say N, which requires the above mentioned "block and unravel" technique. The consistency of the whole ontology then crucially depends on the relations enforced between instances of F and N, and whether the unravelling of the N-part violates atmost number restrictions that instances of F must satisfy. Summing up, a tableaux algorithm for $\mathcal{SHOIQ}$ needs to be able to handle both arbitrarily complex relational structures and finite tree structures representing in-

finite trees, and to make sure that all constraints are satisfied (especially number restrictions on relations between these two parts), while still guaranteeing termination.

Two key intuitions have allowed us to devise a tableaux algorithm that meets all of these requirements. The first intuition is that, when extending a $\mathcal{SHOIQ}$ completion graph, we can distinguish those nodes that may be arbitrarily interconnected (so-called *nominal nodes*) from those nodes that still form a tree structure (*so-called blockable nodes*). Fixing a (double exponential) upper bound on the number of nominal nodes is crucial to proving termination; it is not, however, enough to guarantee termination, as we may repeatedly create and merge nominal nodes (a so-called "yo-yo").

The second intuition is that the yo-yo problem can be overcome by "guessing" the *exact* number of new nominal nodes resulting from interactions between existing nominal nodes, inverse roles and number restrictions. This guessing is implemented by a new expansion rule, the *NN*-rule, which, when applied to a relevant $(\leqslant nR.C)$ concept, generates (non-deterministically) between 1 and n new nominal nodes, all of which are pairwise disjoint. This prevents the repeated yo-yo construction, and termination is now guaranteed by the upper bound on the number of nominal nodes and the use of standard blocking techniques for the blockable nodes. The non-determinism introduced by this rule could clearly be problematical for large values of n, but large values in number restrictions are already known to be problematical for $\mathcal{SHIQ}$. Moreover, the rule has excellent "pay as you go" characteristics: in case number restrictions are functional (i.e., where n is 1),[4] the new rule becomes deterministic; in case there are no interactions between number restrictions, inverse roles and nominals, the rule will never be applied; in case there are no nominals, the new algorithm will behave like the algorithm for $\mathcal{SHIQ}$; and in case there are no inverse roles the new algorithm will behave like the algorithm for $\mathcal{SHOQ}$.

2 Preliminaries

In this section, we introduce the DL $\mathcal{SHOIQ}$. This includes the definition of syntax, semantics, and inference problems. We start with $\mathcal{SHOIQ}$-roles, then introduce some abbreviations, and finally define $\mathcal{SHOIQ}$-concepts.

Definition 1 Let $\mathbf{R}$ be a set of *role names* with both transitive and normal role names $\mathbf{R}_+ \cup \mathbf{R}_\mathsf{P} = \mathbf{R}$, where $\mathbf{R}_\mathsf{P} \cap \mathbf{R}_+ = \emptyset$. The set of $\mathcal{SHOIQ}$*-roles* (or *roles*

[4] A feature of many realistic ontologies; see, e.g., the DAML ontology library at `http://www.daml.org/ontologies/`

for short) is $\mathbf{R} \cup \{R^- \mid R \in \mathbf{R}\}$. A *role inclusion axiom* is of the form $R \sqsubseteq S$, for two roles R and S. A *role hierarchy* is a finite set of role inclusion axioms.

An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty set $\Delta^{\mathcal{I}}$, the *domain* of $\mathcal{I}$, and a function $\cdot^{\mathcal{I}}$ which maps every role to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ such that, for $P \in \mathbf{R}$ and $R \in \mathbf{R}_+$,

$$\langle x, y\rangle \in P^{\mathcal{I}} \text{ iff } \langle y, x\rangle \in P^{-\mathcal{I}},$$
$$\text{and if } \langle x, y\rangle \in R^{\mathcal{I}} \text{ and } \langle y, z\rangle \in R^{\mathcal{I}}, \text{ then } \langle x, z\rangle \in R^{\mathcal{I}}.$$

An interpretation $\mathcal{I}$ *satisfies a role hierarchy* $\mathcal{R}$ if $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$ for each $R \sqsubseteq S \in \mathcal{R}$; such an interpretation is called a *model* of $\mathcal{R}$.

We introduce some notation to make the following considerations easier.

1. The inverse relation on roles is symmetric, and to avoid considering roles such as R^{--}, we define a function Inv which returns the inverse of a role:

$$\mathsf{Inv}(R) := \begin{cases} R^- & \text{if } R \text{ is a role name,} \\ S & \text{if } R = S^- \text{ for a role name } S. \end{cases}$$

2. Since set inclusion is transitive and $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$ implies $\mathsf{Inv}(R)^{\mathcal{I}} \subseteq \mathsf{Inv}(S)^{\mathcal{I}}$, for a role hierarchy $\mathcal{R}$, we introduce $\sqsubseteq^*_{\mathcal{R}}$ as the transitive-reflexive closure of $\sqsubseteq$ on $\mathcal{R} \cup \{\mathsf{Inv}(R) \sqsubseteq \mathsf{Inv}(S) \mid R \sqsubseteq S \in \mathcal{R}\}$. We use $R \equiv_{\mathcal{R}} S$ as an abbreviation for $R \sqsubseteq^*_{\mathcal{R}} S$ and $S \sqsubseteq^*_{\mathcal{R}} R$.

3. Obviously, a role R is transitive if and only if its inverse $\mathsf{Inv}(R)$ is transitive. However, in cyclic cases such as $R \equiv_{\mathcal{R}} S$, S is transitive if R or $\mathsf{Inv}(R)$ is a transitive role name. In order to avoid these case distinctions, the function Trans returns true if R is a transitive role—regardless whether it is a role name, the inverse of a role name, or equivalent to a transitive role name (or its inverse): $\mathsf{Trans}(S, \mathcal{R}) := \mathrm{true}$ if, for some P with $P \equiv_{\mathcal{R}} S$, $P \in \mathbf{R}_+$ or $\mathsf{Inv}(P) \in \mathbf{R}_+$; $\mathsf{Trans}(S, \mathcal{R}) := \mathrm{false}$ otherwise.

4. A role R is called *simple* w.r.t. $\mathcal{R}$ if $\mathsf{Trans}(S, \mathcal{R}) = \mathrm{false}$ for all $S \sqsubseteq^*_{\mathcal{R}} R$.

5. In the following, if $\mathcal{R}$ is clear from the context, then we may abuse our notation and use $\sqsubseteq^*$ and $\mathsf{Trans}(S)$ instead of $\sqsubseteq^*_{\mathcal{R}}$ and $\mathsf{Trans}(S, \mathcal{R})$, and we say that "$S$ is a simple role" instead of "S is simple w.r.t. $\mathcal{R}$".

Definition 2 Let N_C be a set of *concept names* with a subset $N_I \subseteq N_C$ of *nominals*, and let $\mathcal{R}$ be a role hierarchy. The set of $\mathcal{SHOIQ}$*-concepts* (or *concepts* for short) is the smallest set such that

1. every concept name $C \in N_C$ is a concept,

2. if C and D are concepts and R is a role, then $(C \sqcap D)$, $(C \sqcup D)$, $(\neg C)$, $(\forall R.C)$, and $(\exists R.C)$ are also concepts (the last two are called universal and existential restrictions, resp.), and

3. if C is a concept, R is a simple role[5] and $n \in \mathbb{N}$, then $(\leqslant nR.C)$ and $(\geqslant nR.C)$ are also concepts (called atmost and atleast number restrictions).

The interpretation function $\cdot^{\mathcal{I}}$ of an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ maps, additionally, every concept to a subset of $\Delta^{\mathcal{I}}$ such that

$$
\begin{aligned}
(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}, \qquad (C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}, \qquad \neg C^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}, \\
\sharp o^{\mathcal{I}} = 1 \qquad \text{for all } o \in N_I, \\
(\exists R.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} \mid \text{There is a } y \in \Delta^{\mathcal{I}} \text{ with } \langle x, y\rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}, \\
(\forall R.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} \mid \text{For all } y \in \Delta^{\mathcal{I}}, \text{ if } \langle x, y\rangle \in R^{\mathcal{I}}, \text{then } y \in C^{\mathcal{I}}\}, \\
(\leqslant nR.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} \mid \sharp R^{\mathcal{I}}(x, C) \leqslant n\}, \\
(\geqslant nR.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} \mid \sharp R^{\mathcal{I}}(x, C) \geqslant n\},
\end{aligned}
$$

where, for a set M, we denote the cardinality of M by $\sharp M$ and $R^{\mathcal{I}}(x, C)$ is defined as $\{y \mid \langle x, y\rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}$.

For C and D (possibly complex) concepts, $C \dot{\sqsubseteq} D$ is called a *general concept inclusion* (GCI), and a finite set of GCIs is called a *TBox*.

