A Tableau Decision Procedure for $\mathcal{SHOIQ}^{\dagger}$

Ian Horrocks and Ulrike Sattler
School of Computer Science
University of Manchester, UK

July 23, 2007

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.

Keywords: Description Logic, Decision Procedures

1. Introduction

Description Logics (DLs) are a family of logic based knowledge representation formalisms (Baader et al., 2003). Although they have a range of applications (e.g., configuration (McGuinness and Wright, 1998), and information integration (Calvanese et al., 1998)), they are perhaps best known as the basis for widely used ontology languages such as OIL, DAML+OIL and OWL (Horrocks et al., 2003), the last of which is now a World Wide Web Consortium (W3C) recommendation (Bechhofer et al., 2004).

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 (Horrocks et al., 2003).

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 (Horrocks et al., 2003).[2] Standard reasoning problems include the computation

[†] This is an updated version of the paper that appeared in JAR xx(y), and in which the termination proof is slightly flawed. This version extends the definition of the o-rule and clarifies the termination proof.

[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* (Baader and Hanschke, 1991). These can, however, be treated exactly as in $\mathcal{SHOQ}(\mathbf{D})$/$\mathcal{SHOQ}(\mathbf{D}_n)$ (Horrocks and

of the subsumption hierarchy between concepts defined in a given ontology, deciding the satisfiability of all concepts defined in a given ontology, and retrieving all instances of a concept expression. In $\mathcal{SHOIN}$, these standard reasoning problems can all be reduced to the ontology consistency problem, which is known to be decidable: this is a consequence of a reduction of DLs with transitive roles to DLs without such roles (Tobies, 2001) 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 (Pacholski et al., 1997). To the best of our knowledge, however, the algorithm described here is the first "practical" decision procedure for $\mathcal{SHOIN}$, i.e., the first goal-directed procedure that is likely to perform well with realistic ontology derived problems (Tobies, 2001; Horrocks and Sattler, 2001).

In this paper, we present a goal-directed decision procedure for $\mathcal{SHOIQ}$, i.e., $\mathcal{SHOIN}$ extended with *qualified* number restrictions (Hollunder and Baader, 1991). The algorithm extends the well-known tableau algorithm for $\mathcal{SHIQ}$ (Horrocks et al., 1999), which is the basis for several highly successful implementations (Horrocks and Patel-Schneider, 1998; Haarslev and Möller, 2001b; Sirin et al., 2003).

As its name indicates, $\mathcal{SHOIQ}$ extends $\mathcal{SHIQ}$ with ***no**minals*,[3] i.e., concepts with a singleton extension (De Giacomo, 1995). Nominals are a prominent feature of hybrid logics (Blackburn and Seligman, 1995), and can also be viewed as a powerful generalisation of *ABox individuals* (Schaerf, 1994; Baader et al., 2003). A form of nominals was already present in early DL systems such as CLASSIC (Brachman et al., 1991) and CRACK (Bresciani et al., 1995), and they occur naturally in ontologies, e.g., when describing a concept such as EUCountries by enumerating its members, i.e., {Austria, ..., 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.

It has long been recognised that there is a close connection between DLs and propositional modal and dynamic logics (Schild, 1991; De Giacomo and Lenzerini, 1994; Baader et al., 2003). One reason why all these logics enjoy good computational properties, such as being robustly decidable, is that they have some form of tree model property (Vardi, 1997; Grädel, 2001), i.e., if an ontology is consistent, then it has a model whose relational structure forms a tree or can be seen as a tree. This feature is crucial in the design of tableau algorithms, allowing them to search only for tree-like models. More precisely, DL tableau 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

Sattler, 2001; Haarslev and Möller, 2001a; Pan and Horrocks, 2003), so we will not complicate our presentation by considering them here.

[3] For naming conventions of DLs, see the Appendix of (Baader et al., 2003).

tree-shaped completion graphs. For expressive DLs, this restriction is crucial since tableau algorithms for them 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 (Schild, 1991; Baader et al., 2003). On such an input, the $\mathcal{SHIQ}$ tableau 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 (Horrocks et al., 1999). Even when the language includes nominals, but *excludes* one of number restrictions or inverse roles (De Giacomo, 1995; Horrocks and Sattler, 2001; Hladik and Model, 2004), or if nominals are restricted to ABox individuals (Donini et al., 1990; Baader and Hollunder, 1991; Buchheit et al., 1993; Schaerf, 1994; De Giacomo and Lenzerini, 1996; Horrocks et al., 2000), 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 (Tobies, 2000). To see this interaction, consider an ontology containing the following two axioms involving a nominal o and a non-negative integer n:

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

The first statement requires that, in a model of this ontology, every element has a U-edge leading to o; the second statement restricts the number of inverse U-edges going from o to instances of F to at most n. Thus the interaction between the nominal and the number restriction on the inverse of U imposes an upper bound of n on the number of instances of the concept F. If we add further axioms, we might need to consider arbitrarily complex relational structures amongst instances of F. For example, if we add the following axiom, then each instance of F is necessarily R-related to every instance of F, including itself:

$$F \dot{\sqsubseteq} (\geqslant n\, R.F).$$

Similarly, the following axiom would enforce S-cycles over instances of F:

$$F \dot{\sqsubseteq} (\geqslant 1\, S.F) \sqcap (\leqslant 1\, S^{-}.F).$$

Hence a tableau algorithm for $\mathcal{SHOIQ}$ needs to be able to handle arbitrarily complex relational structures, and thus we cannot restrict our attention to completion trees or forests.

Matters are further complicated by the fact that $\mathcal{SHOIQ}$ does not enjoy the finite model property, and hence there are $\mathcal{SHOIQ}$ axioms that enforce the existence of an infinite number of instances of a concept. For example, the concept $\neg N \sqcap \exists P.N$ is satisfiable w.r.t. the following axiom, but only in models with infinitely many instances of N:

$$N \dot{\sqsubseteq} (\leqslant 1\, P^{-}.\top) \sqcap \exists P.N.$$

Now consider an ontology that contains, amongst others, all the above mentioned axioms. The consistency of this ontology then crucially depends on the relations enforced between instances of F and N. For example, the additional axioms

$$\begin{aligned} N &\dot{\sqsubseteq} \exists V.F \quad \text{and} \\ F &\dot{\sqsubseteq} (\leqslant k\, V^{-}.N) \end{aligned}$$

yield an inconsistent ontology since our at most n instances of F cannot play the rôle of V-fillers for infinitely many instances of N when each of them can be the V-filler of at most k instances of N.

Summing up, a tableau algorithm for $\mathcal{SHOIQ}$ needs to be able to handle both arbitrarily complex relational structures and finite tree structures representing infinite 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 tableau 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".[4]

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 $(\leqslant n\, R.C)$ concept in the label of a nominal node, generates (non-deterministically) between 1 and n new nominal nodes, all of which

[4] Franz Baader introduced this term for a similar problem that needs to be overcome in similar algorithms such as those described in (Baader and Hanschke, 1991; Baader and Sattler, 2001).

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 will 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:[5] in case number restrictions are functional (i.e., where n is 1),[6] 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 $\mathcal{SHOQ}$. The algorithm has, in fact, already been implemented in the well known Fact++ and Pellet systems, and has shown very promising behaviour with realistic ontologies (Tsarkov and Horrocks, 2006; Sirin et al., 2006; Gardiner et al., 2006).

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 a set $\mathbf{R}_+ \subseteq \mathbf{R}$ of transitive role names. The set of *$\mathcal{SHOIQ}$-roles* (or *roles* 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. To avoid considering roles such as R^{--}, we define a function Inv which returns the inverse of a role:

[5] In the sense that the behaviour of the algorithm will depend on the expressive power actually used in the input ontology (Patel-Schneider et al., 1990).

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

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

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 as to 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. Note that the interpretation of S may be transitive even if $\mathsf{Trans}(S, \mathcal{R})$ returns false; this could arise, e.g., if the TBox contains an axiom $\top \sqsubseteq \forall S.(\forall S.\bot)$.

4. A role R is called *simple* w.r.t. $\mathcal{R}$ if $\mathsf{Trans}(S, \mathcal{R}) = \mathrm{false}$ for all $S \sqsubseteq^*_{\mathcal{R}} R$, i.e., if it is neither transitive nor has a transitive sub-role. If a role is non-simple, then it may lead to implicit role relationships as a result of transitive (sub-) role chains. For example, if $S \sqsubseteq^*_{\mathcal{R}} R$ and $\mathsf{Trans}(S, \mathcal{R})$, then $\{\langle x, y\rangle, \langle y, z\rangle\} \subseteq S^{\mathcal{I}}$ implies $\langle x, z\rangle \in R^{\mathcal{I}}$. Counting the number of fillers of a non-simple role is, therefore, rather tricky, and it is known that allowing non-simple roles in number restrictions results in the undecidability of standard reasoning problems (Horrocks et al., 1999).

