From Tableaux to Automata for Description Logics

Franz Baader[1], Jan Hladik[1], Carsten Lutz[1], and Frank Wolter[2]

[1] Theoretical Computer Science, TU Dresden,
D-01062 Dresden, Germany
{baader,hladik,lutz}@tcs.inf.tu-dresden.de
[2] Department of Computer Science, University of Liverpool
Liverpool L69 7ZF, U.K.
frank@csc.liv.ac.uk

Abstract. This paper investigates the relationship between automata- and tableau-based inference procedures for Description Logics. To be more precise, we develop an abstract notion of what a tableau-based algorithm is, and then show, on this abstract level, how tableau-based algorithms can be converted into automata-based algorithms. In particular, this allows us to characterize a large class of tableau-based algorithms that imply an ExpTime upper-bound for reasoning in the description logics for which such an algorithm exists.

1 Introduction

Description logics (DLs) [1] are a family of knowledge representation languages which can be used to represent the terminological knowledge of an application domain in a structured and formally well-understood way. The name *description logics* is motivated by the fact that, on the one hand, the important notions of the domain are described by *concept descriptions*, i.e., expressions that are built from atomic concepts (unary predicates) and atomic roles (binary predicates) using the concept and role constructors provided by the particular DL. On the other hand, DLs differ from their predecessors, such as semantic networks and frames [25, 21], in that they are equipped with a formal, *logic*-based semantics, which can, e.g., be given by a translation into first-order predicate logic.

Knowledge representation systems based on description logics (DL systems) [30, 22] provide their users with various inference capabilities (like subsumption and instance checking) that allow them to deduce implicit knowledge from the explicitly represented knowledge. In order to ensure a reasonable and predictable behavior of a DL system, these inference problems should at least be decidable, and preferably of low complexity. Consequently, the expressive power of the DL in question must be restricted in an appropriate way. If the imposed restrictions are too severe, however, then the important notions of the application domain can no longer be expressed. Investigating this trade-off between the expressivity of DLs and the complexity of their inference problems has been one of the most important issues in DL research (see [9] for an overview of complexity results).

The focus of this research has, however, changed in the last 15 years. In the beginning of the 1990ies, DL researchers investigated the border between tractable and intractable DLs [11, 12], and systems that employed so-called structural subsumption algorithms, which first normalize the concept descriptions, and then recursively compare the syntactic structure of the normalized descriptions, were still prevalent [24, 19, 20, 23]. It quickly turned out, however, that structural subsumption algorithms can handle only very inexpressive languages, and that one cannot expect a DL of reasonable expressive power to have tractable inference problems. For expressive DLs, tableau-based inference procedures turned out to be quite useful. After the first such tableau-based subsumption algorithm was developed by Schmidt-Schauß and Smolka [27] for the DL $\mathcal{ALC}$, this approach was extended to various other DLs and also to other inference problems such as the instance problem (see [4] for an overview).

Most of these early tableau-based algorithms for DLs were of optimal worst-case complexity: they treated DLs with a PSpace-complete subsumption problem, and the algorithms needed only polynomial space. Thus, by designing a tableau-based algorithm for such a DL one could solve two problems simultaneously: prove an optimal complexity upper-bound, and describe an algorithm that is easy to implement and optimize [2], thus yielding a practical reasoning system for this DL. Modern tableau-based DL reasoners such as FaCT [15] and RACER [13] are based on very expressive DLs (like $\mathcal{SHIQ}$ [18]), which have an ExpTime-complete subsumption problem. Despite the high worst-case complexity of the underlying logics, the systems FaCT and RACER behave quite well in realistic applications. This is mainly due to the fact that their implementors have developed a great variety of sophisticated optimization techniques for tableau-based algorithms (see [16] for an overview of these techniques). Tableau-based algorithms are, however, notoriously bad at proving ExpTime upper-bounds.[3] In many cases, ExpTime upper-bounds are easily established using automata-based approaches (see, e.g., Section 5.3 in [7]). However, to the best of our knowledge, there exist no practical DL reasoners based on automata techniques. Until now, it was thus common practice to devise two different algorithms for every ExpTime-complete DL, an automata-based one for establishing the exact worst-case complexity, and a tableau-based one for the implementation.

This paper investigates the (rather close) relationship between automata- and tableau-based algorithms. To be more precise, we develop an abstract notion of what a tableau-based algorithm is, and then show, on this abstract level, how tableau-based algorithms can be converted into automata-based algorithms. In particular, this allows us to characterize a large class of tableau-based algorithms that imply an ExpTime upper-bound for reasoning in the DLs for which such an algorithm exists. We consider this to be a very useful result since, in many cases, it eliminates the need for developing two algorithms for the same DL: one can now design a tableau-based algorithm, use our general result to obtain an ExpTime upper-bound, and then base a practical implementation on the very

[3] The only such result we know of [10] treats the case of $\mathcal{ALC}$ with general concept inclusions (GCIs), and even in this simple case the algorithm is very complicated.

same algorithm. We illustrate the usefulness of our framework by reproving the known ExpTime upper-bounds for the description logic $\mathcal{ALC}$ with general concept inclusions [26], and for the extension $\mathcal{ALCQI}$ of $\mathcal{ALC}$ by qualified number restrictions and inverse roles [8].

In the next section, we introduce the abstract notion of a tableau system. In order to motivate and illustrate the technical definitions, we first consider the example of a tableau-based algorithm for $\mathcal{ALC}$ with general concept inclusions. In Section 3 we define additional restrictions on tableau systems that ensure an exponential upper-bound on reasoning. This upper-bound is shown via a translation of tableau systems into looping tree automata. In Section 4 we show how tableau systems can directly be used to obtain a tableau-based decision procedure, which can be the basis for an optimized implementation. The main problem to be solved there is to ensure termination of the tableau-based algorithm. In Section 5 we apply the abstract framework to a more complex DL: we design a tableau system for the DL $\mathcal{ALCQI}$, thus giving an alternative proof of the known ExpTime upper-bound for reasoning in this DL. Finally, in Section 6 we discuss possible variants and extensions of the abstract framework.

2 Formalizing Tableau Algorithms

In this section, we develop an abstract formalization of tableau algorithms. To this end, we first discuss the standard tableau-based algorithm for the basic description logic $\mathcal{ALC}$, and then use this concrete example as a guide when devising the abstract framework.

2.1 A Tableau Algorithm for $\mathcal{ALC}$

We start with introducing the syntax and semantics of $\mathcal{ALC}$:

Definition 1 **($\mathcal{ALC}$ Syntax).** *Let $\mathsf{N_C}$ and $\mathsf{N_R}$ be pairwise disjoint and countably infinite sets of* concept names *and* role names*. The set of $\mathcal{ALC}$-concepts $\mathsf{CON}_{\mathcal{ALC}}$ is the smallest set such that*

- *every concept name is an $\mathcal{ALC}$-concept, and*
- *if C and D are $\mathcal{ALC}$-concepts and r is a role name, then the following expressions are also $\mathcal{ALC}$-concepts: $\neg C$, $C \sqcap D$, $C \sqcup D$, $\exists r.C$, $\forall r.C$.*

A general concept inclusion (GCI) *is an expression $C \sqsubseteq D$, where both C and D are $\mathcal{ALC}$-concepts. A finite set of GCIs is called $\mathcal{ALC}$-*TBox*. We use $\mathsf{TBOX}_{\mathcal{ALC}}$ to denote the set of all $\mathcal{ALC}$-TBoxes.*

As usual, we will use $\top$ as abbreviation for an arbitrary propositional tautology, $\bot$ for $\neg\top$, and $C \rightarrow D$ for $\neg C \sqcup D$.

Note that there exist several different TBox formalisms that vary considerably w.r.t. expressive power (see [3]). The kind of TBoxes adopted here are among the most general ones available. They are supported by modern DL reasoners such as FaCT and RACER.

Like all DLs, $\mathcal{ALC}$ is equipped with a Tarski-style set-theoretic semantics.

Definition 2 **($\mathcal{ALC}$ Semantics).** *An* interpretation $\mathcal{I}$ *is a pair* $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$*, where* $\Delta^{\mathcal{I}}$ *is a non-empty set, called the* domain*, and* $\cdot^{\mathcal{I}}$ *is the* interpretation function*. The interpretation function maps each concept name* A *to a subset* $A^{\mathcal{I}}$ *of* $\Delta^{\mathcal{I}}$ *and each role name* r *to a subset* $r^{\mathcal{I}}$ *of* $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$*. It is extended to arbitrary* $\mathcal{ALC}$*-concepts as follows:*

$$
\begin{aligned}
(\neg C)^{\mathcal{I}} &:= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} &:= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} &:= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\exists r.C)^{\mathcal{I}} &:= \{d \in \Delta^{\mathcal{I}} \mid \textit{There is } e \in \Delta^{\mathcal{I}} \textit{ with } (d,e) \in r^{\mathcal{I}} \textit{ and } e \in C^{\mathcal{I}}\} \\
(\forall r.C)^{\mathcal{I}} &:= \{d \in \Delta^{\mathcal{I}} \mid \textit{For all } e \in \Delta^{\mathcal{I}}, \textit{ if } (d,e) \in r^{\mathcal{I}}, \textit{ then } e \in C^{\mathcal{I}}\}
\end{aligned}
$$

The interpretation $\mathcal{I}$ *is a* model *of the* $\mathcal{ALC}$*-concept* C *iff* $C^{\mathcal{I}} \neq \emptyset$*, and it is a model of the TBox* $\mathcal{T}$ *iff* $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ *holds for all* $C \sqsubseteq D \in \mathcal{T}$*.*

The main inference problems related to a TBox are satisfiability and subsumption of concepts.