An interpretation $\mathcal{I}$ *satisfies* a GCI $C \dot{\sqsubseteq} D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, and $\mathcal{I}$ satisfies a TBox $\mathcal{T}$ if $\mathcal{I}$ satisfies each GCI in $\mathcal{T}$; such an interpretation is called a *model of* $\mathcal{T}$.

A concept C is called *satisfiable with respect to a role hierarchy* $\mathcal{R}$ and a TBox $\mathcal{T}$ if there is a model $\mathcal{I}$ of $\mathcal{R}$ and $\mathcal{T}$ with $C^{\mathcal{I}} \neq \emptyset$. Such an interpretation is called a *model of* C w.r.t. $\mathcal{R}$ and $\mathcal{T}$. A concept D *subsumes* a concept C w.r.t. $\mathcal{R}$ and $\mathcal{T}$ (written $C \sqsubseteq_{\mathcal{R},\mathcal{T}} D$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds in every model $\mathcal{I}$ of $\mathcal{R}$ and $\mathcal{T}$. Two concepts C, D are *equivalent* w.r.t. $\mathcal{R}$ and $\mathcal{T}$ (written $C \equiv_{\mathcal{R},\mathcal{T}} D$) if they are mutually subsuming w.r.t. $\mathcal{R}$ and $\mathcal{T}$.

As usual, subsumption and satisfiability can be reduced to each other. Like in $\mathcal{SHIQ}$, in $\mathcal{SHOIQ}$, we can reduce reasoning w.r.t. *general TBoxes* and role hierarchies to reasoning w.r.t. role hierarchies only: we can use an "approximation" of a universal role U to *internalise* a TBox [12]. The only difference for $\mathcal{SHOIQ}$ is that, in the presence of nominals, we also conjoin $\exists U.o_1 \sqcap \ldots \sqcap \exists U.o_\ell$ to the concept internalising the TBox to make sure that the role U indeed reaches all nominals o_i occurring in the input. More precisely, a $\mathcal{SHOIQ}$ concept D is satisfiable w.r.t. $\mathcal{T}$ and $\mathcal{R}$ iff

$$D \sqcap C_{\mathcal{T}} \sqcap \forall U.C_{\mathcal{T}} \sqcap \exists U.o_1 \sqcap \ldots \sqcap \exists U.o_\ell$$

[5]Restricting number restrictions to simple roles is required in order to yield a decidable logic (see [12]).

is satisfiable w.r.t. $\mathcal{R} \cup \{U^- \sqsubseteq U, \mathsf{Trans}(U)\} \cup \{R \sqsubseteq U \mid R \text{ occurs in } \mathcal{T}, \mathcal{R}, \text{ or } D\}$, where $o_1, \ldots, o_\ell$ are all nominals occurring in D or $\mathcal{T}$ and $C_\mathcal{T} = \bigsqcap_{C_1 \dot{\sqsubseteq} C_2 \in \mathcal{T}} \neg C_1 \sqcup C_2$.

As a consequence, in the remainder of this paper, we restrict our attention without loss of generality to the satisfiability of $\mathcal{SHOIQ}$ concepts w.r.t. a role hierarchy.

Finally, we did not choose to make a *unique name assumption*, i.e., two nominals might refer to the same individual. However, the inference algorithm presented below can easily be adapted to the unique name case by a suitable initialisation of the inequality relation $\neq$.

3 A Tableau for $\mathcal{SHOIQ}$

For ease of presentation, as usual, we assume all concepts to be in *negation normal form* (NNF). Each concept can be transformed into an equivalent one in NNF by pushing negation inwards, making use of de Morgan's laws and the duality between existential and universal restrictions, and between atmost and atleast number restrictions, [13]. For a concept C, we use $\dot{\neg} C$ to denote the NNF of $\neg C$, and we use $\mathsf{sub}(C)$ to denote the set of all subconcepts of C (including C). As usual, for a concept D and a role hierarchy $\mathcal{R}$, we define the set of "relevant sub-concepts" $\mathsf{cl}(D, R)$ as follows:

$$\begin{aligned}\mathsf{cl}(D, \mathcal{R}) := \; & \mathsf{sub}(D) \cup \{\dot{\neg} C \mid C \in \mathsf{sub}(D)\} \cup \\ & \{\forall S.E \mid \forall R.E \in \mathsf{sub}(D) \text{ or } \dot{\neg}\forall R.E \in \mathsf{sub}(D) \text{ and } S \text{ occurs in } \mathcal{R} \text{ or } D\}\end{aligned}$$

When $\mathcal{R}$ is clear from the context, we use $\mathsf{cl}(D)$ instead of $\mathsf{cl}(D, \mathcal{R})$.

Definition 3 If D is a $\mathcal{SHOIQ}$-concept in NNF, $\mathcal{R}$ a role hierarchy, and $\mathbf{R}_D$ is the set of roles occurring in D or $\mathcal{R}$, together with their inverses, a tableau T for D w.r.t. $\mathcal{R}$ is defined to be a triple $(\mathbf{S}, \mathcal{L}, \mathcal{E})$ 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 $\mathsf{cl}(D)$, $\mathcal{E} : \mathbf{R}_D \to 2^{\mathbf{S} \times \mathbf{S}}$ maps each role in $\mathbf{R}_D$ to a set of pairs of individuals, and there is some individual $s \in \mathbf{S}$ such that $D \in \mathcal{L}(s)$. For all $s, t \in \mathbf{S}$, $C, C_1, C_2 \in \mathsf{cl}(D)$, $R, S \in \mathbf{R}_D$, and

$$S^T(s, C) := \{t \in \mathbf{S} \mid \langle s, t \rangle \in \mathcal{E}(S) \text{ and } C \in \mathcal{L}(t)\},$$

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 $\forall R.C \in \mathcal{L}(s)$ and $\langle s,t \rangle \in \mathcal{E}(R)$, then $C \in \mathcal{L}(t)$,

(P5) if $\exists R.C \in \mathcal{L}(s)$, then there is some $t \in \mathbf{S}$ such that $\langle s,t \rangle \in \mathcal{E}(R)$ and $C \in \mathcal{L}(t)$,

(P6) if $\forall S.C \in \mathcal{L}(s)$ and $\langle s,t \rangle \in \mathcal{E}(R)$ for some $R \sqsubseteq\!\!\!^{*} S$ with $\mathsf{Trans}(R)$, then $\forall R.C \in \mathcal{L}(t)$,

(P7) if $(\geqslant nS.C) \in \mathcal{L}(s)$, then $\sharp S^T(s,C) \geqslant n$,

(P8) if $(\leqslant nS.C) \in \mathcal{L}(s)$, then $\sharp S^T(s,C) \leqslant n$, and

(P9) if $(\leqslant nS.C) \in \mathcal{L}(s)$ and $\langle s,t \rangle \in \mathcal{E}(S)$, then $\{C, \dot{\neg}C\} \cap \mathcal{L}(t) \neq \emptyset$,

(P10) if $\langle s,t \rangle \in \mathcal{E}(R)$ and $R \sqsubseteq\!\!\!^{*} S$, then $\langle s,t \rangle \in \mathcal{E}(S)$,

(P11) $\langle s,t \rangle \in \mathcal{E}(R)$ iff $\langle t,s \rangle \in \mathcal{E}(\mathsf{Inv}(R))$, and

(P12) if $o \in \mathcal{L}(s) \cap \mathcal{L}(t)$ for some $o \in N_I$, then $s = t$.

Lemma 4 A $\mathcal{SHOIQ}$-concept D in NNF is satisfiable w.r.t. a role hierarchy $\mathcal{R}$ iff D has a tableau w.r.t. $\mathcal{R}$.

Proof (sketch): Is analogous to the proof found in [12]. Roughly speaking, we construct a model $\mathcal{I}$ from a tableau by taking $\mathbf{S}$ as its interpretation domain and adding the missing role-successorships for transitive roles. Then, by induction on the structure of formulae, we prove that, if $C \in \mathcal{L}(s)$, then $s \in C^{\mathcal{I}}$. (P12) ensures that nominals are indeed interpreted as singletons.

For the converse, we can easily transform any model into a tableau. □

4 A tableau algorithm for $\mathcal{SHOIQ}$

From Lemma 4, an algorithm which constructs a tableau for a $\mathcal{SHOIQ}$-concept D can be used as a decision procedure for the satisfiability of D with respect to a role hierarchy $\mathcal{R}$. Such an algorithm will now be described in detail.

We first define and comment on the underlying data structure and corresponding operations. Next, we provide an example of the algorithm's behaviour, and explain the techniques we have chosen to design a *terminating, sound, and complete* algorithm. Finally, we prove that our algorithm indeed is terminating, sound, and complete.