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 $\mathsf{Trans}(S, \mathcal{R})$.

Definition 2 Let $\mathbf{C}$ be a sets of *concept names* with a subset $\mathbf{C}_o \subseteq \mathbf{C}$ of *nominals*. The set of $\mathcal{SHOIQ}$*-concepts* (or *concepts* for short) is the smallest set such that

1. every concept name $C \in \mathbf{C}$ 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 and $n \in \mathbb{N}$, then $(\leqslant n\, R.C)$ and $(\geqslant n\, R.C)$ are also concepts (called atmost and atleast number restrictions).[7]

[7] Since $\mathcal{SHOIQ}$ extends $\mathcal{SHIN}$, we need to restrict roles in number restrictions to simple ones in order to yield a decidable logic (Horrocks et al., 1999).

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

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

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, $C \mathrel{\dot{\sqsubseteq}} D$ is called a *general concept inclusion* (GCI), and a finite set of GCIs is called a *TBox*. An interpretation $\mathcal{I}$ *satisfies* a GCI $C \mathrel{\dot{\sqsubseteq}} D$ 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 pair $(\mathcal{T}, \mathcal{R})$ is called a $\mathcal{SHOIQ}$ *knowledge base* and is said to be *consistent* if there exists a model of $\mathcal{T}$ and $\mathcal{R}$. A concept D *subsumes* a concept C w.r.t. $(\mathcal{T}, \mathcal{R})$ (written $C \sqsubseteq_{\mathcal{R},\mathcal{T}} D$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds in every model $\mathcal{I}$ of $(\mathcal{T}, \mathcal{R})$. Two concepts C, D are *equivalent* w.r.t. $(\mathcal{T}, \mathcal{R})$, (written $C \equiv_{\mathcal{R},\mathcal{T}} D$) if they are mutually subsuming w.r.t. $(\mathcal{T}, \mathcal{R})$. For an interpretation $\mathcal{I}$, an individual $x \in \Delta^{\mathcal{I}}$ is called an *instance* of a concept C iff $x \in C^{\mathcal{I}}$.

Note that the above definition of a $\mathcal{SHOIQ}$ knowledge base does not include an *ABox*, i.e., a set of assertions of the form $a : C$ or $(a, b) : R$, for a and b nominals, C a concept and R a role, where an interpretation $\mathcal{I}$ satisfies $a : C$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$, and satisfies $(a, b) : R$ if $\langle a^{\mathcal{I}}, b^{\mathcal{I}}\rangle \in R^{\mathcal{I}}$. This is because, in the presence of nominals, such assertions can be transformed into semantically equivalent TBox axioms: $a : C$ is equivalent to $a \mathrel{\dot{\sqsubseteq}} C$, and $(a, b) : R$ is equivalent to $a \mathrel{\dot{\sqsubseteq}} \exists R.b$. An ABox $\mathcal{A}$ is *consistent* with respect to a knowledge base $(\mathcal{T}, \mathcal{R})$ (i.e., there exists a model of $\mathcal{A}$, $\mathcal{T}$ and $\mathcal{R}$) if and only if $(\mathcal{T} \cup \{a \mathrel{\dot{\sqsubseteq}} C \mid a : C \in \mathcal{A}\} \cup \{a \mathrel{\dot{\sqsubseteq}} \exists R.b \mid (a, b) : R \in \mathcal{A}\},\ \mathcal{R})$ is consistent (Baader et al., 2003).

As for other propositionally closed logics, subsumption and satisfiability can be reduced to each other. We can also reduce satisfiability of concepts w.r.t. a knowledge base to knowledge base consistency: C is satisfiable w.r.t. $(\mathcal{T}, \mathcal{R})$ if and only if $(\mathcal{T} \cup \{o \mathrel{\dot{\sqsubseteq}} C\}, \mathcal{R})$ is consistent, for o a nominal that does not occur in C or $\mathcal{T}$. As a consequence, in the remainder of this paper and without loss of generality, we will restrict our attention to knowledge base consistency.

Finally, we did not choose to make a *unique name assumption*, i.e., two nominals might refer to the same individual. We can, however, use an axiom $o_i \dot{\sqsubseteq} \neg o_j$ to assert that nominals o_i and o_j do *not* refer to the same individual, and we can use a set of such axioms for each $o_i \neq o_j$ occurring in $\mathcal{T}$ to mimic the effect of the unique name assumption. Similarly, the inference algorithm presented below can easily be adapted to the unique name case by a suitable initialisation of the inequality relation $\dot{\neq}$.

3. A Tableau for $\mathcal{SHOIQ}$

For ease of presentation, 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, (Horrocks et al., 2000). For a concept C, we use $\mathsf{nnf}(C)$ to denote the NNF of C, we use $\dot{\neg}C$ for the NNF of $\neg C$, and we use $\mathsf{sub}(C)$ to denote the set of all sub-concepts of C (including C). For a knowledge base $(\mathcal{T}, \mathcal{R})$, we define the set of "relevant sub-concepts" $\mathsf{cl}(\mathcal{T}, \mathcal{R})$ as follows:

$$\begin{aligned}
\mathsf{cl}(\mathcal{T}, \mathcal{R}) &:= \bigcup_{C \dot{\sqsubseteq} D \in \mathcal{T}} \mathsf{cl}(\mathsf{nnf}(\neg C \sqcup D), \mathcal{R}) \quad \text{where} \\
\mathsf{cl}(E, \mathcal{R}) &:= \mathsf{sub}(E) \cup \{\dot{\neg}C \mid C \in \mathsf{sub}(E)\} \cup \\
&\quad \{\forall S.C \mid \forall R.C \in \mathsf{sub}(E) \text{ or } \dot{\neg}\forall R.C \in \mathsf{sub}(E) \\
&\qquad \text{and } S \text{ occurs in } \mathcal{T} \text{ or } \mathcal{R}\}
\end{aligned}$$

When $\mathcal{R}$ is clear from the context, we use $\mathsf{cl}(\mathcal{T})$ instead of $\mathsf{cl}(\mathcal{T}, \mathcal{R})$. Note that the set of relevant sub-concepts is closely related to the Fischer-Ladner closure (Fischer and Ladner, 1979; Baader et al., 2003), and extends the notion of sub-concepts to take into account the fact that the tableau expansion rules may introduce some new concepts that do not occur syntactically in the input knowledge base.

Next, we define a tableau as a useful abstraction of a model to make the correctness proof of our algorithm more readable. Intuitively, a tableau contains all the relevant information from a model, with the possible exception of transitive roles and their super-roles. That is, in a tableau, we represent the fact that t is an R-successor of s as $\langle s, t\rangle \in \mathcal{E}(R)$, but we do not impose that, if $\mathsf{Trans}(R)$ and $\langle s, t\rangle$, $\langle t, u\rangle \in \mathcal{E}(R)$, then $\langle t, u\rangle \in \mathcal{E}(R)$. To compensate for these possibly missing edges, (P6) ensures that value restrictions on (super-roles of) transitive roles are "pushed" correctly over relevant edges. Compared with a tableau for $\mathcal{SHIQ}$, the only difference is the addition of (P12) and (P13) which ensure that nominals are handled correctly. Please note that, like models, tableaux may be infinite, and for some knowledge bases may even be necessarily infinite.

Definition 3 If $(\mathcal{T}, \mathcal{R})$ is a $\mathcal{SHOIQ}$ knowledge base, and $\mathbf{R}_{\mathcal{T},\mathcal{R}}$ the set of roles occurring in $\mathcal{T}$ or $\mathcal{R}$ together with their inverses, a *tableau* T *for* $(\mathcal{T}, \mathcal{R})$ is defined to be a triple $(\mathbf{S}, \mathcal{L}, \mathcal{E})$ such that: $\mathbf{S}$ is a set of individuals, $\mathcal{L} : \mathbf{S} \rightarrow 2^{\mathsf{cl}(\mathcal{T})}$ maps each individual to a set of concepts which is a subset of $\mathsf{cl}(\mathcal{T})$, $\mathcal{E} : \mathbf{R}_{\mathcal{T},\mathcal{R}} \rightarrow 2^{\mathbf{S}\times\mathbf{S}}$ maps each role in $\mathbf{R}_{\mathcal{T},\mathcal{R}}$ to a set of pairs of individuals. For all $s, t \in \mathbf{S}$, $C, C_1, C_2 \in \mathsf{cl}(\mathcal{T})$, and $R, S \in \mathbf{R}_{\mathcal{T},\mathcal{R}}$, it holds that:

(P0) if $C \dot{\sqsubseteq} D \in \mathcal{T}$, then $\mathsf{nnf}(\neg C \sqcup D) \in \mathcal{L}(s)$,

(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 $(\geqslant n\, S.C) \in \mathcal{L}(s)$, then $\sharp\{t \in \mathbf{S} \mid \langle s, t\rangle \in \mathcal{E}(S) \text{ and } C \in \mathcal{L}(t)\} \geqslant n$,

(P8) if $(\leqslant n\, S.C) \in \mathcal{L}(s)$, then $\sharp\{t \in \mathbf{S} \mid \langle s, t\rangle \in \mathcal{E}(S) \text{ and } C \in \mathcal{L}(t)\} \leqslant n$,

(P9) if $(\leqslant n\, S.C) \in \mathcal{L}(s)$ 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 $\langle t, s\rangle \in \mathcal{E}(\mathsf{Inv}(R))$,

(P12) if $o \in \mathcal{L}(s) \cap \mathcal{L}(t)$ for some $o \in \mathbf{C}_o$, then $s = t$, and

(P13) for each $o \in \mathbf{C}_o$ occurring in $\mathcal{T}$, there is some $s \in \mathbf{S}$ with $o \in \mathcal{L}(s)$.