Definition 3 **($\mathcal{ALC}$ Inference Problems).** *The* $\mathcal{ALC}$*-concept* C *is* satisfiable w.r.t. the TBox $\mathcal{T}$ *iff* C *and* $\mathcal{T}$ *have a common model, and* C *is* subsumed by *the* $\mathcal{ALC}$*-concept* D w.r.t. the TBox $\mathcal{T}$ *(written* $C \sqsubseteq_{\mathcal{T}} D$*) iff* $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ *holds for all models* $\mathcal{I}$ *of* $\mathcal{T}$*.*

Since $C \sqsubseteq_{\mathcal{T}} D$ iff $C \sqcap \neg D$ is unsatisfiable w.r.t. $\mathcal{T}$, it is sufficient to design a satisfiability algorithm. We now discuss the standard tableau-based satisfiability algorithm for $\mathcal{ALC}$. This algorithm has first been described in [27]; more modern accounts can, e.g., be found in [4]. It can rightfully be viewed as the ancestor from which all state-of-the-art tableau-based algorithms for description logics are descended. Such algorithms are nowadays the standard approach for reasoning in DLs, and they underlie modern and efficient reasoning systems such as FaCT and RACER, which are based on DLs that are much more expressive than $\mathcal{ALC}$.

Tableau algorithms are characterized by an underlying data structure, a set of completion rules, and a number of so-called clash-triggers. To decide the satisfiability of an input concept C w.r.t. an input TBox $\mathcal{T}$, the algorithm starts with an initial instance of the data structure constructed from C and $\mathcal{T}$, and repeatedly applies completion rules to it. This rule application can be viewed as an attempt to construct a model for the input, or as making implicit knowledge explicit. Rule application continues until either one of the clash-triggers applies, which means that the attempt to construct a model has failed, or all implicit knowledge has been made explicit without encountering a clash-trigger. In the latter case, the algorithm has succeeded to construct (a representation of) a model. To be more precise, the tableau algorithms considered in this paper may be non-deterministic, i.e., there may exist completion rules that yield more than one possible outcome. In this case, the algorithm returns "satisfiable" iff there exists at least *one* way to apply the non-deterministic rules such that a

$\mathsf{R}\sqcap$	if $C_1 \sqcap C_2 \in \mathcal{N}(a)$ and $\{C_1, C_2\} \not\subseteq \mathcal{N}(a)$ then $\mathcal{N}(a) := \mathcal{N}(a) \cup \{C_1, C_2\}$
$\mathsf{R}\sqcup$	if $C_1 \sqcup C_2 \in \mathcal{N}(a)$ and $\{C_1, C_2\} \cap \mathcal{N}(a) = \emptyset$ then $\mathcal{N}(a) := \mathcal{N}(a) \cup \{C\}$ for some $C \in \{C_1, C_2\}$
$\mathsf{R}\exists$	if $\exists r.C \in \mathcal{N}(a)$ and there is no r-successor b of a with $C \in \mathcal{N}(b)$, then generate a new successor b of a, and set $\mathcal{E}(a, b) := r$ and $\mathcal{N}(b) := \{C\}$
$\mathsf{R}\forall$	if $\forall r.C \in \mathcal{N}(a)$ and b is an r-successor of a with $C \notin \mathcal{N}(b)$ then set $\mathcal{N}(b) := \mathcal{N}(b) \cup \{C\}$
$\mathsf{R}\mathcal{T}$	if $C \sqsubseteq D \in \mathcal{T}$ and $\mathsf{nnf}(C \to D) \notin \mathcal{N}(a)$ then set $\mathcal{N}(a) := \mathcal{N}(a) \cup \{\mathsf{nnf}(C \to D)\}$

Fig. 1. Completion rules for $\mathcal{ALC}$.

model of the input is obtained. Note that only the choice of the outcome of non-deterministic rules is true "don't know" non-determinism (and thus requires backtracking), whereas the order of rule applications is basically "don't care" non-determinism.

Before we can define the data structure underlying the $\mathcal{ALC}$ tableau algorithm, so-called completion trees, we must introduce some notation. Given an $\mathcal{ALC}$-concept C, its *negation normal form* is an equivalent[4] concept such that negation occurs only in front of concept names. Such a concept can easily be computed by pushing negation as far as possible into concepts, using de Morgan's rules and the usual duality rules for quantifiers. We will denote the negation normal form of C by $\mathsf{nnf}(C)$. If C is an $\mathcal{ALC}$-concept and $\mathcal{T}$ an $\mathcal{ALC}$-TBox, then we use $\mathsf{sub}(C, \mathcal{T})$ to denote the set of all subconcepts of the concepts in the set

$$\{\mathsf{nnf}(C)\} \cup \bigcup_{D \sqsubseteq E \in \mathcal{T}} \{\mathsf{nnf}(D \to E)\}.$$

Definition 4 (Completion Trees). *Let C be an $\mathcal{ALC}$-concept and $\mathcal{T}$ an $\mathcal{ALC}$-TBox. A* completion tree *for C and $\mathcal{T}$ is a labeled tree*[5] *$T = (V, E, \mathcal{N}, \mathcal{E})$ of finite out-degree such that (V, E) is a tree, each node $a \in V$ is labeled with a subset $\mathcal{N}(a)$ of $\mathsf{sub}(C, \mathcal{T})$ and each edge $(a, b) \in E$ is labeled with a role name $\mathcal{E}(a, b)$ occurring in C or $\mathcal{T}$.*

The completion rules are given in Figure 1, where $\mathsf{R}\sqcup$ is the only non-deterministic rule. To decide satisfiability of a concept C w.r.t. a TBox $\mathcal{T}$, the $\mathcal{ALC}$ tableau algorithm starts with the initial completion tree

$$T_{C,\mathcal{T}} := (\{x\}, \emptyset, \{x \mapsto \{\mathsf{nnf}(C)\}, \emptyset)$$

[4] Two concepts are equivalent iff they subsume each other w.r.t. the empty TBox.

[5] Here and in the following, a tree is an acyclic directed graph (V, E) with a unique root where every node other than the root is reachable from the root and has exactly one predecessor. The edge relation E is a sub*set* of $V \times V$, and thus the successors of a given node are not ordered.

and repeatedly applies completion rules. Rule application stops in one of the following two cases:

1. the obtained completion tree $T = (V, E, \mathcal{N}, \mathcal{E})$ *contains a clash*, i.e. there is a node $a \in V$ and a concept name A such that $\{A, \neg A\} \subseteq \mathcal{N}(a)$;
2. T is *saturated*, i.e. no more completion rule is applicable to T.

If we consider only empty TBoxes (and thus drop the $\mathsf{R}\mathcal{T}$ rule), then the described algorithm terminates for any input and any sequence of rule applications. Things are not so simple if we admit non-empty TBoxes: because of the $\mathsf{R}\mathcal{T}$ rule, the algorithm need not terminate, both on satisfiable and on unsatisfiable inputs. For example, rule application to the concept $\top$ and the TBox $\{\top \sqsubseteq \exists R.\top\}$ continues indefinitely. However, the algorithm then computes an infinite "increasing" sequence of completion trees: in each step, the tree and its node labels may only grow but never shrink. In case of non-termination, there thus exists a unique completion tree computed by this run of the algorithm "in the limit". Thus, both terminating and non-terminating runs of the algorithm "compute" a unique completion tree. This (possibly infinite) completion tree is called *saturated* iff no more completion rule is applicable to it.

The tableau algorithm for $\mathcal{ALC}$ is sound and complete in the following sense:

- *Soundness.* If the algorithm computes a saturated and clash-free completion tree for the input $C, \mathcal{T}$, then C is satisfiable w.r.t. $\mathcal{T}$.
- *Completeness.* If the input $C, \mathcal{T}$ is satisfiable, then there is a run of the algorithm that computes a saturated and clash-free completion tree for this input.

Obviously, any terminating run of the algorithm yields a saturated completion tree. For this reason, the order of rule applications is in this case "don't care" non-deterministic. For a non-terminating run, this is only true if we require completion rules to be applied in a *fair*[6] manner. Ensuring fairness is a simple task: we can, e.g., always apply completion rules to those nodes in the tree that are as close to the root as possible. This yields a fair strategy since the out-degree of completion trees constructed for an input $C, \mathcal{T}$ is bounded by the cardinality of the set $\mathsf{sub}(C, \mathcal{T})$.