4.1 Definition of the algorithm

Definition 5 Let $\mathcal{R}$ be a role hierarchy and D a $\mathcal{SHOIQ}$-concept in NNF. A *completion graph* for D with respect to $\mathcal{R}$ is a directed graph $\mathbf{G} = (V, E, \mathcal{L}, \neq)$ where each node $x \in V$ is labelled with a set

$$\mathcal{L}(x) \subseteq \mathsf{cl}(D) \cup N_I \cup \{(\leqslant m R.C) \mid (\leqslant n R.C) \in \mathsf{cl}(D) \text{ and } m \leq n\}$$

and each edge $\langle x, y\rangle \in E$ is labelled with a set of role names $\mathcal{L}(\langle x, y\rangle)$ containing (possibly inverse) roles occurring in D or $\mathcal{R}$. Additionally, we keep track of inequalities between nodes of the graph with a symmetric binary relation $\neq$ between the nodes of $\mathbf{G}$.

In the following, we often use $R \in \mathcal{L}(\langle x, y\rangle)$ as an abbreviation for $\langle x, y\rangle \in E$ and $R \in \mathcal{L}(\langle x, y\rangle)$.

If $\langle x, y\rangle \in E$, then y is called a *successor* of x and x is called a *predecessor* of y. *Ancestor* is the transitive closure of predecessor, and *descendant* is the transitive closure of successor. A node y is called an R-successor of a node x if, for some R' with $R' \sqsubseteq^{*} R$, $R' \in \mathcal{L}(\langle x, y\rangle)$. A node y is called a *neighbour* (R-*neighbour*) of a node x if y is a successor (R-successor) of x or if x is a successor ($\mathsf{Inv}(R)$-successor) of y.

For a role S and a node x in $\mathbf{G}$, we define the set of x's S-neighbours with C in their label, $S^{\mathbf{G}}(x, C)$, as follows:

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

$\mathbf{G}$ is said to contain a *clash* if

1. for some concept name $A \in N_C$ and node x of $\mathbf{G}$, $\{A, \neg A\} \subseteq \mathcal{L}(x)$, *or*
2. for some role S and node x of $\mathbf{G}$, $(\leqslant n S.C) \in \mathcal{L}(x)$ and there are $n+1$ S-neighbours $y_0, \ldots, y_n$ 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. for some $o \in N_I$, there are nodes $x \neq y$ with $o \in \mathcal{L}(x) \cap \mathcal{L}(y)$.

If $o_1, \ldots, o_\ell$ are all the nominals occurring in D, then the tableau algorithm starts with the completion graph $\mathbf{G} = (\{r_0, r_1 \ldots, r_\ell\}, \emptyset, \mathcal{L}, \emptyset)$ with $\mathcal{L}(r_0) = \{D\}$ and $\mathcal{L}(r_i) = \{o_i\}$ for $1 \leq i \leq \ell$. $\mathbf{G}$ is then expanded by repeatedly applying the expansion rules given in Figures 1 and 2, stopping if a clash occurs.

Before describing the tableau algorithm in more detail, we define some terms and operations used in the (application of the) expansion rules, and directly comment on them:

Nominal Nodes and Blockable Nodes We distinguish two types of nodes in $\mathbf{G}$, *nominal* nodes and *blockable* nodes. A node x is a nominal node if $\mathcal{L}(x)$ contains a nominal. A node that is not a nominal node is a blockable node. A nominal $o \in N_I$ is said to be *new in* $\mathbf{G}$ if no node in $\mathbf{G}$ has o in its label.
Comment: like ABox individuals [13], nominal nodes can be arbitrarily interconnected. In contrast, blockable nodes are only found in tree-like structures rooted in nominal nodes (or in r_0); a branch of such a tree may simply end, possibly with a *blocked* node (defined below) as a leaf, or have an edge leading to a nominal node. In case a branch ends in a blocked node, we use standard *unravelling* to construct a tableau from the completion graph, and thus the resulting tableau will contain infinitely many copies of the nodes on the path from the blocking node to the blocked node. This is why there can be no nominal nodes on this path.

In the *NN*-rule, we use *new* nominals to create new nominal nodes—intuitively, to fix the identity of certain, constrained neighbours of nominal nodes. As we will show, it is possible to fix an upper bound on the number of nominal nodes that can be generated in a given completion graph; this is crucial for termination of the construction, given that blocking cannot be applied to nominal nodes.

Blocking A node x is *label blocked* if it has ancestors x', y and y' such that

1. x is a successor of x' and y is a successor of y',
2. y, x and all nodes on the path from y to x are blockable,
3. $\mathcal{L}(x) = \mathcal{L}(y)$ and $\mathcal{L}(x') = \mathcal{L}(y')$, *and*
4. $\mathcal{L}(\langle x', x\rangle) = \mathcal{L}(\langle y', y\rangle)$.

In this case, we say that y *blocks* x. A node is *blocked* if either it is label blocked or it is blockable and its predecessor is blocked; if the predecessor of a safe node x is blocked, then we say that x is *indirectly blocked*.
Comment: blocking is defined exactly as for $\mathcal{SHIQ}$, with the only difference that, in the presence of nominals, we must take care that none of the nodes between a blocking and a blocked one is a nominal node.

Generating and Shrinking Rules and Safe Neighbours The $\geqslant$-, $\exists$- and *NN*-rules are called *generating rules*, and the $\leqslant$- and the o-rule are called *shrinking rules*. An R-neighbour y of a node x is *safe* if (i) x is blockable or if (ii) x is a nominal node and y is not blocked.
Comment: generating rules add new nodes to the completion graph, whereas shrinking rules remove nodes—they merge all information concerning one node into another one (e.g., to satisfy atmost number restrictions), and then remove

the former node. We need the safety of R-neighbours to ensure that enough R-neighbours for nominal nodes are generated.

Pruning When a node y is *merged* into a node x, we "prune" the completion graph by removing y and, recursively, all blockable successors of y. More precisely, pruning a node y (written $\mathsf{Prune}(y)$) in $\mathbf{G} = (V, E, \mathcal{L}, \neq)$ yields a graph that is obtained from $\mathbf{G}$ as follows:

1. for all successors z of y, remove $\langle y, z\rangle$ from E and, if z is blockable, $\mathsf{Prune}(z)$;
2. remove y from V.

Merging In some rules, we "merge" one node into another node. Intuitively, when we merge a node y into a node x, we add $\mathcal{L}(y)$ to $\mathcal{L}(x)$, "move" all the edges leading *to* y so that they lead to x and "move" all the edges leading from y to nominal nodes so that they lead from x to the same nominal nodes; we then remove y (and blockable sub-trees below y) from the completion graph. More precisely, merging a node y into a node x (written $\mathsf{Merge}(y, x)$) in $\mathbf{G} = (V, E, \mathcal{L}, \neq)$ yields a graph that is obtained from $\mathbf{G}$ as follows:

1. for all nodes z such that $\langle z, y\rangle \in E$
   (a) if $\{\langle x, z\rangle, \langle z, x\rangle\} \cap E = \emptyset$, then add $\langle z, x\rangle$ to E and set $\mathcal{L}(\langle z, x\rangle) = \mathcal{L}(\langle z, y\rangle)$,
   (b) if $\langle z, x\rangle \in E$, then set $\mathcal{L}(\langle z, x\rangle) = \mathcal{L}(\langle z, x\rangle) \cup \mathcal{L}(\langle z, y\rangle)$,
   (c) if $\langle x, z\rangle \in E$, then set $\mathcal{L}(\langle x, z\rangle) = \mathcal{L}(\langle x, z\rangle) \cup \{\mathsf{Inv}(S) \mid S \in \mathcal{L}(\langle z, y\rangle)\}$, and
   (d) remove $\langle z, y\rangle$ from E;
2. for all nominal nodes z such that $\langle y, z\rangle \in E$
   (a) if $\{\langle x, z\rangle, \langle z, x\rangle\} \cap E = \emptyset$, then add $\langle x, z\rangle$ to E and set $\mathcal{L}(\langle x, z\rangle) = \mathcal{L}(\langle y, z\rangle)$,
   (b) if $\langle x, z\rangle \in E$, then set $\mathcal{L}(\langle x, z\rangle) = \mathcal{L}(\langle x, z\rangle) \cup \mathcal{L}(\langle y, z\rangle)$,
   (c) if $\langle z, x\rangle \in E$, then set $\mathcal{L}(\langle z, x\rangle) = \mathcal{L}(\langle z, x\rangle) \cup \{\mathsf{Inv}(S) \mid S \in \mathcal{L}(\langle y, z\rangle)\}$, and
   (d) remove $\langle y, z\rangle$ from E;
3. set $\mathcal{L}(x) = \mathcal{L}(x) \cup \mathcal{L}(y)$;

4. add $x \not\approx z$ for all z such that $y \not\approx z$; and

5. $\mathsf{Prune}(y)$.