Please note that, in (P1), we use $\neg C$ and not $\dot{\neg} C$: this suffices for the following lemma and makes the proof of Lemma 7 easier.

Lemma 4 A $\mathcal{SHOIQ}$ knowledge base $(\mathcal{T}, \mathcal{R})$ is consistent iff there exists a tableau for $(\mathcal{T}, \mathcal{R})$.

Proof (sketch): This proof is analogous to the proof found in (Horrocks et al., 1999). As for $\mathcal{SHIQ}$, 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 $s \in C^{\mathcal{I}}$. Due to the restriction to simple roles in

number restrictions, these additional roles-successorships do not lead to any violation of atmost number restrictions. (P0) ensures that $\mathcal{I}$ is indeed a model of $\mathcal{T}$. (P6) ensures, as for $\mathcal{SHIQ}$, that the additional role-successorships can be added without violating any value restrictions. (P12) and (P13) ensure that nominals are indeed interpreted as singletons.

For the converse, we can easily view any model as a tableau. □

4. A Tableau Algorithm for $\mathcal{SHOIQ}$

From Lemma 4, an algorithm which constructs a (finite) representation of a (possibly infinite) tableau for a $\mathcal{SHOIQ}$ knowledge base can be used as a decision procedure for consistency of $\mathcal{SHOIQ}$ knowledge bases, and thus to decide satisfiability and subsumption of $\mathcal{SHOIQ}$-concepts w.r.t. knowledge bases. 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 $(\mathcal{T}, \mathcal{R})$ be a $\mathcal{SHOIQ}$ knowledge base. A *completion graph* for $(\mathcal{T}, \mathcal{R})$ is a directed graph $\mathbf{G} = (V, E, \mathcal{L}, \neq)$ where each node $x \in V$ is labelled with a set

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

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 $\mathcal{T}$ 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 $R' \sqsubseteq^{*} R$, $R' \in \mathcal{L}(\langle x, y\rangle)$; x is called an R-predecessor of y if y is an $\mathsf{Inv}(R)$-successor of x. A node y is called a *neighbour* (R-*neighbour*) of a node x if y is either a successor (R-successor) or a predecessor (R-predecessor) of x.

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 $A \in \mathbf{C}$ 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 $o \in \mathbf{C}_o$, 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 $\mathcal{T}$, then the tableau algorithm starts with the completion graph $\mathbf{G} = (\{r_1 \ldots, r_\ell\}, \emptyset, \mathcal{L}, \emptyset)$ with $\mathcal{L}(r_i) = \{o_i\}$ for $1 \leq i \leq \ell$. For $0 \leq i \leq \ell$, each r_i is called an *initial node*. $\mathbf{G}$ 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 $o \in \mathbf{C}_o$ is said to be *new in* $\mathbf{G}$ if no node in $\mathbf{G}$ has o in its label.
Comment: like ABox individuals, nominal nodes can be arbitrarily interconnected. In contrast, blockable nodes are only found in tree-like structures rooted in nominal nodes; 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 must 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.
Comment: blocking is defined as for $\mathcal{SHIQ}$, with the only difference being 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 $\leqslant$-, $\leqslant_o$- 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 are generated for nominal nodes.

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 $\mathbf{G} = (V, E, \mathcal{L}, \not\approx)$ 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.

Comment: the tree-like structure of the blockable parts of $\mathbf{G}$ ensure that pruning only removes sub-trees. If blockable parts of $\mathbf{G}$ did not have a tree-like structure, e.g., if they were cyclical, then pruning might remove y's predecessor, the node x into which y is being merged, and/or the node on whose label the rule responsible for the merge is operating.

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 $\mathbf{G} = (V, E, \mathcal{L}, \not\approx)$ 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) Recall that $o_1, \ldots, o_\ell$ are all the nominals occurring in the input TBox $\mathcal{T}$. 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 $\leqslant_o$- 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).
3. all other rules are applied with a lower priority.

Comment: this strategy is necessary for termination, and in particular to ensure that the blockable parts of the completion graph remain tree-shaped. In practice, more elaborate strategies may be used for efficiency reasons (Horrocks and Patel-Schneider, 1999; Tsarkov and Horrocks, 2005).

We will now describe the tableau expansion rules and discuss the intuition behind their definitions. Please note that the definition of *successor* and *predecessor* and the way in which the expansion rules introduce new nodes make sure that these relations reflect the origin of a node.

The rules in Figure 1 are very similar to the expansion rules for $\mathcal{SHIQ}$ (Horrocks et al., 1999). The $\dot{\sqsubseteq}$-rule reflects the semantics of GCIs and corresponds to (P0) of Definition 3 (of a $\mathcal{SHOIQ}$ tableau); the $\sqcap$-, $\sqcup$-, $\exists$- and $\forall$-rules directly reflect the semantics of the relevant concept constructors and correspond to (P2)–(P5). A new feature of the $\exists$-rule with respect to its $\mathcal{SHIQ}$ counterpart is that only safe S-neighbours are considered. This is because a nominal node might have an unsafe S-neighbour (i.e., a blocked S-predecessor) that would not correspond to an individual in the tableau and so would not satisfy (P5).

The $\forall_+$-rule corresponds to (P6) of Definition 3, and ensures that the effects of universal restrictions are propagated as necessary in the presence of non-simple roles.

The $\geqslant$-, $\leqslant$- and *ch*-rules correspond to (P7)–(P9) of Definition 3. For a concept $(\geqslant\ n\,S.C) \in \mathcal{L}(x)$, if x does not already have n suitable S-neighbours (note that, as for the $\exists$-rule, only safe S-neighbours are considered), then the $\geqslant$-rule generates them. To prevent the $\leqslant$-rule from merging the new nodes, it sets $y_i \not\doteq y_j$ for each $1 \leq i < j \leq n$. Conversely, if $(\leqslant\ n\,S.C) \in \mathcal{L}(x)$ and x has more than n S-neighbours that are labelled with C, then the $\leqslant$-rule chooses two of them that are not in $\not\doteq$ and merges them. Part (2) of the definition of a clash takes care of the situation where the $\not\doteq$ relation makes it impossible to merge any two S-neighbours of x, while the *ch*-rule ensures that all S-neighbours of x are labelled with either C or $\dot{\neg}C$.

$\dot{\sqsubseteq}$-rule: if $C \mathrel{\dot{\sqsubseteq}} D \in \mathcal{T}$, and $\mathsf{nnf}(\neg C \sqcup D) \notin \mathcal{L}(x)$
then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{\mathsf{nnf}(\neg C \sqcup D)\}$

$\sqcap$-rule: if $C_1 \sqcap C_2 \in \mathcal{L}(x)$, and $\{C_1, C_2\} \not\subseteq \mathcal{L}(x)$
then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1, C_2\}$

$\sqcup$-rule: if $C_1 \sqcup C_2 \in \mathcal{L}(x)$, and $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$
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. $\forall S.C \in \mathcal{L}(x)$, 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. $\forall S.C \in \mathcal{L}(x)$, 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 $\mathcal{L}(y) = \mathcal{L}(y) \cup \{\forall R.C\}$

$\geqslant$-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. $(\leqslant n\, S.C) \in \mathcal{L}(x)$, and
2. $\sharp S^{\mathbf{G}}(x, C) > n$ and there are two S-neighbours y, z of x with
$C \in \mathcal{L}(y) \cap \mathcal{L}(z)$, and not $y \neq z$
then 1. if y is a nominal node, then $\mathsf{Merge}(z, y)$
2. else if z is a nominal node or an ancestor of y, then $\mathsf{Merge}(y, z)$
3. else $\mathsf{Merge}(z, y)$

ch-rule: if 1. $(\leqslant n\, S.C) \in \mathcal{L}(x)$, 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 $E \in \{C, \dot{\neg} C\}$

Figure 1. The tableau expansion rules for $\mathcal{SHIQ}$.