Although the procedure as described until now does not necessarily terminate and thus is no decision procedure for satisfiability, quite surprisingly we will see that it already provides us with enough information to deduce an ExpTime upper-bound for $\mathcal{ALC}$-concept satisfiability (and thus, in particular, with a decidability result). This is shown by a translation into a tree automaton, which basically accepts saturated and clash-free completion trees for the input. We view this as a rather convenient feature of our framework: to obtain an ExpTime decision procedure, it is sufficient to design a sound and complete tableau algorithm and not even bother to prove termination, a usually hard task (see

[6] Intuitively, fairness means that rules are applied such that every applicable rule will eventually be applied unless it is made inapplicable by the application of other rules.

Section 3 for details). Moreover, we will show in Section 4 that a given non-terminating sound and complete tableau procedure can always be turned into a terminating sound and complete procedure. This yields a tableau-based decision procedure, which is, however, not necessarily of ExpTime complexity.

2.2 The General Framework

We now develop a general notion of tableau algorithms. It is in the nature of this endeavor that our formalism will be a rather abstract one. We start with defining the core notion: tableau systems. Intuitively, the purpose of a tableau system is to capture all the details of a tableau algorithm such as the one for $\mathcal{ALC}$ discussed in the previous section. The set $\mathfrak{I}$ of inputs used in the following definition can be thought of as consisting of all possible pairs $(C, \mathcal{T})$ of concepts C and TBoxes $\mathcal{T}$ of the DL under consideration.

Definition 5 (Tableau System). *Let $\mathfrak{I}$ be a set of* inputs*. A* tableau system *for $\mathfrak{I}$ is a tuple*

$$S = (\mathsf{NLE}, \mathsf{EL}, \cdot^S, \mathcal{R}, \mathcal{C}),$$

where NLE and EL are sets of node label elements *and* edge labels*, respectively, and $\cdot^S$ is a function mapping each input $\Gamma \in \mathfrak{I}$ to a tuple*

$$\Gamma^S = (\mathsf{nle}, \mathsf{el}, \mathsf{ini})$$

such that

- $\mathsf{nle} \subseteq \mathsf{NLE}$ *and* $\mathsf{el} \subseteq \mathsf{EL}$ *are finite;*
- ini *is a subset of* $\wp(\mathsf{nle})$*, where* $\wp(\cdot)$ *denotes powerset.*

*The definitions of $\mathcal{R}$ and $\mathcal{C}$ depend on the notion of an S-*pattern*, which is a finite labeled tree*

$$(V, E, n, \ell),$$

of depth at most one with $n : V \to \wp(\mathsf{NLE})$ and $\ell : E \to \mathsf{EL}$ node and edge labeling functions.

- *$\mathcal{R}$, the collection of* completion rules*, is is a function mapping each S-pattern to a finite set of non-empty finite sets of S-patterns;*
- *$\mathcal{C}$, the collection of* clash-triggers*, is a set of S-patterns.*

To illustrate tableau systems, we now define a tableau system $S_{\mathcal{ALC}}$ that describes the $\mathcal{ALC}$ tableau algorithm discussed in the previous section. Intuitively, NLE is the set of elements that may appear in node labels of completion trees, *independently* of the input. In the case of $\mathcal{ALC}$, NLE is thus simply $\mathsf{CON}_{\mathcal{ALC}}$. Similarly, EL is the set of edge labels, also independently of the input. In the case of $\mathcal{ALC}$, EL is thus the set of role names $\mathsf{N_R}$.

The function $\cdot^S$ describes the impact of the input on the form of the constructed completion trees. More precisely, nle fixes the node label elements that may be used in a completion tree for a particular input, and el fixes the edge labels. Finally, ini describes the possible initial node labels of the root of the completion tree. Note that the initial root label is not necessarily unique, but rather there can be many choices—a possible source of (don't know) non-determinism that does not show up in the $\mathcal{ALC}$ algorithm.

To illustrate the function $\cdot^S$, let us define it for the tableau system $S_{\mathcal{ALC}}$. For simplicity, we write $\mathsf{nle}_{S_{\mathcal{ALC}}}(C, \mathcal{T})$ to refer to the first element of the tuple $(C, \mathcal{T})^{S_{\mathcal{ALC}}}$, $\mathsf{el}_{S_{\mathcal{ALC}}}(C, \mathcal{T})$ to refer to the second element of the tuple $(C, \mathcal{T})^{S_{\mathcal{ALC}}}$, and so forth. For each input $C, \mathcal{T} \in \mathsf{CON}_{\mathcal{ALC}} \times \mathsf{TBOX}_{\mathcal{ALC}}$, we have

$$\begin{aligned}
\mathsf{nle}_{S_{\mathcal{ALC}}}(C, \mathcal{T}) &= \mathsf{sub}(C, \mathcal{T});\\
\mathsf{el}_{S_{\mathcal{ALC}}}(C, \mathcal{T}) &= \{r \in \mathsf{N_R} \mid r \text{ appears in C or } \mathcal{T}\};\\
\mathsf{ini}_{S_{\mathcal{ALC}}}(C, \mathcal{T}) &= \{\{\mathsf{nnf}(C)\}\}.
\end{aligned}$$

It remains to formalize the completion rules and clash-triggers. First observe that, in the $\mathcal{ALC}$ tableau, every clash-trigger, every rule premise, and every rule consequence concerns only a single node either alone or together with its successors in the completion tree. This observation motivates our definition of patterns, which formalize clash-triggers as well as pre- and post-conditions of completion rules. The collection of completion rules $\mathcal{R}$ maps patterns to finite sets of finite sets of patterns. Intuitively, if P is a pattern and $\{P_1, \ldots, P_k\} \in \mathcal{R}(P)$, then this means that a rule of the collection can be applied to completion trees matching the pattern P, non-deterministically replacing the "area" matching P with an "area" matching one of the patterns $P_1, \ldots, P_k$ (we will give a formal definition of this later on). If $\{P_1, \ldots, P_k\} \in \mathcal{R}(P)$, then we will sometimes write

$$P \rightarrow_{\mathcal{R}} \{P_1, \ldots, P_k\}$$

to indicate the rule induced by this element of $\mathcal{R}(P)$. Similar to the application of such a rule, a completion tree contains a clash if this completion tree matches a pattern in $\mathcal{C}$.

To illustrate this, let us again consider the case of $\mathcal{ALC}$. For $\mathcal{ALC}$, the set of clash-triggers $\mathcal{C}$ consists of all patterns whose root label contains both A and $\neg A$ for some concept name A. Thus, a completion tree contains a clash if one of its nodes labels contains A and $\neg A$ for some concept name A.

With one exception, the collection of completion rules is defined by a straightforward translation of the rules in Figure 1. For each pattern $P = (V, E, n, \ell)$ with root v_0, $\mathcal{R}(P)$ is the smallest set of finite sets of patterns such that the following holds:

- $\mathsf{R}\sqcap$ if the root label $n(v_0)$ contains the concept $C \sqcap D$ and $\{C, D\} \not\subseteq n(v_0)$, then $\mathcal{R}(P)$ contains the singleton set $\{(V, E, n', \ell)\}$, where $n'(v) = n(v)$ for all $v \in V \setminus \{v_0\}$ and $n'(v_0) = n(v_0) \cup \{C, D\}$;
- $\mathsf{R}\sqcup$ if the root label $n(v_0)$ contains the concept $C \sqcup D$ and $\{C, D\} \cap n(v_0) = \emptyset$, then $\mathcal{R}(P)$ contains the set $\{(V, E, n', \ell), (V, E, n'', \ell)\}$, where $n'(v) = n''(v) = n(v)$ for all $v \in V \setminus \{v_0\}$ and $n'(v_0) = n(v_0) \cup \{C\}$ and $n''(v_0) = n(v_0) \cup \{D\}$;

R∃ if the root label $n(v_0)$ contains the concept $\exists r.C$, $u_1, \ldots, u_m$ are all the sons of v_0 with $\ell(v_0, u_i) = r$, and $C \notin n(u_i)$ for all $i, 1 \leq i \leq m$, then $\mathcal{R}(P)$ contains the set $\{P_0, P_1, \ldots, P_m\}$, where

- $P_0 = (V_0, E_0, n_0, \ell_0)$, where u_0 is a node not contained in V, $V_0 = V \cup \{u_0\}$, $E' = E \cup \{(v_0, u_0)\}$, $n_0 = n \cup \{u_0 \mapsto \{C\}\}$, $\ell' = \ell \cup \{(v_0, u_0) \mapsto r\}$,
- for $i = 1, \ldots, m$, $P_i = (V, E, n_i, \ell)$, where $n_i(v) = n(v)$ for all $v \in V \setminus \{u_i\}$ and $n_i(u_i) = n(u_i) \cup \{C\}$;

R∀ if $n(v_0)$ contains the concept $\forall r.C$, $\ell(r, v_1) = r$ for some $v_1 \in V$, and $C \notin n(v_1)$, then $\mathcal{R}(P)$ contains $\{(V, E, n', \ell)\}$, where $n'(v) = n(v)$ for all $v \in V \setminus \{v_1\}$ and $n'(v_1) = n(v_1) \cup \{C\}$;

R$\mathcal{T}$ if $C \sqsubseteq D \in \mathcal{T}$ and $\mathsf{nnf}(C \to D) \notin n(v_0)$, then $\mathcal{R}(P)$ contains the set $\{(V, E, n', \ell)\}$, where $n'(v) = n(v)$ for all $v \in V \setminus \{v_0\}$ and $n'(v_0) = n(v_0) \cup \{\mathsf{nnf}(C \to D)\}$.