If y was merged into x, we call x a *direct heir* of y, and we use being an *heir* of another node for the transitive closure of being a "direct heir".
Comment: merging is the generalisation of what is often done to satisfy an atmost number restriction for a node x in case that x has too many neighbours. However, since we might need to merge nominal nodes that are related in some arbitrary, non-tree-like way, merging gets slightly more tricky since we must take care of all incoming and outgoing edges. The usage of "heir" is quite intuitive since, after y has been merged into x, x has "inherited" all of y's properties, i.e., its label, its inequalities, and its incoming and outgoing edges (except for any outgoing edges removed by Prune).

Level (of Nominal Nodes) Let $o_1, \ldots, o_\ell$ be all the nominals occurring in the input concept D. We define the *level* of a node inductively as follows:

- each (nominal) node x with an $o_i \in \mathcal{L}(x)$, $1 \leq i \leq \ell$, is of level 0, and
- a nominal node x is of level i if x is not of some level $j < i$ and x has a neighbour that is of level $i - 1$.

Comment: if a node with a lower level is merged into another node, the level of the latter node may be reduced, but it can never be increased because Merge preserves all edges connecting nominal nodes. The completion graph initially contains only level 0 nodes.

Strategy (of Rule Application) the expansion rules are applied according to the following strategy:

1. the o-rule is applied with highest priority,
2. next, the $\leqslant$- and the *NN*-rule are applied, and they are applied first to nominal nodes with lower levels (before they are applied to nodes with higher levels). In case they are both applicable to the same node, the *NN*-rule is applied first.
3. all other rules are applied with a lower priority.

Comment: this strategy is necessary for termination, and in particular to fix an upper bound on the number of applications of the *NN*-rule. The general idea is to apply shrinking rules before any other rules (with the exception that the *NN*-rule

is applied to a node *before* applying the $\leqslant$-rule to it), and to apply these "crucial" rules to lower level nodes before applying them to higher level nodes.

We are now ready to finish the description of the tableau algorithm:

A completion graph is *complete* if it contains a clash, or when none of the rules is applicable. If the expansion rules can be applied to D and $\mathcal{R}$ in such a way that they yield a complete, clash-free completion graph, then the algorithm returns "D is *satisfiable* w.r.t. $\mathcal{R}$", and "D is *unsatisfiable* w.r.t. $\mathcal{R}$" otherwise.

4.2 Example application of the algorithm

We consider two examples, a rather easy one and a slightly more tricky one.

First, consider the TBox

$$\mathcal{T} = \{A \doteq \exists R.(A \sqcap \exists R.A), \\ A \doteq o\}.$$

As described in Section 2, to decide the satisfiability of A w.r.t. $\mathcal{T}$ (and the empty role hierarchy), we test the following concept

$$A \sqcap C_{\mathcal{T}} \sqcap \forall U.C_{\mathcal{T}} \sqcap \exists U.o$$

for satisfiability w.r.t. $\{R \sqsubseteq U, R^- \sqsubseteq U\}$, where U is transitive and $C_{\mathcal{T}} = (\neg A \sqcup \exists R.(A \sqcap \exists R.A)) \sqcap (\neg A \sqcup o)$.

Our tableau algorithm starts with a completion graph consisting of two nodes, r_0 and r_1, with $\mathcal{L}(r_0) = \{A \sqcap C_{\mathcal{T}} \sqcap \forall U.C_{\mathcal{T}} \sqcap \exists U.o\}$ and $\mathcal{L}(r_1) = \{o\}$. After some applications of the $\sqcap$- and the $\sqcup$-rule (the latter in such a way that we do not get a clash $A, \neg A \in \mathcal{L}(r_0)$), we obtain a completion graph with

$$\mathcal{L}(r_0) = \{A, C_{\mathcal{T}}, \exists R.(A \sqcap \exists R.A), o, \forall U.C_{\mathcal{T}}, \exists U.o\}.$$

Now the o-rule is applied immediately, and we merge r_0 and r_1, which leaves us with a single node, say r_0.

Next, we apply the $\exists$-rule three times and the $\sqcap$-rule several times, which yields three new nodes x_0, x_1, and y_0, where x_0 is an R-successors of r_0, y_0 is a U-successor of r_0, and x_1 is an R-successor of x_0 with

$$\begin{aligned} \mathcal{L}(x_0) &= \{A \sqcap \exists R.A, A, \exists R.A\}, \\ \mathcal{L}(x_1) &= \{A\}, \\ \mathcal{L}(y_0) &= \{o\}. \end{aligned}$$

Next, we merge y_0 into r_0 using the o-rule, which makes r_0 a U-successor of itself.

Since x_0 is also U-successor of r_0, we can apply the $\forall$-rule and the $\forall_+$-rule to $\forall U.C_{\mathcal{T}} \in \mathcal{L}(r_0)$ (recall that U is transitive), which adds $C_{\mathcal{T}}$ and $\forall U.C_{\mathcal{T}}$ to x_0's

$\sqcap$-rule:	if 1. $C_1 \sqcap C_2 \in \mathcal{L}(x)$, x is not indirectly blocked, and 2. $\{C_1, C_2\} \not\subseteq \mathcal{L}(x)$ then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1, C_2\}$
$\sqcup$-rule:	if 1. $C_1 \sqcup C_2 \in \mathcal{L}(x)$, x is not indirectly blocked, and 2. $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$ then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C\}$ for some $C \in \{C_1, C_2\}$
$\exists$-rule:	if 1. $\exists S.C \in \mathcal{L}(x)$, x is not blocked, and 2. x has no safe S-neighbour y with $C \in \mathcal{L}(y)$, then create a new node y with $\mathcal{L}(\langle x, y\rangle) = \{S\}$ and $\mathcal{L}(y) = \{C\}$
$\forall$-rule:	if 1. $\forall S.C \in \mathcal{L}(x)$, x is not indirectly blocked, and 2. there is an S-neighbour y of x with $C \notin \mathcal{L}(y)$ then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$
$\forall_+$-rule:	if 1. $\forall S.C \in \mathcal{L}(x)$, x is not indirectly blocked, and 2. there is some R with $\mathsf{Trans}(R)$ and $R \sqsubseteq^* S$, 3. there is an R-neighbour y of x with $\forall R.C \notin \mathcal{L}(y)$ then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{\forall R.C\}$
choose-rule:	if 1. $(\leqslant n S.C) \in \mathcal{L}(x)$, x is not indirectly blocked, and 2. there is an S-neighbour y of x with $\{C, \dot{\neg}C\} \cap \mathcal{L}(y) = \emptyset$ then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{E\}$ for some $E \in \{C, \dot{\neg}C\}$
$\geqslant$-rule:	if 1. $(\geqslant n S.C) \in \mathcal{L}(x)$, x is not blocked, and 2. there are not n safe S-neighbours $y_1, \ldots, y_n$ of x with $C \in \mathcal{L}(y_i)$ and $y_i \neq y_j$ for $1 \leq i < j \leq n$ then create n new nodes $y_1, \ldots, y_n$ with $\mathcal{L}(\langle x, y_i\rangle) = \{S\}$, $\mathcal{L}(y_i) = \{C\}$, and $y_i \neq y_j$ for $1 \leq i < j \leq n$.
$\leqslant$-rule:	if 1. $(\leqslant n S.C) \in \mathcal{L}(z)$, z is not indirectly blocked, and 2. $\sharp S^{\mathbf{G}}(z, C) > n$ and there are two S-neighbours x, y of z with $C \in \mathcal{L}(x) \cap \mathcal{L}(y)$, and not $x \neq y$ then 1. if x is a nominal node, then $\mathsf{Merge}(y, x)$ 2. else if y is a nominal node or an ancestor of x, then $\mathsf{Merge}(x, y)$ 3. else $\mathsf{Merge}(y, x)$

Figure 1: The tableaux expansion rules for $\mathcal{SHIQ}$

label. Similarly, the these rules add $C_\mathcal{T}$ and $\forall U.C_\mathcal{T}$ to $\mathcal{L}(x_1)$ when applied to $\forall U.C_\mathcal{T} \in \mathcal{L}(x_0)$.