Note that the generating rules are not applicable to blocked nodes. This prevents the algorithm attempting to generate an infinite completion graph, e.g., when $\{\top \mathrel{\dot{\sqsubseteq}} \exists S.\top\} \subseteq \mathcal{T}$. The other rules are applicable to all nodes,

o-rule:	if	for some $o \in \mathbf{C}_o$ there are 2 nodes x, y with $o \in \mathcal{L}(x) \cap \mathcal{L}(y)$ and not $x \neq y$,
	then	1. if y is an initial node, then $\mathsf{Merge}(x, y)$ 2. else $\mathsf{Merge}(y, x)$.
NN-rule:	if	1. $(\leqslant n\, S.C) \in \mathcal{L}(x)$, x is a nominal node, and there is a blockable S-predecessor y of x such that $C \in \mathcal{L}(y)$, and 2. there is no $(\leqslant \ell\, S.C) \in \mathcal{L}(x)$ such that $\ell \leqslant n$ and there exist ℓ nominal S-neighbours $z_1, \ldots, z_\ell$ of x with $C \in \mathcal{L}(z_i)$ and $z_i \neq z_j$ for all $1 \leq i < j \leq \ell$.
	then	1. guess m with $1 \leqslant m \leqslant n$ and set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{(\leqslant m\, S.C)\}$ 2. create m new nodes $z_1, \ldots, z_m$ with $\mathcal{L}(\langle x, z_i \rangle) = \{S\}$, $\mathcal{L}(z_i) = \{C, o_i\}$ with $o_i \in \mathbf{C}_o$ new in $\mathbf{G}$, and $z_i \neq z_j$ for $1 \leq i < j \leq m$.
$\leqslant_o$-rule:	if	1. $(\leqslant m\, S.C) \in \mathcal{L}(x)$, x is a nominal node, and there is a blockable S-neighbour y of x such that $C \in \mathcal{L}(y)$, 2. 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 $1 \leq i < j \leq m$, and 3. there is a nominal S-neighbour z of x with $C \in \mathcal{L}(z)$, and not $y \neq z$
	then	$\mathsf{Merge}(y, z)$

Figure 2. The three new expansion rules for $\mathcal{SHOIQ}$.

even blocked nodes. This is necessary because, in the presence of inverse roles, applying expansion rules to blocked nodes could result in information being added to the labels of unblocked nodes, possibly resulting in a clash or in previously blocked nodes becoming unblocked (Horrocks et al., 1999).

The rules in Figure 2 deal with nominals. The o-rule corresponds to (P12) of Definition 3, and ensures that any two nodes having the same nominal in their label are immediately merged. The combination of the *NN*-rule and the $\leqslant_o$-rule ensure that the blockable parts of the completion graph are always tree shaped (which is necessary in order to ensure that pruning can only remove blockable sub-trees), and prevent the repeated generation and merging of new nominal nodes. The blockable part of the graph could become non-tree shaped if a blockable predecessor of a nominal node were merged with another blockable neighbour. In order to prevent this situation arising, if a nominal node x has a blockable S-predecessor y such that $C \in \mathcal{L}(y)$ and $(\leqslant n\, S.C) \in \mathcal{L}(x)$, then the *NN*-rule guesses the exact number $m \leqslant n$ of S-neighbours of x with C in their label. It then generates m new nominal

S-neighbours $z_1, \ldots, z_m$ of x, with $\mathcal{L}(z_i) = \{C, o_i\}$, where $o_i \in \mathbf{C}_o$ is a new nominal, i.e., a nominal not already occurring in $\mathbf{G}$. Finally, to prevent these new nodes from being merged, it additionally sets $z_i \neq z_j$ for each $1 \leq i < j \leq m$. The $\leqslant_o$-rule ensures that each blockable S-neighbour of x with C in its label is merged with one of the new nominal S-neighbours of x. The $\leqslant_o$-rule is applied with a higher priority than the $\leqslant$-rule, ensuring that blockable neighbours of x cannot be merged with each other before being merged into the new nominal nodes. An extended example of the action of these rules, and the need for them, will be presented in Section 4.2.

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 $\mathbf{G}$ in such a way that they yield a complete, clash-free completion graph, then the algorithm returns "$(\mathcal{T}, \mathcal{R})$ is consistent", and "$(\mathcal{T}, \mathcal{R})$ is inconsistent" otherwise.

4.2. Example application of the algorithm

We consider three examples, starting with a rather easy one. First, consider the TBox

$$\begin{aligned}\mathcal{T} = \{&A \mathrel{\dot{\sqsubseteq}} \exists R.(A \sqcap \exists R.A),\\ &A \mathrel{\dot{\sqsubseteq}} o\\ &o \mathrel{\dot{\sqsubseteq}} A\}.\end{aligned}$$

Our tableau algorithm starts with a completion graph consisting of a single node, r, with $\mathcal{L}(r) = \{o\}$. After three applications of the $\dot{\sqsubseteq}$-rule, we obtain a completion graph with

$$\mathcal{L}(r) = \{o, \neg A \sqcup \exists R.(A \sqcap \exists R.A), \neg A \sqcup o, \neg o \sqcup A\}.$$

The only way to apply the $\sqcup$-rule without causing clashes adds A and $\exists R.(A \sqcap \exists R.A)$ to $\mathcal{L}(r)$. Next, we can choose to apply the $\exists$-rule, the $\sqcap$-rule and the $\exists$-rule again, which yields two new nodes x_0 and x_1, where x_0 is an R-successor of r and x_1 is an R-successor of x_0, with

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

To these nodes, we can apply the $\dot{\sqsubseteq}$-, $\sqcap$-, and the $\sqcup$-rule, and we are free to choose where we apply them first. If we apply these rules to x_0 first, and in such a way as to avoid clashes, the result is that

$$\mathcal{L}(x_0) \supseteq \{A \sqcap \exists R.A, A, \exists R.A, o, \exists R.(A \sqcap \exists R.A)\}.$$

Following our rule application strategy, we now need to apply the o-rule. This rule merges x_0 into r, makes r an R-neighbour of itself, and removes x_1

through pruning. As a result, we have a clash-free and complete completion graph, and thus our algorithm returns "satisfiable". Indeed, it corresponds to a model $\mathcal{I}$ with a single element r, $R^{\mathcal{I}} = \{(r, r)\}$, and $A^{\mathcal{I}} = o^{\mathcal{I}} = \{r\}$. Please note that, for termination, it was crucial that we removed x_1: otherwise, we could have generated an R-successor of x_1 before adding o to $\mathcal{L}(x_1)$ and merging x_1 with r, and we could have repeated this "generate-and-merge" action without ever terminating.

Alternatively, we could have chosen to apply the $\dot{\sqsubseteq}$-rule to x_i only w.r.t. $A \dot{\sqsubseteq} \exists R.(A \sqcap \exists R.A)$, along with applications of the $\exists$-, $\sqcap$- and $\sqcup$-rules such that clashes are avoided. This will lead to the generation of one or more sequences of R-successor nodes that will eventually give rise to blocking conditions. At this point, (clash avoiding) applications of the $\dot{\sqsubseteq}$- and $\sqcup$-rules will lead to the addition of o to the label of some node, followed by an application of the o-rule. Pruning will ensure that this does not lead to any further applications of the $\exists$-rule, and further (clash avoiding) applications of the $\dot{\sqsubseteq}$-, $\sqcup$- and o-rules will eventually cause all the nodes to collapse into a clash-free and complete completion graph consisting, as before, of a single node that is an R-neighbour of itself.

Finally, note that the eager application of the o-rule is necessary for termination in this latter case because adding o to a node label might prevent blocking (there can be no nominal nodes on the path between blocking and blocked nodes), and thus would allow us to continue with applications of the $\exists$-rule.

Our second example is motivated by the observation that the above mentioned "generate-and-merge" problem is very similar to the well-known "yo-yo" effect,[8] but that it was solved in a different manner: we use pruning to prevent "yo-yo"-ing whereas, in other tableau algorithms such as the one described in (Baader and Sattler, 2001), a strategy of rule applications sufficed. Since pruning means discarding work which may subsequently need to be repeated, we present the next example to demonstrate that it is indeed necessary. Consider the following TBox, which is similar to the previous one, but with the difference that, due to the universal value restriction, we do not find the nominal o in a node's label until after we have generated an R-successor, by which time it is already too late to prevent a repetition of the generate-and-merge cycle.

$$\begin{aligned}\mathcal{T} = \{&A \dot{\sqsubseteq} \exists R.(A \sqcap \exists R.(A \sqcap \forall R^{-}.o)),\\ &o \dot{\sqsubseteq} A\}.\end{aligned}$$

Similarly to the first example, we start with a single node r with $\mathcal{L}(r) = \{o\}$ and, after the application of a few rules, we have expanded our completion

[8] For example, this effect led to the introduction of "fork elimination" in (Baader and Hanschke, 1991) and is described in detail in (Baader and Sattler, 2001).