The exception is the treatment of existential restrictions. The rule in Figure 1 is deterministic: it always generates a *new* r-successor of the given node. In contrast, the rule handling existential restrictions introduced above (don't know) non-deterministically chooses between generating a new successor or re-using one of the old ones. Basically, this is the price we have to pay for having a very general framework. The reason why one can always create a new individual when treating existential restrictions in $\mathcal{ALC}$ is that $\mathcal{ALC}$ is invariant under bisimulation [5], and thus one can duplicate successors in models without changing validity. We could have tailored our framework such that the deterministic rule for $\mathcal{ALC}$ can be used, but then we basically would have restricted its applicability to DLs invariant under bisimulation (see Section 6 for a more detailed discussion of this issue).

Let us now continue with the general definitions. Tableau systems are a rather general notion. In fact, as described until now they are too general to be useful for our purposes. For example, tableau algorithms described by such tableau systems need not be monotonic: completion rules could repeatedly (even indefinitely) add and remove the same piece of information. To prevent such pathologic behavior, we now formulate a number of conditions that "well-behaved" tableau systems are supposed to satisfy. For the following definitions, fix a set of inputs $\mathfrak{I}$ and a tableau system $S = (\mathsf{NLE}, \mathsf{EL}, \cdot^S, \mathcal{R}, \mathcal{C})$ for $\mathfrak{I}$. Before we can define admissibility of tableau systems, we must introduce an "inclusion relation" between patterns.

Definition 6. *Let $P = (V, E, n, \ell)$ and $P' = (V', E', n', \ell')$ be S-patterns. We write $P \precsim P'$ iff the following conditions are satisfied: there is an injection $\pi : V \to V'$ that maps the root of P to the root of P' and satisfies the following conditions:*

- *for all $x \in V$, we have $n(x) \subseteq n'(\pi(x))$;*
- *for all $x, y \in V$, if $(x, y) \in E$, then $(\pi(x), \pi(y)) \in E'$ and $\ell(x, y) = \ell'(\pi(x), \pi(y))$;*

If π is the identity on V, then we write $P \preceq P'$ (and $P \prec P'$ if, additionally, $P \neq P'$). If π is a bijection and $n(x) = n'(\pi(x))$ for all $x \in V$, then we write

$P \sim P'$. *To make the injection (bijection)* π *explicit, we sometimes write* $P \precsim_\pi P'$ *(*$P \sim_\pi P'$*).*

Let $\Gamma \in \Im$ be an input. We say that P is a *pattern for* Γ iff the labels of all nodes in P are subsets of $\mathsf{nle}_S(\Gamma)$ and the labels of all edges in P belong to $\mathsf{el}_S(\Gamma)$. The pattern P is *saturated* iff $\mathcal{R}(P) = \emptyset$.

Definition 7 (Admissible). *The tableau system* S *is called* admissible *iff it satisfies, for all* S*-patterns* $P = (V, E, n, \ell)$ *and* $P' = (V', E', n', \ell')$*, the following conditions:*

1. *If* $P \rightarrow_\mathcal{R} \{P_1, \ldots, P_k\}$*, then* $P \prec P_i$ *for all* $i, 1 \leq i \leq k$*.*
2. *If* $P \rightarrow_\mathcal{R} \{P_1, \ldots, P_k\}$*,* P' *is saturated, and* $P \precsim P'$*, then there exists an* $i, 1 \leq i \leq k$*, such that* $P_i \precsim P'$*.*
3. *For all inputs* $\Gamma \in \Im$*, if* P *is a pattern for* Γ *and* $P \rightarrow_\mathcal{R} \{P_1, \ldots, P_k\}$*, then the patterns* P_i *are patterns for* Γ*.*
4. *If* $P \in \mathcal{C}$ *and* $P \precsim P'$*, then* $P' \in \mathcal{C}$*.*

It is in order to discuss the intuition underlying the above conditions. Condition 1 basically says that rule application always adds nodes or elements of node labels. Condition 2 can be understood as follows. Assume that a (non-deterministic) rule is applicable to P and that P' is a "superpattern" of P that is saturated (i.e., all applicable rules have already been applied). Then the non-deterministic rule can be applied in such a way that the obtained new pattern is still a subpattern of P'. Intuitively, this condition can be used to reach P' from P by repeated rule application. Condition 3 says that, by applying completion rules for some input Γ, we stay within the limits given by the values of the $\cdot^S$ function. Condition 4 states that applicability of clash-triggers is monotonic, i.e., if a pattern triggers a clash, all its "superpatterns" also trigger a clash.

It is easy to see that these conditions are satisfied by the tableau system $S_{\mathcal{ALC}}$ for $\mathcal{ALC}$. For Condition 1, this is obvious since the rules only add nodes or elements of node labels, but never remove them. Condition 3 holds since rules only add subconcepts of existing concepts to the node label. Condition 4 is also clear: if the label of the root of P contains A and $\neg A$, then the label of the root of every superpattern also contains A and $\neg A$.

The most interesting condition is Condition 2. We illustrate it by considering the treatment of disjunction and of existential restrictions in $S_{\mathcal{ALC}}$. First, assume that $P \rightarrow_\mathcal{R} \{P_1, P_2\}$ where the root label of P contains $C \sqcup D$ and the root labels of P_1 and P_2 are obtained from the root label of P by respectively adding C and D. If $P \precsim P'$, then the root label of P' also contains $C \sqcup D$. If, in addition, P' is saturated, then its root label already contains C or D. In the first case, $P' \precsim P_1$ and in the second $P' \precsim P_2$.

Second, consider the rules handling existential restrictions. Thus, let $P \precsim P'$, and assume that the root label of P contains the existential restriction $\exists r.C$ and that the root of P has m r-successors $u_1, \ldots, u_m$. Then the existential restriction $\exists r.C$ induces the rule $P \rightarrow_\mathcal{R} \{P_0, \ldots, P_m\}$ where the patterns $P_0, \ldots, P_m$ are as defined above. If, in addition, P' is saturated, then its root has an r-successor whose label contains C. If this is a "new" r-successor (i.e., one not in the range of the injection π that ensures $P \precsim P'$), then $P_0 \precsim P'$.[7] Otherwise, there is an r-successor u_i of the root of P such that the label of $\pi(u_i)$ in P' contains C. In this case, $P_i \precsim P'$.

We now introduce S-trees, the abstract counterpart of completion trees, and define what it means for a pattern to match into an S-tree.

Definition 8 (S-tree, Matching). *An S-*tree *is a labeled tree $T = (V, E, n, \ell)$ with finite out-degree, a countable set of nodes V, and the node and edge labeling functions $n : V \to \wp(\mathsf{NLE})$ and $\ell : E \to \mathsf{EL}$.*

Any node $x \in V$ defines a pattern T, x, the neighborhood *of x in T, as follows: $T, x := (V', E', n', \ell')$ where*

- *$V' = \{x\} \cup \{y \in V \mid (x, y) \in E\}$;*
- *E', n', ℓ' are the restrictions of E, n, ℓ to V';*

If $P = (V', E', n', \ell')$ is an arbitrary S-pattern and $x \in V$, then we say that P matches *x in T iff $P \sim T, x$ (see Definition 6).*

For the tableau system for $\mathcal{ALC}$ introduced above, $S_{\mathcal{ALC}}$-trees are exactly the completion trees defined in Section 2.

We are now ready to describe rule application on an abstract level. Intuitively, the rule $P \to_{\mathcal{R}} \{P_1, \ldots, P_k\}$ can be applied to the node x in the tree T if $P \sim T, x$, and its application yields the new tree T', which is obtained from T by adding new successor nodes of x and/or extending the node labels, as indicated by some P_i. This intuition is formalized in the following definition.

Definition 9 (Rule Application). *Let S be an admissible tableau system, $T = (V, E, n, \ell)$ be an S-tree, and $P \to_{\mathcal{R}} \{P_1, \ldots, P_k\}$ be a rule of S. The S-tree $T' = (V', E', n', \ell')$ is obtained from T by application of this rule to a node $x \in V$ iff the following holds:*

1. *$V \subseteq V'$;*
2. *$E' = E \cup \{(x, y) \mid y \in V' \setminus V\}$;*
3. *ℓ' extends ℓ, i.e., $\ell(y, z) = \ell'(y, z)$ for all $(y, z) \in E$;*
4. *$P \sim_\pi T, x$ for some bijection π;*
5. *$P_i \sim_{\pi'} T', x$ for some $i, 1 \le i \le k$ and bijection π' extending π;*[8]
6. *for all $y \in V$ with $y \notin \mathsf{ran}(\pi)$, we have $n(y) = n'(y)$.*

Thus, rule application may add some new successors of x, may extend the labels of the existing successors of x and of x itself, and otherwise leaves the

[7] This shows that we cannot replace $\precsim$ by $\preceq$ in the statement of Condition 2. In fact, we cannot be sure that the new successor introduced in P_0 has the same name as the new successor in P'.