After further application of the $\sqcap$- and the $\sqcup$-rule (again, the latter in such a way that we do not get a clash $A, \neg A \in \mathcal{L}(x_i)$), we obtain a completion graph with

$$\mathcal{L}(x_0) = \mathcal{L}(x_1) = \{A \sqcap \exists R.A, A, \exists R.A, C_\mathcal{T}, \forall U.C_\mathcal{T}, \exists R.(A \sqcap \exists R.A), o\}$$

o-rule:	if	for some $o \in N_I$ there are 2 nodes x, y with $o \in \mathcal{L}(x) \cap \mathcal{L}(y)$ and not $x \neq y$
	then	$\mathsf{Merge}(x, y)$
NN-rule:	if	1. $(\leqslant nS.C) \in \mathcal{L}(x)$, x is a nominal node, and there is a blockable S-neighbour y of x such that $C \in \mathcal{L}(y)$ and x is a successor of y,
		2. there is no m such that $1 \leqslant m \leqslant n$, $(\leqslant mS.C) \in \mathcal{L}(x)$, and there exist m nominal S-neighbours $z_1, \ldots, z_m$ of x with $C \in \mathcal{L}(z_i)$ and $z_i \neq z_j$ for all $1 \leq i < j \leq m$.
	then	1. guess m with $1 \leqslant m \leqslant n$ and set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{(\leqslant mS.C)\}$
		2. create m new nodes $y_1, \ldots, y_m$ with $\mathcal{L}(\langle x, y_i \rangle) = \{S\}$, $\mathcal{L}(y_i) = \{C, o_i\}$ for each $o_i \in N_I$ new in $\mathbf{G}$, and $y_i \neq y_j$ for $1 \leq i < j \leq m$,

Figure 2: The new expansion rules for $\mathcal{SHOIQ}$: the o-rule and the *NN*-rule

Finally, we apply the o-rule twice and obtain a graph that consists of a single node, say r_0, which is an R-successor of itself, and with

$$\mathcal{L}(r_0) = \{A, C_{\mathcal{T}}, \exists R.(A \sqcap \exists R.A), o, \forall U.C_{\mathcal{T}}, \exists U.o, A \sqcap \exists R.A, \exists R.A\}.$$

This completion graph is complete and clash-free, hence our algorithm returns "satisfiable". Indeed, it corresponds to a model $\mathcal{I}$ with a single elements r and $U^{\mathcal{I}} = R^{\mathcal{I}} = \{(r, r)\}$ and $A^{\mathcal{I}} = o^{\mathcal{I}} = \{r\}$. Please notice that our strategy of applying the o-rule with highest priority was crucial for termination: otherwise, we might have continued generating an infinite chain of R-successors of x_2, even while merging some of the x_i with r_0.

Secondly, we discuss a slightly more involved example. Here, we do not present all details, but only sketch the run of the algorithm. We invite the reader to verify on this example that the *NN*-rule is indeed needed.

Assume we want to decide the satisfiability of $o_1 \sqcap \exists R_1^-.\top$ w.r.t. the following TBox, where we use $\top$ as an abbreviation of $A \sqcup \neg A$:

$$\begin{aligned}\mathcal{T} = \{&o_1 \doteq (\leqslant 2R_1^-.\top) \sqcap \forall R_1^-.\exists S_1.\exists S_2^-.\exists R_2.o_2,\\ &o_2 \doteq (\leqslant 2R_2^-.\top) \sqcap \forall R_2^-.\exists S_2.\exists S_1^-.\exists R_1.o_1\}.\end{aligned}$$

Again, we would internalise our TBox and use a transitive super-role U of R_i, S_i, and their inverses. Let C be the result of internalising $\mathcal{T}$ into $o_1 \sqcap \exists R_1^-.\top$. We start the algorithm with three nodes, say r_0, r_1, and r_2, but after few applications of the $\sqcap$-rule to $C \in \mathcal{L}(r_0)$, we find $o_1 \in \mathcal{L}(r_0)$, and thus merge r_0 and r_1, keeping r_1. After further applications of the $\sqcap$- and the $\sqcup$-rules (again, without causing a

clash), we can apply the $\exists$-rule to $\exists R_1^-.\top \in \mathcal{L}(r_1)$ and to $\exists U.o_i \in \mathcal{L}(r_1)$, for $i \in \{1, 2\}$. The latter creates two nodes that are immediately merged into r_1 and r_2, respectively, and will make each r_i a U-successor of r_1. The former creates an R_1^--successor of r_1, and we can thus apply the $\forall$-rule to $\forall R_1^-.\exists S_1.\exists S_2^-.\exists R_2.o_2 \in \mathcal{L}(r_1)$. Next, we can apply three more times the $\exists$-rule, and obtain a chain of the following form: r_1 has an R_1^--successor x_0, which has an S_1-successor x_1, which has an S_2^--successor x_2, which has an R_2-successor x_3 with $o_2 \in \mathcal{L}(x_3)$. Thus we need to apply the o-rule and merge x_3 into r_2, which becomes an R_2-successor of x_2. As a consequence, the *NN*-rule becomes applicable to $(\leqslant 2R_2^-.\top) \in \mathcal{L}(r_2)$.[6] We guess $m = 2$ and create two new R_2^--successors n_1, n_2 of r_2 with $\mathcal{L}(n_i) = \{\hat{o}_i\}$ and $n_1 \neq n_2$. Now r_2 has three R_2^--neighbours, and we can apply the $\leqslant$-rule to r_2, to merge x_2 into n_1. Please note that $\neq$ prevents us from merging n_1 into n_2 or vice versa.

Since n_1 is an R_2^--neighbour of r_2, the $\forall$-rule adds $\exists S_2.\exists S_1^-.\exists R_1.o_1$ to n_1. Three application of the $\exists$-rule yield a chain similar to the first one: r_2 has an R_2^--successor n_1, which has an S_2-successor x_1', which has an S_2-successor x_0', which has an R_1-successor y with $o_1 \in \mathcal{L}(y)$. Next, y is merged with r_1, and thus r_1 is an R_1-successor of x_0'. Again, as a consequence, the *NN*-rule becomes applicable, and we guess again $m = 2$ and create two new R_1^--successors m_1, m_2 of r_1 with $\mathcal{L}(m_i) = \{\tilde{o}_i\}$ and $m_1 \neq m_2$. Again, we can apply the $\leqslant$-rule to r_1. Assume that we merge both x_0 and x_0' into m_1.

After this, we can apply the $\forall$-rule to add $\exists S_1.\exists S_2^-.\exists R_2.o_2$ to $\mathcal{L}(m_2)$ and $\exists S_2.\exists S_1^-.\exists R_1.o_1$ to $\mathcal{L}(n_2)$. This yields two more chains between r_1 and r_2, both going via n_i and m_j after two more applications of the $\leqslant$-rule—once to r_1 and once to r_2. Then the tableau algorithm stops with a complete and clash-free completion graph.

Let us point out some important properties of our algorithm that were having effects in this application:

- We can apply the *NN*-rule only to a nominal node r and some $(\leqslant nR.C) \in \mathcal{L}(r)$ if there is a blockable node x that has r as its R^--successor. Hence we could only apply it to r_1 after it got his second R_1^--neighbour x_0'.

- The newly created nominal nodes made us merge newly created R_i^--neighbours of r_i immediately into the new nominal nodes n_j and m_ℓ. Without the explicit inequalities $n_1 \neq n_2$ and $m_1 \neq m_2$ between these new nominal nodes, we could have created another kind of "yo-yo" effect: when applying the $\leqslant$-rule to $(\leqslant 2R_i^-.\top) \in \mathcal{L}(r_i)$, we could have always merged those nodes that both already have the exists restrictions in their labels, and

[6] We assume that $\top$ is present in all node labels.

to which the $\exists$-rule has already been applied. This would clearly cause non-termination.

- If one were to implement the algorithm for internalised TBoxes, one should clearly start with a completion graph where all initial nominal nodes r_i are already inter-related via (the approximation of) the universal role U and omit the $\exists U.o_i$ from the input concept. However, for optimisation purposes, TBoxes will be treated directly, and this is thus not necessary. To treat TBoxes directly, for each GCI $C \dot{\sqsubseteq} D \in \mathcal{T}$, we add $\neg C \sqcup D$ to the label of each node, while using standard pre-processing and optimisation techniques [8].

4.3 Proof of the algorithm's correctness and termination

Lemma 6 Let D be a $\mathcal{SHOIQ}$ concept in NNF and $\mathcal{R}$ a role hierarchy.

1. When started with D and $\mathcal{R}$, the tableau algorithm terminates.
2. D has a tableau w.r.t. $\mathcal{R}$ if and only if the expansion rules can be applied to D and $\mathcal{R}$ such that they yield a complete, clash-free completion graph.

Proof: Let $m = |\mathsf{cl}(D)|$, k the number of roles and their inverses in D and $\mathcal{R}$, $(n_{\geqslant})$ the maximal number in atleast number restrictions, $(n_{\leqslant})$ the maximal number in atmost number restrictions, and $o_1, \ldots, o_\ell$ be all nominals occurring in D, and let $\lambda := 2^{2m+k}$. The algorithm constructs a graph that consists of a set of arbitrarily interconnected nominal nodes, and "trees" of blockable nodes with each tree rooted in r_0 or in a nominal node, and where branches of these trees might end in an edge leading to a nominal node.