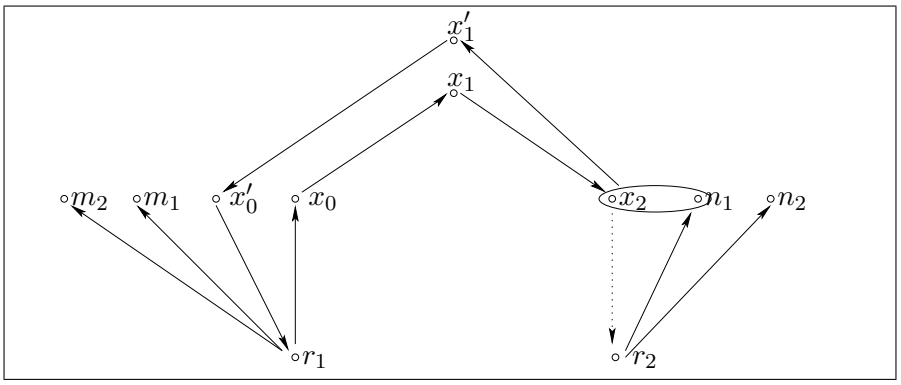

Figure 3. Completion graph of the third example.

graph to contain a chain r, x_0, and x_1 of R-successors with

$$\begin{aligned}\mathcal{L}(r) &\supseteq \{o, A, \exists R.(A \sqcap \exists R.(A \sqcap \forall R^{-}.o))\},\\ \mathcal{L}(x_0) &\supseteq \{A, \exists R.(A \sqcap \forall R^{-}.o)\},\\ \mathcal{L}(x_1) &\supseteq \{A, \forall R^{-}.o\}.\end{aligned}$$

Application of the $\forall$-rule to x_1 adds o to $\mathcal{L}(x_0)$, and thus leads immediately to the o-rule merging x_0 into r and removing x_1. Please note that not removing x_1 would potentially lead to non-termination: we could apply the $\dot{\sqsubseteq}$- and the $\sqcup$-rule to x_1, and thus find $\exists R.(A \sqcap \forall R^{-}.o)$ in $\mathcal{L}(x_1)$. Hence we would apply the $\exists$-rule again, create an R-successor x_2 of x_1 with $(A \sqcap \forall R^{-}.o) \in \mathcal{L}(x_2)$, and could continue this generation and merging forever.

Our third example is presented to demonstrate that the *NN*-rule is indeed needed. Assume we want to decide the consistency of the following TBox:

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

We start the algorithm with two nodes, say r_1 and r_2, with $o_i \in \mathcal{L}(r_i)$. After a few applications of the $\dot{\sqsubseteq}$- and the $\sqcap$-rule, we find

$$\begin{aligned}\mathcal{L}(r_1) &\supseteq \{o_1, (\leqslant 2\, R_1^{-}.A), \forall R_1^{-}.\exists S_1.\exists S_2^{-}.(A \sqcap \exists R_2.o_2), \exists R_1^{-}.A, \},\\ \mathcal{L}(r_2) &\supseteq \{o_2, (\leqslant 2\, R_2^{-}.A), \forall R_2^{-}.\exists S_2.\exists S_1^{-}.(A \sqcap \exists R_1.o_1)\}.\end{aligned}$$

Next, we apply the $\exists$-rule to $\exists R_1^{-}.A \in \mathcal{L}(r_1)$. This creates an R_1^{-}-successor x_0 of r_1, and we can thus apply the $\forall$-rule to $\forall R_1^{-}.\exists S_1.\exists S_2^{-}.(A \sqcap \exists R_2.o_2) \in \mathcal{L}(r_1)$. Next, we can apply the $\exists$-rule three more times, 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

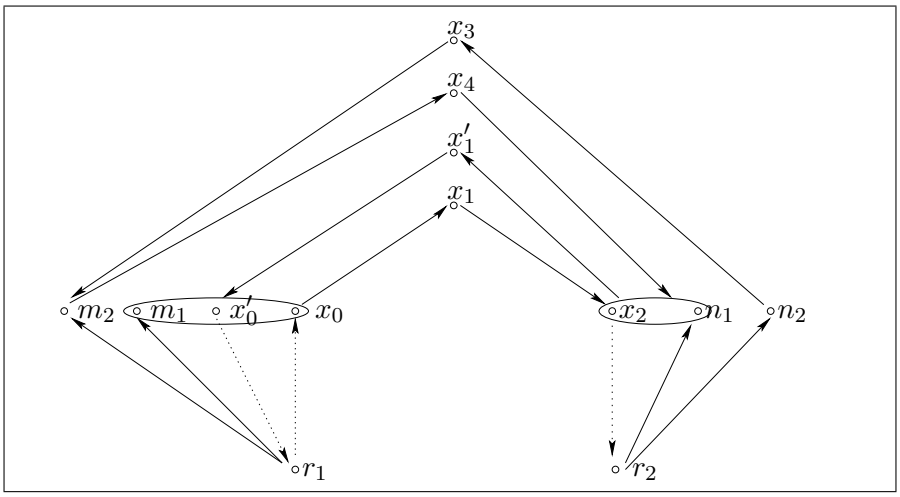

Figure 4. Final completion graph of the third example.

and has highest priority. 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 apply the $\leqslant_o$-rule to r_2 and choose to merge x_2 into n_1. Figure 3 shows the completion graph in a slightly later stage, with arrows that indicate successorship rather than whether a role name or an inverse role labels an edge, and where an edge is dotted to indicate that it has been removed during merging.

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 $\mathcal{L}(n_1)$. Three more applications 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_1^--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 $m_1 \neq m_2$; the resulting completion graph is shown in Figure 3. We apply the $\leqslant_o$-rule to r_1 and choose to merge both x_0' and x_0 into m_1.[9] Please note that the $\leqslant_o$-rule makes us indeed merge the blockable nodes x_0 and x_0' into nominal nodes.

After this, we can apply the $\forall$-rule to add $\exists S_1.\exists S_2^-.(A \sqcap \exists R_2.o_2)$ to $\mathcal{L}(m_2)$ and $\exists S_2.\exists S_1^-.(A \sqcap \exists R_1.o_1)$ to $\mathcal{L}(n_2)$. This yields two more chains between r_1 and r_2, but then the tableau algorithm stops with a complete and clash-free completion graph. This graph is shown in Figure 4, where the successors of x_3 and x_4 are not shown explicitly, and we assume that the $\leqslant_o$-rule-rule has merged x_3's successor into m_2 and x_4's successor into n_1.

[9] Moreover, pruning causes x_1 to be removed, but a similar path will be re-built and merged into r_2 and either n_1 or n_2.

Let us point out three important properties of our algorithm that were crucial for correctness and termination in this last example: (1) As we have illustrated in the introduction, we need to take care of the relations between the finite, arbitrarily complex relational structure around instances of the "original" nominals o_i and the tree-like structure induced by blockable nodes. This is due to the facts that a blocking situation represents an infinite tree, and that we need to take care that atmost restrictions on nominal nodes are satisfied even when considering this infinite tree explicitly. The details will become more clear in the proof of Lemma 7 when we construct a tableau from a complete and clash-free completion graph. We have chosen the *NN*-rule as a means to take care of these relations. (2) We can apply the *NN*-rule only to a nominal node r and some $(\leqslant\, n\, R.C) \in \mathcal{L}(r)$ if there is a blockable node x that has r as its R^-*-successor*. Hence we could only apply it to r_1 after we merged the blockable node y into it. Also, the resulting new nominal nodes will never be pruned since pruning only removes blockable nodes, and they will never be merged due to the use of $\not\approx$. (3) The newly created nominal nodes n_ℓ and m_j made us merge existing and newly created R_i^--neighbours of r_i immediately into n_ℓ and m_j. The explicit inequalities between these new nominal nodes n_ℓ, m_j and the $\leqslant_o$-rule are crucial to prevent another kind of "yo-yo" effect: without them, when applying the $\leqslant$-rule to $(\leqslant\, 2\, R_i^-.A) \in \mathcal{L}(r_i)$, we could have merged n_2 into n_1 and m_2 into m_1 and—even with some modifications to our algorithm, e.g., to make the blockable sibling node of n_ℓ and m_j a nominal node—the $\forall$-rule would add $\exists S_i.\exists S_j^-.(A \sqcap \exists R_j.o_j)$ to each R_i-neighbour of r_i, thereby continuously causing new paths to be built from r_i to r_j, which we could have merged in a similar way, thus causing non-termination.

4.3. Proof of the algorithm's correctness and termination

Lemma 6 When started with a $\mathcal{SHOIQ}$ knowledge base $(\mathcal{T}, \mathcal{R})$, the tableau algorithm terminates.