[8] Note that Condition 1 in the definition of admissibility implies that P_i differs from P in that the root may have additional successors, and that the node labels may be larger. Thus, π' differs from π in that the additional successors of the root are mapped to the elements of $V' \setminus V$.

edge relation and the node and edge labels unchanged. For a fixed rule $P \to_{\mathcal{R}} \{P_1, \dots, P_k\}$, a fixed choice of P_i, and a fixed node x in T, the results of the rule application is unique up to the names of the new nodes in $V' \setminus V$. It is easy to check that, in the case of $S_{\mathcal{ALC}}$, rule application as defined above captures precisely the intuitive understanding of rule application employed in Section 2.

To finish our abstract definition of tableau algorithms, we need some way to describe the set of S-trees that can be obtained by starting with an initial S-tree for an input Γ, and then repeatedly applying completion rules. This leads to the notion of S-trees for Γ.

Definition 10 (S-tree for Γ). *Let S be an admissible tableau system, and let Γ be an input for S. The set of* S-trees for Γ *is the smallest set of S-trees such that*

1. *All* initial S-trees for Γ *belong to this set, where an initial S-tree for Γ is of the form*
$$(\{v_0\}, \emptyset, \{v_0 \mapsto \Lambda\}, \emptyset)$$
where v_0 is a node and $\Lambda \in \mathsf{ini}_S(\Gamma)$.
2. *If T is an S-tree for Γ and T' can be obtained from T by the application of a completion rule, then T' is an S-tree for Γ.*
3. *If $T_0, T_1, \dots$ is an infinite sequence of S-trees for Γ with $T_i = (V_i, E_i, n_i, \ell_i)$ such that*
   (a) T_0 is an initial S-tree for Γ and
   (b) for all $i \geq 0$, T_{i+1} can be obtained from T_i by the application of a completion rule,

   then the tree $T^\omega = (V, E, n, \ell)$ is also an S-tree for Γ, where
   - $V = \bigcup_{i \geq 0} V_i$,
   - $E = \bigcup_{i \geq 0} E_i$,
   - $n = \bigcup_{i \geq 0} n_i$, *and*
   - $\ell = \bigcup_{i \geq 0} \ell_i$.

Rule application may terminate after finitely many steps or continue forever. The last case of Definition 10 deals with such infinite sequences of rule applications. The S-tree T^ω can be viewed as the limit of the sequence of S-trees $T_0, T_1, \dots$ This limit exists since admissibility of S implies that rule application is monotonic, i.e., it extends S-trees by new nodes or by additional elements in node labels, but it never removes nodes or elements of node labels.

Let us now define when an S-tree is saturated and clash-free.

Definition 11 (Saturated, Clash-free). *Let S be an admissible tableau system. We say that the S-tree $T = (V, E, n, \ell)$ is*

- saturated *if, for every node x in T and every pattern P, $P \sim T, x$ implies $\mathcal{R}(P) = \emptyset$;*
- clash-free *if, for every node x in T and every $P \in \mathcal{C}$, we have $P \not\sim T, x$.*

Saturatedness says that no completion rule is applicable to the S-tree, and an S-tree is clash-free if no clash-trigger can be applied to any of its nodes.

Finally, we define soundness and completeness of tableau systems w.r.t. a certain property of its set of inputs. If the inputs are pairs consisting of a concept and a TBox, the property is usually satisfiability of the concept w.r.t. the TBox.

Definition 12 (Sound, Complete). *Let $\mathcal{P} \subseteq \mathfrak{I}$ be a property. The tableau system S is called*

- sound for $\mathcal{P}$ *iff, for any $\Gamma \in \mathfrak{I}$, the existence of a saturated and clash-free S-tree for Γ implies that $\Gamma \in \mathcal{P}$;*
- complete for $\mathcal{P}$ *iff, for any $\Gamma \in \mathcal{P}$, there exists a saturated and clash-free S-tree for Γ.*

It should be noted that the algorithmic treatment of tableau systems requires a stronger notion of completeness: an additional condition is needed to ensure that the out-degree of S-trees is appropriately bounded (see Definition 13 and Definition 20 below).

Taking into account the known soundness and completeness results for the $\mathcal{ALC}$ tableau algorithm described in Figure 1, it is straightforward to check that the tableau system $S_{\mathcal{ALC}}$ is sound and complete w.r.t. satisfiability of concepts w.r.t. TBoxes. Note, in particular, that saturated S-trees for an input Γ are precisely those S-trees for Γ that can be obtained by exhaustive or infinite and *fair* rule application.

3 ExpTime Automata-based Decision Procedures from Tableau Systems

In this section, we define the class of "ExpTime-admissible" tableau systems. If such a tableau system is sound and complete for a property $\mathcal{P}$, then it gives rise to an ExpTime algorithm for deciding $\mathcal{P}$.[9] In the case where $\mathcal{P}$ is satisfiability of description logic concepts w.r.t. a (general) TBox, this means that the mere existence of an ExpTime-admissible tableau system for the DL implies an ExpTime upper-bound for concept satisfiability w.r.t. (general) TBoxes in this DL. The ExpTime upper-bound is shown via a translation of the inputs of the ExpTime-admissible tableau system into certain automata working on *infinite* trees. For this reason, ExpTime-admissible tableau systems need *not* deal with the issue of termination. Indeed, non-terminating tableau algorithms such as the one for $\mathcal{ALC}$ with general TBoxes introduced in Section 2.1 may yield ExpTime-admissible tableau systems.

Throughout this section, we consider a fixed set of inputs $\mathfrak{I}$ and a fixed tableau system $S = (\mathsf{NL}, \mathsf{EL}, \cdot^S, \mathcal{R}, \mathcal{C})$ for $\mathfrak{I}$, which is sound and complete w.r.t. some property $\mathcal{P}$.[9] As usual, the exponential upper-bound of deciding $\mathcal{P}$ is assumed to be in the "size" of the input $\Gamma \in \mathfrak{I}$. Thus, we assume that the set

[9] More precisely, we must demand a slightly stronger version of completeness, as introduced in Definition 13 below.

of inputs is equipped with a size function, which assigns to an input $\Gamma \in \mathfrak{I}$ a natural number, its size $|\Gamma|$.

3.1 Basic Notions

Recall that a tableau system S is sound and complete for a property $\mathcal{P}$ if, for any input Γ, we have $\Gamma \in \mathcal{P}$ iff there exists a (potentially infinite) saturated and clash-free S-tree for Γ. The fundamental idea for obtaining an ExpTime upper-bound for deciding $\mathcal{P}$ is to use automata on infinite trees to check for the existence of a clash-free and saturated S-tree for a given input Γ. More precisely, each input Γ is converted into a tree automaton $\mathcal{A}_\Gamma$ such that there exists a clash-free and saturated S-tree for Γ iff $\mathcal{A}_\Gamma$ accepts a non-empty language. Since tree automata work on trees of some fixed out-degree, this approach only works if the (size of the) input determines such a fixed out-degree for the S-trees to be considered. This motivates the following definition.

Definition 13 (p-Complete). *Let p be a polynomial. The tableau system S is called* p-complete *for $\mathcal{P}$ iff, for any $\Gamma \in \mathcal{P}$, there exists a saturated and clash-free S-tree for Γ with out-degree bounded by $p(|\Gamma|)$.*

Throughout this section, we assume that there exists a polynomial p such that the fixed tableau system S is p-complete w.r.t. the property $\mathcal{P}$ under consideration.

The tableau system $S_{\mathcal{ALC}}$ defined in Section 2 is easily proved to be i-complete, with i being the identity function on the natural numbers: using the formulation of the rules, it is easily proved that the out-degree of every $S_{\mathcal{ALC}}$-tree for the input $(C, \mathcal{T})$ is bounded by the number of concepts of the form $\exists r.D$ in $\mathsf{sub}(C, \mathcal{T})$ and thus also by

$$|(C, \mathcal{T})| := |C| + \sum_{C_1 \sqsubseteq C_2 \in \mathcal{T}} (|\mathsf{nnf}(C_1 \rightarrow C_2)|),$$

where $|E|$ denotes the length of the concept E.

It should be noted that most standard description logic tableau algorithms also exploit p-completeness of the underlying logic: although this is not made explicit in the formulation of the algorithm itself, it is usually one of the central arguments in termination proofs. The intuition that p-completeness is *not* an artefact of using an automata-based approach is reinforced by the fact that a similar strengthening of completeness is needed in Section 4, where we construct tableau-based decision procedures from tableau systems.

To ensure that the automaton $\mathcal{A}_\Gamma$ can be computed and tested for emptiness in exponential time, we require the function $\cdot^S$ of the tableau system S and the rules of S to exhibit an "acceptable" computational behavior. This is captured by the following definition. In this definition, we assume that all patterns are appropriately encoded in some finite alphabet, and thus can be the input for a decision procedure. The *size of a pattern P* is the sum of the sizes of its node and edge labels, where the size of a node label is the sum of the sizes of its node label elements.