Termination is a consequence of the usual $\mathcal{SHIQ}$ conditions with respect to the blockable tree parts of the graph, plus the fact that there is a bound on the number of new nominal nodes that can be added to $\mathbf{G}$ by the *NN*-rule. More precisely, termination is due to the following four properties, the first three of which are very similar to those used in the termination proof for $\mathcal{SHIQ}$ given in [12], and the last of which provides the upper bound on the number of new nominal nodes generated by the *NN*-rule.

1. All but the shrinking rules strictly extend the completion graph by adding new nodes (and edges) or extending node labels, while neither removing nodes (or edges) nor removing elements from node labels. This is an obvious consequence of the definition of the rules.

2. New nodes are only added by the generating rules, and each of these rules can be triggered at most once for a given concept in the label of a given node x or in its heirs.

   This is obvious if no shrinking rule is applied. If such a rule is applied, then, intuitively, this observation is due to the fact that, if an S-neighbour y of x is merged into a node z, then $\mathcal{L}(y)$ is added to $\mathcal{L}(z)$, z "inherits" all of the inequalities from y, and either z is an S-neighbour of x (if x is a nominal node or if y is a successor of x), or x is removed from the graph by an application of $\mathsf{Prune}(x)$ (if x is a blockable node and x is a successor of y). More precisely, we distinguish the following three cases.

   - For the $\exists$-rule, if it is applied to a concept $\exists S.C \in \mathcal{L}(x)$, then a new node y of x is created with $\mathcal{L}(\langle x, y\rangle) = S$ and $\mathcal{L}(y) = C$. Subsequently, either x is removed from the graph, or x has an S-neighbour y' which is an heir of y, i.e., with $C \in \mathcal{L}(y')$. Hence the $\exists$-rule is no longer applicable to $\exists S.C \in \mathcal{L}(x)$.
   - For the $\geqslant$-rule, if it is applied to a concept $(\geqslant n S.C) \in \mathcal{L}(x)$, then n new nodes $y_1, \ldots, y_n$ are created with $\mathcal{L}(\langle x, y_i\rangle) = \{S\}$, $\mathcal{L}(y_i) = \{C\}$, and $y_i \neq y_j$ for $1 \leq i < j \leq n$. Subsequently, either x is removed from the graph, or x has n S-neighbours $y'_1, \ldots, y'_n$ which are heirs of the y_i, i.e., $C \in \mathcal{L}(y'_i)$ and $y'_i \neq y'_j$ for $1 \leq i < j \leq n$. Hence the $\geqslant$-rule is no longer applicable to $(\geqslant n S.C) \in \mathcal{L}(x)$.
   - For the *NN*-rule, if it is applied to a concept $(\leqslant n S.C) \in \mathcal{L}(x)$, then for some m with $1 \leqslant m \leqslant n$, m new nominal nodes $y_1, \ldots, y_m$ are created with $\mathcal{L}(\langle x, y_i\rangle) = \{S\}$, $\mathcal{L}(y_i) = \{C\}$, $y_i \neq y_j$ for $1 \leq i < j \leq m$, and $(\leqslant m S.C) \in \mathcal{L}(x)$. Subsequently, either x is removed from the graph, or $(\leqslant m S.C)$ is still in $\mathcal{L}(x)$ and x has m S-neighbours $y'_1, \ldots, y'_m$ which are heirs of the y_i, i.e., $C \in \mathcal{L}(y'_i)$ and $y'_i \neq y'_j$ for $1 \leq i < j \leq m$. Hence the *NN*-rule is no longer applicable to $(\leqslant n S.C) \in \mathcal{L}(x)$.

   As for the $\mathcal{SHIQ}$ case, a generating rule being applied to a concept in the label of a node x can generate at most $(n_{\geqslant})$ blockable successors. As there are at most m concepts in $\mathcal{L}(x)$, a node can have at most $m(n_{\geqslant})$ blockable successors.

3. As for $\mathcal{SHIQ}$ [12], the blocking condition ensures that the length of a path consisting entirely of blockable nodes is bounded by λ. This is due to the fact that, for x a blockable node, $\mathcal{L}(x) \subseteq \mathsf{cl}(D, \mathcal{R})$ and thus does not contain any nominals, neither those contained in the input nor those added later by the *NN*-rule.

4. The number of nominal nodes is bounded by $\mathcal{O}(\ell(m(n_{\leqslant}))^{\lambda})$.

   First, we observe that the *NN*-rule can only be applicable after a nominal has been added to the label of a blockable node x in a branch of one of the blockable "trees" rooted in r_o or in a nominal node; otherwise, it is not possible that a blockable node has a nominal node as a successor. Now $o_1, \ldots, o_\ell$ are the only nominals that can be added to the label of a blockable node (we assume that the nodes generated by the *NN*-rule are nominal nodes right from their generation), and thus x is a level 0 node, and the o-rule (which is applied with top priority) will ensure that x is immediately merged with an existing level 0 node having the same nominal in its label.

   As a consequence of this merging of x with a nominal node, say r_i, it is possible that the predecessor of x is merged into a nominal node n_1 by the $\leqslant$-rule (due to the pruning part of merging, this cannot happen to a successor of x). By definition, n_1 is of level 0 or 1. Repeating this argument, it is possible that all ancestors of x are merged into nominal nodes.

   However, as the maximum length of a sequence of blockable nodes is λ, blockable ancestors of x can only be merged into nominal nodes of level below λ. This together with the precondition of the *NN*-rule implies that we can only apply the *NN*-rule to nominal nodes of level below λ.

   Secondly, when the *NN*-rule has been applied to a concept $(\leqslant nR.C)$ in the label of a node x, it can never be applied again to $(\leqslant nR.C)$ in the label of x or an heir of x.

   The remainder is a simple counting exercise: the *NN*-rule can be applied at most m times to a given nominal node, and each such application can generate at most $(n_{\leqslant})$ new nominal nodes. As $\mathbf{G}$ was initialised with ℓ nominal nodes, the *NN*-rule can be applied at most ℓm times to level 0 nodes and can generate at most $\ell m(n_{\leqslant})$ level 1 nodes; similarly, the *NN*-rule can be applied at most $\ell m(m(n_{\leqslant}))^{i-1}$ to level i nodes, and generate at most $\ell(m(n_{\leqslant}))^{i}$ level $i+1$ nodes. As the *NN*-rule is only applicable to nodes with level $< \lambda$ and the level of a node or its heirs can only decrease, this gives an upper bound of

$$\ell m \sum_{0 \leqslant i < \lambda} (m(n_{\leqslant}))^{i} = \ell m \frac{1 - (m(n_{\leqslant}))^{\lambda}}{1 - m(n_{\leqslant})}$$

   on the number of times that the *NN*-rule can be applied, and an upper bound of

$$\ell \sum_{0 < i \leqslant \lambda} (m(n_{\leqslant}))^{i} = \ell \frac{1 - (m(n_{\leqslant}))^{\lambda+1}}{1 - m(n_{\leqslant})} - \ell$$

   on the number of nominal nodes that can be generated.

To sum up, there is a bound $\mathcal{O}(\ell(m(n_{\leqslant}))^{\lambda})$ on the number of nominal nodes that can be generated, and this implies a bound on the number of blockable nodes. Hence any sequence of rule applications must eventually result in $\mathbf{G}$ being complete.

For the second claim in Lemma 6, for the "if" direction, we can obtain a tableau $T = (\mathbf{S}, \mathcal{L}', \mathcal{E})$ from a complete and clash-free completion graph $\mathbf{G}$ by *unravelling* blockable "tree" parts of the graph as usual (these are the only parts where blocking can apply).

More precisely, paths are defined as follows. For a label blocked node x, let $b(x)$ denote a node that blocks x.

A *path* is a sequence of pairs of blockable nodes of $\mathbf{G}$ of the form $p = \langle (x_0, x'_0), \ldots, (x_n, x'_n) \rangle$. For such a path, we define $\mathsf{Tail}(p) := x_n$ and $\mathsf{Tail}'(p) := x'_n$. With $\langle p | (x_{n+1}, x'_{n+1}) \rangle$ we denote the path $\langle (x_0, x'_0), \ldots, (x_n, x'_n), (x_{n+1}, x'_{n+1}) \rangle$. The set $\mathsf{Paths}(\mathbf{G})$ is defined inductively as follows:

- For each blockable node x of $\mathbf{G}$ that is a successor of a nominal node or a root node, $\langle (x, x) \rangle \in \mathsf{Paths}(\mathbf{G})$, and
- For a path $p \in \mathsf{Paths}(\mathbf{G})$ and a blockable node y in $\mathbf{G}$:
  - if y is a successor of $\mathsf{Tail}(p)$ and y is not blocked, then $\langle p | (y, y) \rangle \in \mathsf{Paths}(\mathbf{G})$, and
  - if y is a successor of $\mathsf{Tail}(p)$ and y is blocked, then $\langle p | (b(y), y) \rangle \in \mathsf{Paths}(\mathbf{G})$.