Proof: Let

- $m := |\mathsf{cl}(\mathcal{T})|$,
- k the number of roles and their inverses in $\mathcal{T}$ and $\mathcal{R}$,
- n the maximal number in number restrictions occurring in $\mathsf{cl}(\mathcal{T})$
- $\lambda := 2^{2m+k}$, and
- $o_1, \ldots, o_\ell$ be all the nominals occurring in $\mathcal{T}$.

The algorithm constructs a graph that consists of a set of arbitrarily interconnected nominal nodes, and "trees" of blockable nodes with each such tree

rooted 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 five properties, the first of which establishes that the blockable parts of the graph are indeed tree-shaped, the second, third and fourth of which are very similar to those used in the termination proof for $\mathcal{SHIQ}$ given in (Horrocks et al., 1999), and the last of which establishes an upper bound on the number of new nominal nodes generated by the *NN*-rule.

1. The structure among blockable nodes is tree-shaped. More precisely, no blockable node can have more than one predecessor node.

   Let us assume that at some stage during rule application the structure among blockable nodes is indeed tree-shaped but that, after applying one of the rules, there is some blockable node x with predecessors y_1 and y_2 such that $y_1 \neq y_2$. This can only happen as a result of an application of a shrinking rule: the generating rules always generate new "leaf" nodes, and the remaining rules do not change the structure of the graph. In fact, it is easy to see that this can only happen if y_1 and y_2 each have blockable successors, one of which is x and one of which is, say, x', and an application of the $\leqslant$-rule causes x' to be merged into x; this in turn implies the existence of some node z and role R such that both x and x' are R-neighbours of z $(\leqslant n\, R.C) \in \mathcal{L}(z)$, and $C \in \mathcal{L}(x) \cap \mathcal{L}(x')$. Moreover, z must be a

   - nominal node: otherwise the structure among blockable nodes is already non-tree-shaped; and
   - z must be a successor of at least one of x or x': otherwise, either $y_1 = y_2 = z$ or the structure among blockable nodes is already non-tree-shaped since x or x' has two different predecessors.

   As a consequence, there must be some $m \leqslant n$ such that $(\leqslant m\, R.C) \in \mathcal{L}(z)$ and z has m nominal R-neighbours $z_1, \ldots, z_m$ with $C \in \mathcal{L}(z_i)$ and $z_i \not\doteq z_j$ for all $1 \leq i < j \leq m$: otherwise the *NN*-rule would be applicable to z and, according to our strategy would be applied *before* the $\leqslant$-rule could merge x and x'. Next, the $\leqslant_o$-rule would be applicable with highest priority and would merge both x and x' in one or other of $z_1, \ldots, z_m$. Thus, the $\leqslant_o$-rule together with the high priority of the *NN*-rule and the $\leqslant_o$-rule ensure that the tree shape among blockable is preserved.

2. 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.

3. 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.

   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). Since pruning only removes blockable nodes, and since these form a tree-structure (as we have shown in property 1 above), we remove neither x nor any of its predecessors. 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 $(\leqslant m\, S.C)$ remains 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 blockable successors. As there are at most m concepts in $\mathcal{L}(x)$, a node can have at most mn blockable successors.

4. If a node x has a blockable successor y, then y was added through the application of a generating rule to x. That is, blockable nodes are not inherited.

   This is a direct consequence of the usage of pruning in the definition of merging.

(3+4) together imply that, over time, each node can have at most mn blockable successors. That is, at any time during a run of the algorithm, for each node x, there are at most mn nodes that have, at some point, been a blockable successor of x.

5. As for $\mathcal{SHIQ}$ (Horrocks et al., 1999), 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, $\mathcal{L}(x) \subseteq \mathsf{cl}(\mathcal{T})$ and thus does not contain any nominals, neither those contained in the input nor those added later by the *NN*-rule.

6. Let us use *root node* for those nominal nodes that are initial nodes or that are generated by the *NN*-rule. The number of root nodes generated is bounded by $\mathcal{O}(\ell(mn)^{\lambda})$.

   Firstly, we observe that new nominals are only introduced by the *NN*-rule, and that the *NN*-rule is only 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 a nominal node; otherwise, it is not possible that a blockable node has a nominal node as a successor. The only nominals that can be added to the label of a blockable node are those that occur in $\mathsf{cl}(\mathcal{T})$, i.e., $o_1, \ldots, o_\ell$, and such an addition is the only way we can have nominal nodes that are not root nodes.

   Secondly, we observe that root nodes are never "indirectly" removed through pruning: a root node y can be merged into another node z, but only if z is a root node, and then z inherits y's root node neighbours.

   Now let us consider the blockable node x to whose label one of the o_i was added: by definition, x is a level 0 node, and the o-rule—which is applied with top priority—will ensure that x is immediately merged with a root node, i.e., an existing level 0 node r_i having the same nominal in its label.

   As a consequence of this merging of x with r_i, two things might happen: (a) the *NN*-rule might become applicable to r_i, and (b) the predecessor x' of x might be merged into a neighbour of r_i by the $\leqslant_o$-rule—please note that, due to the pruning part of merging, this cannot happen to a successor of x and, by definition of the $\leqslant_o$-rule, x' is merged into a nominal neighbour of r_i, say n_1. Repeating this argument, it is possible that

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

Concerning (a), as we have seen above, the strategy of rule application implies that the *NN*-rule is applied to r_i before the $\leqslant$- or the $\leqslant_o$-rule, thereby ensuring that any predecessor x' of x is indeed merged into a nominal node. Moreover, when the *NN*-rule has been applied to a concept $(\leqslant\ n\,R.C)$ in the label of a node y, the way in which edges are "inherited" in the definition of merging and the fact that nominal nodes are not pruned ensures that it can never be applied again to $(\leqslant n\,R.C)$ in the label of y or an heir of y.

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

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

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

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

on the number of root nodes that can be generated.

To sum up, there is a bound $\mathcal{O}(\ell(mn)^\lambda)$ on the number of root nodes that can be generated, and this, along with the fact that, over time, they can have only a bounded number of blockable successors and with the usual argument for the tree-shaped, blockable parts of $\mathbf{G}$, implies a bound on the number of blockable nodes that can be generated. Hence any sequence of rule applications must eventually result in $\mathbf{G}$ being complete. □

Lemma 7 If, when started with a $\mathcal{SHOIQ}$ knowledge base $(\mathcal{T}, \mathcal{R})$, the expansion rules can be applied in such a way as to yield a complete and clash-free completion graph, then there exists a tableau for $(\mathcal{T}, \mathcal{R})$.

Proof: Let $\mathbf{G} = (V, E, \mathcal{L}, \not\doteq)$ be a complete and clash-free completion graph yielded by the expansion rules when started with $(\mathcal{T}, \mathcal{R})$. We can obtain a tableau $T = (\mathbf{S}, \mathcal{L}', \mathcal{E})$ for $(\mathcal{T}, \mathcal{R})$ from $\mathbf{G}$ as described below.

First, let us define paths. For a label blocked node $x \in V$, 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, $\langle (x, x) \rangle \in \mathsf{Paths}(\mathbf{G})$, and
- For a path $p \in \mathsf{Paths}(\mathbf{G})$ and a blockable successor y of $\mathsf{Tail}(p)$:
  - if y is not blocked, then $\langle p | (y, y) \rangle \in \mathsf{Paths}(\mathbf{G})$, and
  - if 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 the predecessor of x' in $\mathbf{G}$ is not blocked. Furthermore, the blocking condition implies $\mathcal{L}(x) = \mathcal{L}(x')$.

It is easy to see that, when $\mathbf{G}$ does not contain any blocks, the elements of $\mathsf{Paths}(\mathbf{G})$ are in direct correspondence with the blockable nodes in V: for each blockable node $x \in V$ there is exactly one path $p \in \mathsf{Paths}(\mathbf{G})$ such that $\mathsf{Tail}(p) = \mathsf{Tail}'(p) = x$. When $\mathbf{G}$ contains one or more blocks, it constitutes a finite representation of an infinite tableau: we can think of each blocked node x as being identified with the node $b(x)$ that is blocking x, and $\mathsf{Paths}(\mathbf{G})$ as being obtained by repeatedly "unravelling" the resulting cycles. This is made possible by the fact that both x and $b(x)$ must occur in the same blockable tree-shaped part of the graph.

The reason for using pairs of nodes in a path is that there might be a node z with two successors x and y, both of which are blocked by the same node, i.e., $b(x) = b(y)$. If paths only mentioned the blocking node, then the paths corresponding to the two successors would be the same, and (P7) of Definition 3 might not be satisfied in the resulting tableau (e.g., if x and y are both S-successors of z, with $C \in \mathcal{L}(x)$, $C \in \mathcal{L}(y)$ and $(\geqslant 2\, S.C) \in \mathcal{L}(z)$).

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.

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

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

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

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 $(\mathcal{T}, \mathcal{R})$.