Definition 14 (ExpTime-admissible). *The tableau system S is called* ExpTime-admissible *iff the following conditions are satisfied:*

1. *S is admissible (see Definition 7);*
2. *$\mathsf{ini}_S(\Gamma)$ and $\mathsf{el}_S(\Gamma)$ can be computed in time exponential in $|\Gamma|$, and the size of each edge label in $\mathsf{el}_S(\Gamma)$ is polynomial in $|\Gamma|$;*
3. *the cardinality of $\mathsf{nle}_S(\Gamma)$ and the size of each node label element in $\mathsf{nle}_S(\Gamma)$ is polynomial in $|\Gamma|$, and $\mathsf{nle}_S(\Gamma)$ can be computed in time exponential in $|\Gamma|$;*
4. *for each pattern P it can be checked in time exponential in the size of P whether, for all patterns P', $P' \sim P$ implies $\mathcal{R}(P') = \emptyset$;*
5. *for each pattern P it can be checked in time exponential in the size of P whether there is a clash-trigger $P' \in \mathcal{C}$ such that $P' \sim P$.*

Note that Point 2 of ExpTime-admissibility implies that, for each $\Gamma \in \mathfrak{I}$, the cardinality of the sets $\mathsf{ini}_S(\Gamma)$ and $\mathsf{el}_S(\Gamma)$ are at most exponential in $|\Gamma|$. The cardinality of the set of node label elements $\mathsf{nle}_S(\Gamma)$ is explicitly required (in Point 3) to be polynomial. For the actual set of node labels (which are sets of node label elements), this yields an exponential upper-bound on its cardinality, but the size of each node label is polynomial in $|\Gamma|$. Since p-completeness implies that we consider only S-trees T of out-degree bounded by $p(|\Gamma|)$, and since the sizes of edge and node labels are polynomial in $|\Gamma|$, the size of each neighborhood T, x is polynomial in $|\Gamma|$. Thus, the fourth point ensures that the saturatedness condition can be checked in time exponential in $|\Gamma|$ for a given neighborhood T, x of T. The fifth point yields the same for clash-freeness.

Most standard description logic tableau algorithms for ExpTime-complete DLs trivially satisfy the conditions of ExpTime-admissibility. For example, it is easy to show that the tableau system $S_{\mathcal{ALC}}$ defined in Section 2 is ExpTime-admissible. We have already shown admissibility of $S_{\mathcal{ALC}}$, and Point 2 and 3 are immediate consequences of the definitions of $\mathsf{ini}_{S_{\mathcal{ALC}}}$, $\mathsf{nle}_{S_{\mathcal{ALC}}}$, and $\mathsf{el}_{S_{\mathcal{ALC}}}$. To see that Points 4 and 5 are satisfied as well, first note that the definition of the rules and clash-triggers in $S_{\mathcal{ALC}}$ is invariant under isomorphism of patterns. For this reason, the decision problem in Point 4 reduces to checking whether a given pattern P is saturated (see the definition of this notion below Definition 6), and the decision problem in Point 5 reduces to checking whether a given pattern is a clash-trigger. As an example, we consider the rule handling existential restrictions. Let $P = (V, E, n, \ell)$ be a pattern with root v_0, and assume that $\exists r.C \in n(v_0)$. This existential restriction contributes a set of patterns to $\mathcal{R}(P)$ iff $C \notin n(u)$ for all r-successors u of v_0. Obviously, this can be checked in time polynomial in the size of the pattern.

The remainder of the present section is concerned with converting tableau systems into automata-based decision procedures, as outlined above. The major challenge is to bring together the different philosophies underlying tableau algorithms and automata-based approaches for deciding concept satisfiability: tableau algorithm actively try to *construct* a model for the input by applying rules, as reflected in the Definitions 9 and 10, whereas automata are based on the concept of "acceptance" of a tree, i.e., they verifying whether a *given* tree actually describes a model. Of course, the emptiness test for the automaton then again checks whether such a tree exists. Due to these different perspectives, it is not straightforward to construct automata that directly check for the existence of S-trees for an input Γ. To overcome this problem, we first introduce the (less constructive) notion of S-trees *compatible with* Γ, and investigate the relationship of this notion to S-trees *for* Γ, as introduced in Definition 10.

Definition 15 (S-tree compatible with Γ). *Let Γ be an input and $T = (V, E, n, \ell)$ an S-tree with root v_0. Then T is* compatible with Γ *iff it satisfies the following conditions:*

1. $n(x) \subseteq \wp(\mathsf{nle}_S(\Gamma))$ *for each* $x \in V$*;*
2. $\ell(x, y) \in \mathsf{el}_S(\Gamma)$ *for each* $(x, y) \in E$*;*
3. *there exists* $\Lambda \in \mathsf{ini}_S(\Gamma)$ *such that* $\Lambda \subseteq n(v_0)$*;*
4. *the out-degree of* T *is bounded by* $p(|\Gamma|)$*.*

Below, we will show that, given an ExpTime-admissible tableau system S that is sound and p-complete for some property $\mathcal{P}$ and an input Γ for S, we can construct a looping tree automaton of size exponential in the size of Γ that accepts exactly the saturated and clash-free S-trees compatible with Γ. Since the emptiness problem for looping tree automata can be decided in time polynomial (actually, linear) in the size of the automaton, this shows that the existence of saturated and clash-free S-trees compatible with Γ can be decided in exponential time. Since S is sound and p-complete for $\mathcal{P}$, we have $\Gamma \in \mathcal{P}$ iff there is a saturated and clash-free S-tree *for* Γ. Thus, we must investigate the connection between S-trees for Γ and S-trees compatible with Γ. This is done in the next lemma.

In the proof of the lemma, we need sub-tree relations between S-trees in analogy to the inclusion relations "$\precsim$", "$\preceq$", and "$\prec$" between patterns introduced in Definition 6. These relations are defined on trees exactly as for patterns, and we also use the same relation symbols for them.

Lemma 1. *There exists a clash-free and saturated S-tree that is* compatible with Γ *iff there exists a clash-free and saturated S-tree* for Γ*.*

Proof. The "if" direction is straightforward: let $T = (V, E, n, \ell)$ be a clash-free and saturated S-tree for Γ. Since S is sound and p-complete for $\mathcal{P}$, we can w.l.o.g. assume that the out-degree of T is bounded by $p(|\Gamma|)$. It is not hard to show that T is compatible with Γ, i.e. satisfies Conditions 1 to 4 of Definition 15:

- Each initial S-tree satisfies Conditions 1 and 2 of compatibility, and Condition 3 of admissibility ensures that rule application adds only node label elements from $\mathsf{nle}_S(\Gamma)$ and edge labels from $\mathsf{el}_S(\Gamma)$.
- Each initial S-tree satisfies Condition 3 of compatibility, and rule application cannot delete elements from node labels.
- Since we assume the out-degree of T to be bounded by $p(|\Gamma|)$, Condition 4 of compatibility is also satisfied.

Now for the "only if" direction. Let $T = (V, E, n, \ell)$ be a clash-free and saturated S-tree with root v_0 that is compatible with Γ. To construct a clash-free and saturated S-tree for Γ, we first construct a (possibly infinite) sequence

$$T_1 \preceq T_2 \preceq T_3 \preceq \cdots$$

of S-trees for Γ such that $T_i \precsim_{\pi_i} T$ for all $i \geq 1$. The construction will be such that the injections π_i that yield $T_i \precsim T$ also build an increasing chain, i.e., π_{i+1} extends π_i for all $i \geq 1$. In the construction, we use a countably infinite set V' from which the nodes of the trees T_i are taken. We fix an arbitrary enumeration $x_0, x_1, \ldots$ of V', and write $x < y$ if $x \in V'$ occurs before $y \in V'$ in this enumeration. We then proceed as follows:

- Since T is compatible with Γ, there exists $\Lambda \in \mathsf{ini}_S(\Gamma)$ such that $\Lambda \subseteq n(v_0)$. Define T_1 to be the initial S-tree $(\{x_0\}, \emptyset, \{x_0 \mapsto \Lambda\}, \emptyset)$. Obviously, $T_1 \precsim_{\pi_1} T$ for $\pi_1 := \{x_0 \mapsto v_0\}$.
- Now, assume that $T_i \precsim_{\pi_i} T$ is already constructed. If T_i is saturated, then T_i is the last tree in the sequence. Otherwise, choose the least node x in T_i (w.r.t. the fixed ordering $<$ on V') such that $P \sim T_i, x$ for some pattern P that is not saturated, i.e. there exists a rule $P \rightarrow_{\mathcal{R}} \{P_1, \ldots, P_k\}$. Since $T_i \precsim_{\pi_i} T$, we have $P \precsim T, \pi_i(x)$. Since T is saturated, the pattern $T, \pi_i(x)$ is saturated. By Condition 2 of admissibility, we have $P_j \precsim T, \pi_i(x)$ for some j with $1 \leq j \leq k$. We apply the rule $P \rightarrow_{\mathcal{R}} \{P_1, \ldots, P_k\}$ to x in T_i such that $P_j \sim T_{i+1}, x$. If T_{i+1} contains new nodes, then they are taken from V'. Admissibility yields $T_i \preceq T_{i+1}$ and the fact that $P_j \precsim T, \pi_i(x)$ implies that we can define an injection π_{i+1} such that $T_{i+1} \precsim_{\pi_{i+1}} T$.