Please note that, due to the construction of Paths, all nodes occurring in a path are blockable and, for $p \in \mathsf{Paths}(\mathbf{G})$ with $p = \langle p' | (x, x') \rangle$, x is not blocked, x' is blocked iff $x \neq x'$, and x' is never indirectly blocked. Furthermore, the blocking condition implies $\mathcal{L}(x) = \mathcal{L}(x')$.

Next, we use $\mathsf{Nom}(\mathbf{G})$ for the set of nominal nodes in $\mathbf{G}$, and define a tableau $T = (\mathbf{S}, \mathcal{L}', \mathcal{E})$ from $\mathbf{G}$ as follows.

$$\mathbf{S} = \mathsf{Nom}(\mathbf{G}) \cup \mathsf{Paths}(\mathbf{G})$$

$$\mathcal{L}'(p) = \begin{cases} \mathcal{L}(\mathsf{Tail}(p)) & \text{if } p \in \mathsf{Paths}(\mathbf{G}) \\ \mathcal{L}(p) & \text{if } p \in \mathsf{Nom}(\mathbf{G}) \end{cases}$$

$$\begin{aligned} \mathcal{E}(R) = \{\langle p, q\rangle \in \mathsf{Paths}(\mathbf{G}) \times \mathsf{Paths}(\mathbf{G}) \mid \\ q = \langle p|(x, x')\rangle \text{ and } x' \text{ is an } R\text{-successor of } \mathsf{Tail}(p) \text{ or} \\ p = \langle q|(x, x')\rangle \text{ and } x' \text{ is an } \mathsf{Inv}(R)\text{-successor of } \mathsf{Tail}(q)\} \cup \end{aligned}$$

$$\begin{aligned} &\{\langle p, x\rangle \in \mathsf{Paths}(\mathbf{G}) \times \mathsf{Nom}(\mathbf{G}) \mid x \text{ is an } R\text{-neighbour of } \mathsf{Tail}(p)\} \cup \\ &\{\langle x, p\rangle \in \mathsf{Nom}(\mathbf{G}) \times \mathsf{Paths}(\mathbf{G}) \mid \mathsf{Tail}(p) \text{ is an } R\text{-neighbour of } x\} \cup \\ &\{\langle x, y\rangle \in \mathsf{Nom}(\mathbf{G}) \times \mathsf{Nom}(\mathbf{G}) \mid y \text{ is an } R\text{-neighbour of } x\} \end{aligned}$$

We already commented above on $\mathbf{S}$, and $\mathcal{L}'$ is straightforward. Unfortunately, $\mathcal{E}$ is slightly cumbersome because we must distinguish between blockable and nominal nodes.

CLAIM: T is a tableau for D with respect to $\mathcal{R}$.

Firstly, by definition of the algorithm, there is an heir x_0 of r_0 with $D \in \mathcal{L}(x_0)$. By the $\leqslant$-rule, x_0 is either a root node or a nominal node, and thus cannot be blocked. Hence there is some $s \in \mathbf{S}$ with $D \in \mathcal{L}'(s)$. Next, we prove that T satisfies each (Pi).

- (P1) to (P3) are trivially implied by the definition of $\mathcal{L}'$ and completeness of $\mathbf{G}$.
- for (P4), consider a tuple $\langle s, t\rangle \in \mathcal{E}(R)$ with $\forall R.C \in \mathcal{L}'(s)$. We distinguish four different cases:
  - if $\langle s, t\rangle \in \mathsf{Paths}(\mathbf{G}) \times \mathsf{Paths}(\mathbf{G})$, then $\forall R.C \in \mathcal{L}(\mathsf{Tail}(s))$ and
    - either $\mathsf{Tail}'(t)$ is an R-successor of $\mathsf{Tail}(s)$. Hence completeness implies $C \in \mathcal{L}(\mathsf{Tail}'(t))$, and either $\mathsf{Tail}'(t) = \mathsf{Tail}(t)$ or the blocking condition implies $\mathcal{L}(\mathsf{Tail}'(t)) = \mathcal{L}(\mathsf{Tail}(t))$.
    - or $\mathsf{Tail}'(s)$ is an $\mathsf{Inv}(R)$-successor of $\mathsf{Tail}(t)$. Either $\mathsf{Tail}'(t) = \mathsf{Tail}(t)$ or the blocking condition implies $\forall R.C \in \mathcal{L}(\mathsf{Tail}'(s))$, and thus completeness implies that $C \in \mathcal{L}(\mathsf{Tail}(t))$.
  - if $\langle s, t\rangle \in \mathsf{Nom}(\mathbf{G}) \times \mathsf{Nom}(\mathbf{G})$, then $\forall R.C \in \mathcal{L}(s)$ and t is an R-neighbour of s. Hence completeness implies $C \in \mathcal{L}(t)$.
  - if $\langle s, t\rangle \in \mathsf{Nom}(\mathbf{G}) \times \mathsf{Paths}(\mathbf{G})$, then $\forall R.C \in \mathcal{L}(s)$ and $\mathsf{Tail}(t)$ is an R-neighbour of s. Hence completeness implies $C \in \mathcal{L}(\mathsf{Tail}(t))$.

– if $\langle s,t\rangle \in \mathsf{Paths}(\mathbf{G}) \times \mathsf{Nom}(\mathbf{G})$, then $\forall R.C \in \mathcal{L}(\mathsf{Tail}(s))$ and t is an R-neighbour of $\mathsf{Tail}(s)$. Hence non-applicability of the $\forall$-rule implies $C \in \mathcal{L}(t)$.

In all four cases, by definition of $\mathcal{L}'$, we have $C \in \mathcal{L}'(t)$.

- for (P5), consider some $s \in \mathbf{S}$ with $\exists R.C \in \mathcal{L}'(s)$.
  - If $s \in \mathsf{Paths}(\mathbf{G})$, then $\exists R.C \in \mathcal{L}(\mathsf{Tail}(s))$, $\mathsf{Tail}(s)$ is not blocked, and completeness of $\mathcal{T}$ implies the existence of an R-neighbour y of $\mathsf{Tail}(s)$ with $C \in \mathcal{L}(y)$.
    * If y is a nominal node, then $y \in \mathbf{S}$, $C \in \mathcal{L}'(y)$, and $\langle s,y\rangle \in \mathcal{E}(R)$.
    * If y is blockable and a successor of $\mathsf{Tail}(s)$, then $\langle s|(\tilde{y},y)\rangle \in \mathbf{S}$, for $\tilde{y} = y$ or $\tilde{y} = b(y)$, $C \in \mathcal{L}'(\langle s|(\tilde{y},y)\rangle)$, and $\langle s,\langle s|(\tilde{y},y)\rangle\rangle \in \mathcal{E}(R)$.
    * If y is blockable and a predecessor of $\mathsf{Tail}(s)$, then $s = \langle p|(y,y)|(\mathsf{Tail}(s),\mathsf{Tail}'(s))\rangle$, $C \in \mathcal{L}'(\langle p|(y,y)\rangle)$, and $\langle s,\langle p|(y,y)\rangle\rangle \in \mathcal{E}(R)$.
  - If $s \in \mathsf{Nom}(\mathbf{G})$, then completeness implies the existence of some R-successor x of s with $C \in \mathcal{L}(x)$.
    * If x is a nominal node, then $\langle s,x\rangle \in \mathcal{E}(R)$ and $C \in \mathcal{L}'(x)$.
    * If x is a blockable node, then x is a safe R-neighbour of s and thus not blocked. Hence there is a path $p \in \mathsf{Paths}(\mathbf{G})$ with $\mathsf{Tail}(p) = x$, $\langle s,p\rangle \in \mathcal{E}(R)$ and $C \in \mathcal{L}'(p)$.
- (P6) is analogous to (P4).
- for (P7), consider some $s \in \mathbf{S}$ with $(\geqslant nR.C) \in \mathcal{L}'(s)$.
  - if $s \in \mathsf{Nom}(\mathbf{G})$, then completeness implies the existence of n safe R-neighbours $y_1,\ldots,y_n$ of s with and $y_j \neq y_j$, for each $i \neq j$, and $C \in \mathcal{L}(y_i)$, for each $1 \leq i \leq n$. By construction, each y_i corresponds to a $t_i \in \mathbf{S}$ with $t_i \neq t_j$, for each $i \neq j$:
    * if y_i is blockable, then it cannot be blocked since it is a safe R-neighbour of s. Hence there is a path $\langle p|(y_i,y_i)\rangle \in \mathbf{S}$ and $\langle s,\langle p|(y_i,y_i)\rangle\rangle \in \mathcal{E}(R)$.
    * if y_i is a nominal node, then $\langle s,y_i\rangle \in \mathcal{E}(R)$.
  - if $s \in \mathsf{Paths}(\mathbf{G})$, then completeness implies the existence of n R-neighbours $y_1,\ldots,y_n$ of $\mathsf{Tail}(s)$ with $y_j \neq y_j$, for each $i \neq j$, and $C \in \mathcal{L}(y_i)$, for each $1 \leq i \leq n$. By construction, each y_i corresponds to a $t_i \in \mathbf{S}$ with $t_i \neq t_j$, for each $i \neq j$:

- if y_i is safe, then it can be blocked if it is a successor of $\mathsf{Tail}(s)$. In this case, the "pair" construction in our definition of paths ensure that, even if $b(y_i) = b(y_j)$, for some $i \neq j$, we still have $\langle p|(b(y_i), y_i)\rangle \neq \langle p|(b(y_j), b_j)\rangle$.
- if y_i is unsafe, then $\langle s, y_i\rangle \in \mathcal{E}(R)$.