It suffices to prove that T satisfies each (Pi) of Definition 3

- (P0) to (P3) are trivially implied by the definition of $\mathcal{L}'$ and the fact that $\mathbf{G}$ is a complete completion graph (CCG).
- 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
    1. $\mathsf{Tail}'(t)$ is an R-successor of $\mathsf{Tail}(s)$, and $\mathbf{G}$ being a CCG 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
    2. $\mathsf{Tail}'(s)$ is an $\mathsf{Inv}(R)$-successor of $\mathsf{Tail}(t)$, and hence either $\mathsf{Tail}'(t) = \mathsf{Tail}(t)$ or the blocking condition implies $\forall R.C \in \mathcal{L}(\mathsf{Tail}'(s))$, and thus $\mathbf{G}$ being a CCG 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 $\mathbf{G}$ being a CCG 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 $\mathbf{G}$ being a CCG 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 $\mathbf{G}$ being a CCG 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 $\mathbf{G}$ being a CCG 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 $(\geqslant n\, R.C) \in \mathcal{L}'(s)$.

  - if $s \in \mathsf{Nom}(\mathbf{G})$, then $\mathbf{G}$ being a CCG implies the existence of n safe R-neighbours $y_1, \ldots, y_n$ of 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 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 $\mathbf{G}$ being a CCG 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 blockable, 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 ensures 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 a nominal node, 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 $(\leqslant n\, R.C) \in \mathcal{L}'(s)$. 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 $\mathbf{G}$ being a CCG 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 $\mathbf{G}$ being a CCG and the fact that nominal nodes are not "unravelled".

– (P13) is due to the way in which $\mathbf{G}$ was initialised, the fact that nominal nodes are never removed through pruning, and the fact that the labels of merged nodes are merged. □

Lemma 8 If there exists a tableau of a $\mathcal{SHOIQ}$ knowledge base $(\mathcal{T}, \mathcal{R})$, then, when started with $(\mathcal{T}, \mathcal{R})$, the expansion rules can be applied in such a way as to yield a complete and clash-free completion graph.

Proof: Given a tableau $T = (\mathbf{S}, \mathcal{L}', \mathcal{E})$ for $(\mathcal{T}, \mathcal{R})$, we can apply the non-deterministic rules, i.e., the $\sqcup$-, *choose*-, $\leqslant$-, $\leqslant_o$-, and *NN*-rule, in such a way that we obtain a complete and clash-free completion graph: inductively with the generation of new nodes, we define a mapping π from nodes in the completion graph to individuals in $\mathbf{S}$ in such a way that the following four properties hold throughout the run of the tableau algorithm:

(C1) for each node x, $\mathcal{L}(x) \cap \mathsf{cl}(\mathcal{T}, \mathcal{R}) \subseteq \mathcal{L}'(\pi(x))$,

(C2) for each pair of nodes x, y and each role R, if y is an R-successor of x, then $\langle \pi(x), \pi(y)\rangle \in \mathcal{E}(R)$,

(C3) $x \neq y$ implies $\pi(x) \neq \pi(y)$, and

(C4) if $(\leqslant n\, S.C) \in \mathcal{L}(x) \setminus \mathsf{cl}(\mathcal{T}, \mathcal{R})$, then $\sharp S^T(\pi(x), C) = n$,

where $S^T(s, C) = \{t \in \mathbf{S} \mid \langle s, t\rangle \in \mathcal{E}(S) \text{ and } C \in \mathcal{L}'(t)\}$.

We first explain why a completion graph satisfying (C1) to (C4) is clash-free: due to (C1), (P1), and the fact that node labels outside of $\mathsf{cl}(\mathcal{T}, \mathcal{R})$ are

either new nominals or atmost number restrictions, the completion graph does not contain a clash of the first kind; as a consequence of (C1) to (C3) and (P8) or as a consequence of (C2) to (C4), the completion graph does not contain a clash of the second kind; and as a consequence of (C1), (C3) and (P12), the completion graph does not contain a clash of the third kind.

Next, we show by induction on the number of rule applications how the expansion rules can be applied in such a way that (C1) to (C4) are preserved. We initialise π as follows: the initial completion graph contains, for each nominal o_i in $\mathcal{T}$, a node r_i with $\mathcal{L}(r_i) = \{o_i\}$, and thus we set $\pi(r_i) = s_i$ for $s_i \in \mathbf{S}$ with $o_i \in \mathcal{L}'(s_i)$. (P13) ensures that such s_i do indeed exist. Obviously, π satisfies (C1) to (C4).

For the induction step, it is not hard to see that (C4) may only be violated through an application of the *NN*-rule. By definition of the semantics, the application of the $\sqcap$- and the $\dot{\sqsubseteq}$-rule preserve (C1), and all other conditions are preserved trivially. By definition of the semantics, we can choose to apply the $\sqcup$-rule such that (C1) is preserved, and again all other conditions are preserved trivially. For the $\exists$- and the $\geqslant$-rule, we need to extend π to the newly introduced nodes—by definition of the semantics, this is possible in such a way that (C1) to (C3) are preserved, and the proof is analogous to the one in (Horrocks et al., 1999). Similarly, the fact that we can apply all remaining rules from Figure 1 without violating (C1) to (C3) can be found in (Horrocks et al., 1999). Thus we concentrate on the new rules in Figure 2.

If the o-rule is applied to x and y with $o \in \mathcal{L}(x) \cap \mathcal{L}(y)$, the fact that the *NN*-rule is the only one introducing new nominals and that it always introduces only new ones implies that $o \in \mathsf{cl}(\mathcal{T}, \mathcal{R})$. Hence (C1) implies that $o \in \mathcal{L}'(\pi(x)) \cap \mathcal{L}'(\pi(y))$, and thus (P12) implies $\pi(x) = \pi(y)$. As a consequence, applying the o-rule preserves (C1) to (C3).

If the *NN*-rule is applied to x with $(\leqslant n\, S.C) \in \mathcal{L}(x)$, let $m' = \sharp S^T(\pi(x), C)$ and note that the preconditions of the *NN*-rule imply that it has not been applied before to $(\leqslant n\, S.C) \in \mathcal{L}(x)$. As a consequence, $(\leqslant n\, S.C) \in \mathsf{cl}(\mathcal{T}, \mathcal{R})$, and thus (C1) and (P8) imply that $m' \leq n$. Hence we can apply it with $m := m'$ and extend π in the obvious way to map each newly introduced node to a different member of $S^T(\pi(x), C)$ without violating (C1) to (C4).

If the $\leqslant_o$-rule is applied to $(\leqslant n\, S.C) \in \mathcal{L}(x)$, y, and z_i as in the precondition of the $\leqslant_o$-rule, then (C1) together with (P8) or (C4) imply that $n \leq \sharp S^T(\pi(x), C)$. Hence (C2) and the inequalities $z_i \not\approx z_j$ for all $i < j \leq n$ together with (C3) imply $\pi(y) = \pi(z_i)$ for some $1 \leq i \leq m$, and thus we can apply the $\leqslant_o$-rule in such a way that it preserves (C1) to (C4).

Summing up, we can use T to steer the application of the non-deterministic rules to preserve (C1) to (C4), thereby avoiding clashes. Due to Lemma 6, this leads to a complete and clash-free completion graph. □

As an immediate consequence of Lemmas 4, 6, 7, and 8, the tableau algorithm always terminates, and answers with "$(\mathcal{T}, \mathcal{R})$ is consistent" iff $(\mathcal{T}, \mathcal{R})$ is consistent. As mentioned in Section 2, satisfiability and (non)subsumption can be reduced to knowledge base consistency.

Theorem 9 The tableau algorithm presented in Definition 5 is a decision procedure for consistency of $\mathcal{SHOIQ}$ knowledge bases, and for satisfiability and subsumption of $\mathcal{SHOIQ}$ concepts w.r.t. knowledge bases.

5. Outlook

In this paper, 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 (Tobies, 2000; Pacholski et al., 1997), it is clear that, in the worst case, any decision procedure will behave very badly, i.e., not terminate in practice. Our algorithm is far from being optimal in the worst case: it creates, in a non-deterministic way, a structure of size double exponential in the size of the input. However, it is designed to behave well in many typically encountered cases, and to exhibit a "pay as you go" behaviour: if an input knowledge base and concept(s) 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 $\mathcal{SHIQ}$ (Horrocks and Patel-Schneider, 1998; Haarslev and Möller, 2001b); it has, in fact, already been implemented in the well known Fact++ and Pellet systems, and has been shown to work well in practice (Tsarkov and Horrocks, 2006; Sirin et al., 2006; Gardiner et al., 2006).

References

Baader, F., D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider (eds.): 2003, *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

Baader, F. and P. Hanschke: 1991, 'A Schema for Integrating Concrete Domains into Concept Languages'. In: *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*. pp. 452–457.