In the definition of the clash-free and saturated S-tree T^* for Γ, we distinguish two cases:

1. if the constructed sequence is finite and T_n is the last tree in the sequence, then set $T^* := T_n$;
2. otherwise, let T^* be the S-tree T^ω obtained from the sequence $T_1, T_2, \ldots$ as in Case 3 of Definition 10.

In both cases, T^* is obviously an S-tree for Γ by definition. In addition, we have $T^* \precsim_\pi T$ where π is the injection obtained as the union of the injections π_i for $i \geq 1$.

It remains to be shown that T^* is clash-free and saturated. We concentrate on the second case, where $T^* = T^\omega$, since the first case is similar, but simpler. Clash-freeness is an easy consequence of $T^* \precsim T$. In fact, by Condition 4 of admissibility, clash-freeness of T implies that $T^* \precsim T$ is also clash-free.

To show saturatedness of T^*, we must look at T^* and its relationship to the trees T_i in more detail. Since $T_i \preceq T^* \precsim T$ and the out-degree of T is bounded by $p(|\Gamma|)$, the out-degrees of the trees T_i and T^* are also bounded by $p(|\Gamma|)$. For a given node x of T^*, we consider its neighborhood T^*, x. Since the rules of S only add nodes or elements of node labels (see Condition 1 in the definition of admissibility), and since the out-degree of x is bounded by $p(|\Gamma|)$ and the set

$\mathsf{nle}_S(\Gamma)$ is finite, there is an i such that x is a node of T_i and "the neighborhood of x does not change after step i," i.e., $T_i, x = T_{i+1}, x = \ldots = T^*, x$.

Now assume that T^* is not saturated, i.e., there exists a node x in T^* to which a rule applies, i.e., $P \sim T^*, x$ for some pattern P with $\mathcal{R}(P) \neq \emptyset$. Let i be such that $T_i, x = T_{i+1}, x = \ldots = T^*, x$. Thus, for $j \geq i$, a rule applies to the node x in T_i. In the construction of the sequence $T_1, T_2, T_3, \ldots$, we apply a rule only to the least node to which a rule is applicable. Consequently, from the ith step on, we only apply rules to nodes $y \leq x$. Since there are only finitely many such nodes (see the definition of the order $<$ above), there is one node $y \leq x$ to which rules are applied infinitely often. However, each rule application strictly increases the number of successors of y, or the label of y or of one of its successors. This contradicts the fact that the out-degree of y in the trees T_i is bounded by $p(|\Gamma|)$ and all node labels are subsets of the finite set $\mathsf{nle}_S(\Gamma)$. $\square$

3.2 Accepting Compatible S-trees Using Looping Automata

Recall that we assume our tableau system S to be sound and p-complete w.r.t. a property $\mathcal{P}$. By Lemma 1, to check whether an input has property $\mathcal{P}$, it thus suffices to verify the existence of a saturated and clash-free S-tree that is compatible with Γ. In this section, we show how this can be done using an automata-based approach.

As usual, the automata work on k-ary infinite trees (for some fixed natural number k) whose nodes are labeled by elements of a finite label set and whose edges are ordered, i.e., we can talk about the i-th son of a node. To be more precise, let M be a set and $k \geq 1$. A *k-ary M-tree* is a mapping $T : \{1, \ldots, k\}^* \to M$ that labels each node $\alpha \in \{1, \ldots, k\}^*$ with $T(\alpha) \in M$. Intuitively, the node αi is the i-th child of α. We use ϵ to denote the empty word, corresponding to the root of the tree.

Definition 16 (Looping Tree Automata). *A* looping tree automaton $\mathcal{A} = (Q, M, I, \Delta)$ *working on k-ary M-trees consists of a finite set Q of states, a finite alphabet M, a set $I \subseteq Q$ of initial states, and a transition relation $\Delta \subseteq Q \times M \times Q^k$.*

A run *of $\mathcal{A}$ on an M-tree T is a mapping $R : \{1, \ldots, k\}^* \to Q$ such that $R(\epsilon) \in I$ and*

$$(R(\alpha), T(\alpha), R(\alpha 1), \ldots, R(\alpha k)) \in \Delta$$

for each $\alpha \in \{1, \ldots, k\}^*$. The language *of k-ary M-trees* accepted by *$\mathcal{A}$ is*

$$L(\mathcal{A}) := \{T \mid \textit{there is a run of } \mathcal{A} \textit{ on the } k\textit{-ary } M\textit{-tree } T\}.$$

Note that, in contrast to the S-trees considered above, the trees defined here are infinite trees of a *fixed* arity k, where edges are not labeled, but ordered. It is, however, not hard to convert S-trees compatible with a given input into k-ary M-trees for appropriate k and M. This is achieved by (i) "padding" with additional dummy nodes, and (ii) representing edge labels via node labels.

Definition 17 (Padding). *Let $\Gamma \in \mathfrak{I}$ be an input and $T = (V, E, n, \ell)$ an S-tree with root $v_0 \in V$ that is compatible with Γ. For each $x \in V$, we use $d(x)$ to denote the out-degree of x in T. We assume that the successors of each node $x \in V$ are linearly ordered and that, for each node $x \in V \setminus \{v_0\}$, $s(x) = i$ iff x is the i-th successor of its predecessor. We inductively define a function m from $\{1, \ldots, p(|\Gamma|)\}^*$ to $V \cup \{\sharp\}$ (where $\sharp \notin V$) as follows:*

- $m(\epsilon) = v_0$;
- *if* $m(\alpha) = x$, $(x, y) \in E$, *and* $s(y) = i$, *then* $m(\alpha i) = y$;
- *if* $m(\alpha) = x$ *and* $d(x) < i$, *then* $m(\alpha i) = \sharp$;
- *if* $m(\alpha) = \sharp$, *then* $m(\alpha i) = \sharp$ *for all* $i \in \{1, \ldots, p(|\Gamma|)\}$.

Let $\mathsf{tl}_S(\Gamma)$ denote the set $(\wp(\mathsf{nle}_S(\Gamma)) \times \mathsf{el}_S(\Gamma)) \cup \{(\sharp, \sharp)\}$. The padding *$P_T$ of T is the $p(|\Gamma|)$-ary $\mathsf{tl}_S(\Gamma)$-tree defined by setting*

1. $P_T(\epsilon) = (n(v_0), e_0)$ *where e_0 is an arbitrary (but fixed) element of $\mathsf{el}_S(\Gamma)$;*
2. $P_T(\alpha) = (n(x), \Theta)$ *if $\alpha \neq \epsilon$, $m(\alpha) = x \neq \sharp$, and $\ell(y, x) = \Theta$ where y is the (unique) predecessor of x in T;*
3. $P_T(\alpha) = (\sharp, \sharp)$ *if* $m(\alpha) = \sharp$.

We now define, for each input $\Gamma \in \mathfrak{I}$, a looping automaton $\mathcal{A}_\Gamma$ that accepts a non-empty language iff there exists a saturated and clash-free S-tree that is compatible with Γ.

Definition 18 (Automaton for Input Γ). *Let $\Gamma \in \mathfrak{I}$ be an input and $h = p(|\Gamma|)$. The automaton $\mathcal{A}_\Gamma$ is defined as follows:*

- $Q := M := \mathsf{tl}_S(\Gamma)$;
- $I := \{(\Psi, e_0) \mid \Lambda \subseteq \Psi$ *for some* $\Lambda \in \mathsf{ini}_S(\Gamma)\}$;
- $((\Lambda_0, \Theta_0), (\Lambda, \Theta), (\Lambda_1, \Theta_1), \ldots, (\Lambda_h, \Theta_h)) \in \Delta$ *iff the following two conditions are satisfied:*
  1. $(\Lambda_0, \Theta_0) = (\Lambda, \Theta)$;
  2. *either* $\Lambda_0 = \Lambda_1 = \cdots = \Lambda_h = \sharp$,
     or there is a $0 \leq k \leq h$ *such that* $\Lambda_0, \ldots, \Lambda_k$ *differ from* $\sharp$, $\Lambda_{k+1} = \cdots = \Lambda_h = \sharp$, *and the pattern* $P^* = (V^*, E^*, n^*, \ell^*)$ *defined as*
     - $V^* := \{i \mid 0 \leq i \leq k\}$,
     - $E^* := \{(0, i) \mid i \in V^* \setminus \{0\}\}$,
     - $n^* = \{i \mapsto \Lambda_i \mid i \in V^*\}$, *and*
     - $\ell^* := \{(0, i) \mapsto \Theta_i \mid i \in V^* \setminus \{0\}\}$

     satisfies the following conditions:

     (a) for each pattern P with $P \sim P^*$, P is saturated (i.e. $\mathcal{R}(P) = \emptyset$);
     (b) for each pattern $P \in \mathcal{C}$, we have $P \not\sim P^*$.

The following lemma shows that the automaton $\mathcal{A}_\Gamma$ accepts exactly the paddings of saturated and clash-free S-trees compatible with Γ. Consequently, it accepts a non-empty set of trees iff there exists a saturated and clash-free S-trees compatible with Γ.