Hence all t_i are different and, by construction, $C \in \mathcal{L}'(t_i)$, for each $1 \leq i \leq n$.

- for (P8), consider some $s \in \mathbf{S}$ with $(\leqslant nR.C) \in \mathcal{L}'(s)$. Clash-freeness implies the existence of atmost n R-neighbours y_i of s with $C \in \mathcal{L}(y_i)$. By construction, each $t \in \mathbf{S}$ with $\langle s, t\rangle \in \mathcal{E}(R)$ corresponds to an R-neighbour y_i of s or $\mathsf{Tail}(s)$, and none of these R-neighbours gives raise to more than one such y_i. Moreover, since $\mathcal{L}'(t) = \mathcal{L}(y_i)$, (P8) is satisfied.

- (P9) is satisfied due to completeness of $\mathbf{G}$ and the fact that each $t \in \mathbf{S}$ with $\langle s, t\rangle \in \mathcal{E}(R)$ corresponds to an R-neighbour of s (in case $s \in \mathsf{Nom}(\mathbf{G})$) or of $\mathsf{Tail}(s)$ (in case $s \in \mathsf{Paths}(\mathbf{G})$).

- (P10) and (P11) are immediate consequences of the definition of "R-successor" and "R-neighbour".

- (P12) is due to completeness of $\mathbf{G}$ and the fact that nominal nodes are not "unravelled".

For the "only if" direction, given a tableau $T = (\mathbf{S}, \mathcal{L}', \mathcal{E})$ for D w.r.t. $\mathcal{R}$, we can apply the non-deterministic rules, i.e., the $\sqcup$-, *choose*-, $\leqslant$-, and *NN*-rule, in such a way that we obtain a complete and clash-free graph: inductively with the generation of new nodes, we define a mapping π from nodes in the completion graph to individuals in $\mathbf{S}$ of the tableau in such a way that,

1. for each node x, $\mathcal{L}(x) \subseteq \mathcal{L}'(\pi(x))$,
2. for each pair of nodes x, y and each role R, if y is an R-successor of x, then $\langle \pi(x), \pi(y)\rangle \in \mathcal{E}(R)$, and
3. $x \neq\!\!\!\!\cdot\; y$ implies $\pi(x) \neq \pi(y)$.

This is analogous to the proof in [12] with the additional observation that, due to (P12), application of the o-rule does not lead to a clash of the form (3). □

As an immediate consequence of Lemmas 6 and 4, the tableau algorithm always terminates, and answers with "D is satisfiable w.r.t. $\mathcal{R}$" iff D is satisfiable w.r.t. $\mathcal{R}$. Next, subsumption can be reduced to (un)satisfiability. Finally, as we mentioned in Section 2, $\mathcal{SHOIQ}$ can internalise general TBoxes.

Theorem 7 The tableau algorithm presented in Definition 5 is a decision procedure for satisfiability and subsumption of $\mathcal{SHOIQ}$ concepts w.r.t. TBoxes and role hierarchies.

5 Outlook

In this report, we have presented what is, to the best of our knowledge, the first goal-directed decision procedure for $\mathcal{SHOIQ}$ (and so $\mathcal{SHOIN}$). Given that $\mathcal{SHOIQ}$ is NExpTime-complete [19], it is clear that, in the worst case, any decision procedure will behave very badly, i.e., not terminate in practise. However, the algorithm given here is designed to behave well in many typically encountered cases, and to exhibit a "pay as you go" behaviour: if an input TBox, role hierarchy, and concept do not involve any one of inverse roles, number restrictions, or nominals, then the *NN*-rule will not be applied, and the corresponding non-deterministic guessing is avoided. This is even true for inputs that do involve all of these three constructors, but only in a "harmless" way. Hence, our $\mathcal{SHOIQ}$ algorithm can be implemented to perform just as well on $\mathcal{SHIQ}$ knowledge bases as state-of-the-art DL reasoners for $\mathcal{SHIQ}$ [9, 5, 17]. To find out whether our algorithm can handle some non-trivial, "true" $\mathcal{SHOIQ}$ inputs, we are currently extending a highly optimised DL reasoner, FaCT++ [21], to implement the algorithm described here.

References

[1] Franz Baader and Philipp Hanschke. A schema for integrating concrete domains into concept languages. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, pages 452–457, 1991.

[2] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. OWL web ontology language reference. W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/owl-ref/`.

[3] Patrick Blackburn and Jerry Seligman. Hybrid languages. *J. of Logic, Language and Information*, 4:251–272, 1995.

[4] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Description logic framework for information integration. In *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 2–13, 1998.

[5] Volker Haarslev and Ralf Möller. RACER system description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 701–705. Springer, 2001.

[6] J. Hladik and J. Model. Tableau systems for SHIO and SHIQ. In *Proc. of the 2004 Description Logic Workshop (DL 2004)*. CEUR, 2004. Available from `ceur-ws.org`.

[7] Bernhard Hollunder and Franz Baader. Qualifying number restrictions in concept languages. Technical Report RR-91-03, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Kaiserslautern (Germany), 1991. An abridged version appeared in *Proc. of the 2nd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'91)*.

[8] I. Horrocks and S. Tobies. Reasoning with axioms: Theory and practice. In A. G. Cohn, F. Giunchiglia, and B. Selman, editors, *Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR-00)*. Morgan Kaufmann, Los Altos, 2000.

[9] Ian Horrocks and Peter F. Patel-Schneider. FaCT and DLP: Automated reasoning with analytic tableaux and related methods. In *Proc. of the 2nd Int. Conf. on Analytic Tableaux and Related Methods (TABLEAUX'98)*, pages 27–30, 1998.

[10] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *J. of Web Semantics*, 1(1):7–26, 2003.

[11] Ian Horrocks and Ulrike Sattler. Ontology reasoning in the $\mathcal{SHOQ}$(D) description logic. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 199–204, 2001.

[12] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for expressive description logics. In Harald Ganzinger, David McAllester, and Andrei Voronkov, editors, *Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes in Artificial Intelligence, pages 161–180. Springer, 1999.

[13] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Reasoning with individuals for the description logic $\mathcal{SHIQ}$. In David McAllester, editor, *Proc. of the 17th Int. Conf. on Automated Deduction (CADE 2000)*, volume 1831 of *Lecture Notes in Computer Science*, pages 482–496. Springer, 2000.

[14] Deborah L. McGuinness and Jon R. Wright. An industrial strength description logic-based configuration platform. *IEEE Intelligent Systems*, pages 69–77, 1998.

[15] Leszek Pacholski, Wieslaw Szwast, and Lidia Tendera. Complexity of two-variable logic with counting. In *Proc. of the 12th IEEE Symp. on Logic in Computer Science (LICS'97)*, pages 318–327. IEEE Computer Society Press, 1997.

[16] Jeff Pan and Ian Horrocks. Web ontology reasoning with datatype groups. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *Proc. of the 2003 International Semantic Web Conference (ISWC 2003)*, number 2870 in Lecture Notes in Computer Science, pages 47–63. Springer, 2003.

[17] Pellet OWL reasoner. Maryland Information and Network Dynamics Lab, 2003. `http://www.mindswap.org/2003/pellet/index.shtml`.

[18] Andrea Schaerf. Reasoning with individuals in concept languages. *Data and Knowledge Engineering*, 13(2):141–176, 1994.

[19] Stephan Tobies. The complexity of reasoning with cardinality restrictions and nominals in expressive description logics. *J. of Artificial Intelligence Research*, 12:199–217, 2000.

[20] Stephan Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, LuFG Theoretical Computer Science, RWTH-Aachen, Germany, 2001.

[21] Dmitry Tsarkov and Ian Horrocks. Efficient reasoning with range and domain constraints. In *Proc. of the 2004 Description Logic Workshop (DL 2004)*, pages 41–50, 2004.

[22] Moshe Y. Vardi. Why is modal logic so robustly decidable. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 31, pages 149–184. American Mathematical Society, 1997.