Baader, F. and B. Hollunder: 1991, 'A Terminological Knowledge Representation System with Complete Inference Algorithms'. In: *Proc. of the First International Workshop on Processing Declarative Knowledge*, Vol. 572 of *Lecture Notes in Computer Science*. pp. 67–85, Springer Verlag.

Baader, F. and U. Sattler: 2001, 'An Overview of Tableau Algorithms for Description Logics'. *Studia Logica* **69**(1), 5–40.

Bechhofer, S., F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein: 2004, 'OWL Web Ontology Language Reference'. W3C Recommendation. Available at `http://www.w3.org/TR/owl-ref/`.

Blackburn, P. and J. Seligman: 1995, 'Hybrid Languages'. *J. of Logic, Language and Information* **4**, 251–272.

Brachman, R. J., D. L. McGuinness, P. F. Patel-Schneider, L. A. Resnick, and A. Borgida: 1991, 'Living with CLASSIC: When and how to Use a KL-ONE-like Language'. In: J. F. Sowa (ed.): *Principles of Semantic Networks*. Morgan Kaufmann, Los Altos, pp. 401–456.

Bresciani, P., E. Franconi, and S. Tessaris: 1995, 'Implementing and Testing Expressive Description Logics: Preliminary Report'. In: *Proc. of the 1995 Description Logic Workshop (DL'95)*. pp. 131–139.

Buchheit, M., F. M. Donini, and A. Schaerf: 1993, 'Decidable Reasoning in Terminological Knowledge Representation Systems'. *J. of Artificial Intelligence Research* **1**, 109–138.

Calvanese, D., G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati: 1998, 'Description Logic Framework for Information Integration'. In: *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*. pp. 2–13, Morgan Kaufmann, Los Altos.

De Giacomo, G.: 1995, 'Decidability of Class-Based Knowledge Representation Formalisms'. Ph.D. thesis, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza".

De Giacomo, G. and M. Lenzerini: 1994, 'Boosting the correspondence between description logics and propositional dynamic logics (extended abstract)'. In: *Proc. of the 12th Nat. Conf. on Artificial Intelligence (AAAI-94)*. AAAI Press.

De Giacomo, G. and M. Lenzerini: 1996, 'TBox and ABox Reasoning in Expressive Description Logics'. In: *Proc. of the 5th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'96)*. pp. 316–327, Morgan Kaufmann, Los Altos.

Donini, F. M., M. Lenzerini, and D. Nardi: 1990, 'Using Terminological Reasoning in Hybrid Systems'. *AI Communications—The European Journal for Artificial Intelligence* **3**(3), 128–138.

Fischer, M. J. and R. E. Ladner: 1979, 'Propositional Dynamic Logic of Regular Programs'. *J. of Computer and System Sciences* **18**, 194–211.

Gardiner, T., D. Tsarkov, and I. Horrocks: 2006, 'Framework For an Automated Comparison of Description Logic Reasoners'. In: *Proc. of the 2006 International Semantic Web Conference (ISWC 2006)*, Vol. 4273 of *Lecture Notes in Computer Science*. pp. 654–667, Springer Verlag.

Grädel, E.: 2001, 'Why are modal logics so robustly decidable?'. In: G. Paun, G. Rozenberg, and A. Salomaa (eds.): *Current Trends in Theoretical Computer Science. Entering the 21st Century*. World Scientific, pp. 393–408.

Haarslev, V. and R. Möller: 2001a, 'The Description Logic $\mathcal{ALCNH}_{R+}$ Extended with Concrete Domains: A Practically Motivated Approach'. In: *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, Vol. 2083 of *Lecture Notes in Artificial Intelligence*. pp. 29–44, Springer Verlag.

Haarslev, V. and R. Möller: 2001b, 'RACER System Description'. In: *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, Vol. 2083 of *Lecture Notes in Artificial Intelligence*. pp. 701–705, Springer Verlag.

Hladik, J. and J. Model: 2004, 'Tableau Systems for $\mathcal{SHIO}$ and $\mathcal{SHIQ}$'. In: *Proc. of the 2004 Description Logic Workshop (DL 2004)*. CEUR. Available from `ceur-ws.org`.

Hollunder, B. and F. Baader: 1991, 'Qualifying Number Restrictions in Concept Languages'. Technical Report RR-91-03, Deutsches Forschungszentrum für Künstliche Intelligenz

(DFKI), Kaiserslautern (Germany). An abridged version appeared in *Proc. of the 2nd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'91)*.

Horrocks, I. and P. F. Patel-Schneider: 1998, '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)*, Vol. 1397 of *Lecture Notes in Artificial Intelligence*. pp. 27–30, Springer Verlag.

Horrocks, I. and P. F. Patel-Schneider: 1999, 'Optimizing Description Logic Subsumption'. *J. of Logic and Computation* **9**(3), 267–293.

Horrocks, I., P. F. Patel-Schneider, and F. van Harmelen: 2003, 'From $\mathcal{SHIQ}$ and RDF to OWL: The Making of a Web Ontology Language'. *J. of Web Semantics* **1**(1), 7–26.

Horrocks, I. and U. Sattler: 2001, 'Ontology Reasoning in the $\mathcal{SHOQ}$(D) Description Logic'. In: *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*. pp. 199–204, Morgan Kaufmann, Los Altos.

Horrocks, I., U. Sattler, and S. Tobies: 1999, 'Practical Reasoning for Expressive Description Logics'. In: H. Ganzinger, D. McAllester, and A. Voronkov (eds.): *Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99)*, Vol. 1705 of *Lecture Notes in Artificial Intelligence*. pp. 161–180, Springer Verlag.

Horrocks, I., U. Sattler, and S. Tobies: 2000, 'Reasoning with Individuals for the Description Logic $\mathcal{SHIQ}$'. In: D. McAllester (ed.): *Proc. of the 17th Int. Conf. on Automated Deduction (CADE 2000)*, Vol. 1831 of *Lecture Notes in Computer Science*. pp. 482–496, Springer Verlag.

McGuinness, D. L. and J. R. Wright: 1998, 'An Industrial Strength Description Logic-based Configuration Platform'. *IEEE Intelligent Systems* pp. 69–77.

Pacholski, L., W. Szwast, and L. Tendera: 1997, 'Complexity of Two-Variable Logic with Counting'. In: *Proc. of the 12th IEEE Symp. on Logic in Computer Science (LICS'97)*. pp. 318–327, IEEE Computer Society Press.

Pan, J. and I. Horrocks: 2003, 'Web Ontology Reasoning with Datatype Groups'. In: D. Fensel, K. Sycara, and J. Mylopoulos (eds.): *Proc. of the 2003 International Semantic Web Conference (ISWC 2003)*, Vol. 2870 of *Lecture Notes in Computer Science*. pp. 47–63, Springer Verlag.

Patel-Schneider, P. F., B. Owsnicki-Klew, A. Kobsa, N. Guarino, R. MacGregor, W. S. Mark, D. McGuinness, B. Nebel, A. Schmiedel, and J. Yen: 1990, 'Term Subsumption Languages in Knowledge Representation'. *AI Magazine* **11**(2), 16–23.

Schaerf, A.: 1994, 'Reasoning with Individuals in Concept Languages'. *Data and Knowledge Engineering* **13**(2), 141–176.

Schild, K.: 1991, 'A Correspondence Theory for Terminological Logics: Preliminary Report'. In: *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*. pp. 466–471, Morgan Kaufmann, Los Altos.

Sirin, E., B. C. Grau, and B. Parsia: 2006, 'From Wine to Water: Optimizing Description Logic Reasoning for Nominals'. In: *Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2006)*. pp. 90–99, AAAI Press.

Sirin, E., B. Parsia, B. Cuenca Grau, A. Kalyanpur, and Y. Katz: 2003, 'Pellet: A Practical OWL-DL Reasoner'. Submitted to the Journal of Web Semantics.

Tobies, S.: 2000, 'The Complexity of Reasoning with Cardinality Restrictions and Nominals in Expressive Description Logics'. *J. of Artificial Intelligence Research* **12**, 199–217.

Tobies, S.: 2001, 'Complexity Results and Practical Algorithms for Logics in Knowledge Representation'. Ph.D. thesis, LuFG Theoretical Computer Science, RWTH-Aachen, Germany.

Tsarkov, D. and I. Horrocks: 2005, 'Ordering Heuristics for Description Logic Reasoning'. In: *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*. pp. 609–614, Morgan Kaufmann, Los Altos.

Tsarkov, D. and I. Horrocks: 2006, 'FaCT++ Description Logic Reasoner: System Description'. In: *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, Vol. 4130 of *Lecture Notes in Artificial Intelligence*. pp. 292–297, Springer Verlag.

Vardi, M. Y.: 1997, 'Why Is Modal Logic so Robustly Decidable'. In: *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol. 31. pp. 149–184, American Mathematical Society.