Lemma 2. *Let $\Gamma \in \Im$ be an input. Then*

$$L(\mathcal{A}_\Gamma) = \{P_T \mid T \text{ is a saturated and clash-free } S\text{-trees compatible with } \Gamma\}.$$

Proof. First, assume that T is a saturated and clash-free S-trees compatible with Γ. We claim that P_T itself is a run of $\mathcal{A}_\Gamma$ on P_T. In fact, $P_T(\epsilon) \in I$ is an immediate consequence of the definition of padding and Condition 3 in the definition of S-trees compatible with Γ. Now, consider some node α of P_T. The first condition in the definition of Δ is satisfied since we have P_T as run on itself. Thus, consider the second condition. If $P_T(\alpha) = (\sharp, \sharp)$, then the definition of padding implies that all the successor nodes of α also have label $(\sharp, \sharp)$, and thus the second condition in the definition of Δ is satisfied. Otherwise, it is easy to see that the pattern P^* defined in the second condition in the definition of Δ is also a pattern in T. Since T is saturated and clash-free, P^* thus satisfies (a) and (b) in the second condition in the definition of Δ. This completes the proof that P_T is a run of $\mathcal{A}_\Gamma$ on P_T, and thus shows that $P_T \in L(\mathcal{A}_\Gamma)$.

Second, assume that $\widehat{T}$ is a tree accepted by $\mathcal{A}_\Gamma$. Because of the first condition in the definition of Δ, $\widehat{T}$ itself is a run of $\mathcal{A}_\Gamma$ on $\widehat{T}$. The definitions of Q, I, and Δ imply that there is an S-tree T compatible with Γ such that $P_T = \widehat{T}$. This tree can be obtained from $\widehat{T}$ by removing all the padding. It remains to be shown that T is saturated and clash-free. Thus, consider a node x of T, and let α be the corresponding node in $P_T = \widehat{T}$. Since x is a node in T, the node α has a label different from $(\sharp, \sharp)$. Let us now consider the transition from α to its successor nodes. It is easy to see that the pattern P^* defined in the second condition in the definition of the transition relation coincides with P, x. Thus (a) and (b) in this condition imply that no rule and no clash-trigger is applicable to x. □

We are now ready to prove the main result of this section: the ExpTime upper-bound induced by ExpTime-admissible tableau systems.

Theorem 1. *Let $\Im$ be a set of inputs, $\mathcal{P} \subseteq \Im$ a property, and p a polynomial. If there exists an ExpTime-admissible tableau system S for $\Im$ that is sound and p-complete for $\mathcal{P}$, then $\mathcal{P}$ is decidable in ExpTime.*

Proof. Let $\Gamma \in \Im$ be an input. To decide whether $\Gamma \in \mathcal{P}$, we construct the automaton $\mathcal{A}_\Gamma$ and then check whether it accepts a non-empty language. By Lemmas 1 and 2, this algorithm is correct. Thus, it remains to be shown that it can be executed in exponential time. To see that the automaton $\mathcal{A}_\Gamma$ can be constructed in time exponential in $|\Gamma|$, note that, by Conditions 2 and 3 of ExpTime-admissibility, we can compute $\wp(\mathsf{nle}_S(\Gamma))$ and $\mathsf{el}_S(\Gamma)$ in time exponential in $|\Gamma|$, and thus the same holds for $\mathsf{tl}_S(\Gamma) = Q = M$, and I. The transition relation Δ can be computed in exponential time due to the Conditions 4 and 5 of ExpTime-admissibility and the fact that p is a polynomial. Since the automaton can be computed in exponential time, its size is at most exponential in $|\Gamma|$. Thus, it remains to note that the emptiness test for looping tree automata can be realized in polynomial time [29]. □

Since we have shown that the tableau system $S_{\mathcal{ALC}}$ is ExpTime-admissible as well as sound and p-complete (for some polynomial p) for satisfiability of $\mathcal{ALC}$-concepts w.r.t. (general) TBoxes, we can immediately put Theorem 1 to work:

Corollary 1. *$\mathcal{ALC}$-concept satisfiability w.r.t. TBoxes is in ExpTime.*

4 Tableau-based Decision Procedures from Tableau Systems

The tableau systems described in Section 2.2 cannot immediately be used as tableau-based decision procedures since rule application need not terminate. The purpose of this section is to show that, under certain natural conditions, the addition of a straightforward cycle detection mechanism turns them into (terminating) decision procedures. The resulting procedures are structurally similar to standard tableau-based algorithms for description logics, such as the ones underlying systems like FaCT and RACER. In contrast to the ExpTime algorithm constructed in the previous section, the procedures obtained here are usually not worst-case optimal—a price we have to pay for more easily implementable and optimizable decision procedures.

Fix a set of inputs $\mathfrak{I}$ and a tableau system $S = (\mathsf{NLE}, \mathsf{EL}, \cdot^S, \mathcal{R}, \mathcal{C})$ for $\mathfrak{I}$. As in the previous section, we require that S has a number of computational properties. Since we do not consider complexity issues in this section, it is sufficient for our purposes to impose effectiveness (and not efficiency) constraints. We start with modifying Definition 14:

Definition 19 (Recursive Tableau System). *S is called* recursive *iff the following conditions are satisfied:*

1. *S is admissible (see Definition 7);*
2. *$\mathsf{ini}_S(\Gamma)$ can be computed effectively;*
3. *for each pattern P it can be checked effectively whether, for all patterns P', $P' \sim P$ implies $\mathcal{R}(P') = \emptyset$; if this is not the case, then we can effectively determine a rule*
$$P' \to_{\mathcal{R}} \{P_1, \ldots, P_k\}$$
and a bijection π such that $P' \sim_\pi P$.
4. *for each pattern P it can be checked effectively whether there is a clash-trigger $P' \in \mathcal{C}$ such that $P' \sim P$.*

The main difference between this definition and Definition 14 is Condition 3, which now requires that, besides checking the applicability of rules, we can effectively apply at least one rule whenever some rule is applicable at all. Another difference is that we do not actually need to compute the sets $\mathsf{el}_S(\Gamma)$ and $\mathsf{nle}_S(\Gamma)$ in order to apply rules.

Analogously to the case of ExpTime-admissibility, it can be verified that the tableau system $S_{\mathcal{ALC}}$ is recursive. In particular, for the second part of Condition 3 we can again use the fact that the rules of $S_{\mathcal{ALC}}$ are invariant under isomorphism

Preconditions: Let $\mathfrak{I}$ be a set of inputs, $\mathcal{P} \subseteq \mathfrak{I}$ a property, f a recursive function, and S a recursive tableau system for $\mathfrak{I}$ that is sound and f-complete for $\mathcal{P}$.

Algorithm: Return true on input $\Gamma \in \mathfrak{I}$ if the procedure tableau(T) defined below returns true for at least one initial S-tree T for Γ. Otherwise return false.

procedure tableau(T)

if $P \sim T, x$ for some $P \in \mathcal{C}$ and node x in T or the out-degree of T exceeds $f(|\Gamma|)$,
  then return false

if no rule is applicable to a non-blocked node x in T,
  then return true

take a a non-blocked node x in T and a rule $P \to_{\mathcal{R}} \{P_1, \ldots, P_k\}$ with $P \sim T, x$,

let T_i be the result of applying the above rule such that $P_i \sim T_i, x$, for $1 \leq i \leq k$;

if at least one of tableau(T_1), tableau(T_2), ..., tableau(T_k) returns true
  then return true

return false

Fig. 2. Decision procedure for $\mathcal{P}$.

of patterns: this means that it suffices to compute, for a given non-saturated pattern P, a set of patterns $\{P_1, \ldots, P_k\}$ such that $P \to_{\mathcal{R}} \{P_1, \ldots, P_k\}$. It is easy to see that this can be effectively done for the rules of $S_{\mathcal{ALC}}$.

We now define a more relaxed variant of Definition 13.

Definition 20 (f-complete). *Let $f : \mathbb{N} \to \mathbb{N}$ be a recursive function. The tableau system S is called* f-complete *for $\mathcal{P}$ iff, for any $\Gamma \in \mathcal{P}$, there exists a saturated and clash-free S-tree for Γ with out-degree bounded by $f(|\Gamma|)$.*

Since we have already shown that $S_{\mathcal{ALC}}$ is p-complete for some polynomial p, $S_{\mathcal{ALC}}$ is clearly f-complete for the (computable) function f induced by the polynomial p.

In order to implement a cycle detection mechanism, we introduce the notion of blocking: given an S-tree $T = (V, E, n, \ell)$, we denote by E^* the transitive and reflexive closure of E and say that $x \in V$ is *blocked* iff there exist distinct $u, v \in V$ such that uE^*x, vE^*x, and $n(u) = n(v)$. Note that this corresponds to the well-known "equality-blocking" technique that is used in various DL tableau algorithms [17, 4].

The tableau-based decision procedure for $\mathcal{P}$ induced by the tableau system S is described in Figure 2. Note that the selection of rules and nodes in the "else" part of the procedure tableau is "don't care" non-deterministic: for the soundness and completeness of the algorithm, it does not matter which rule we apply when to which node.

Let us verify that the individual steps performed by the algorithm in Figure 2 are actually effective:

- the initial trees for an input Γ can be computed effectively, since $\mathsf{ini}_S(\Gamma)$ can be computed effectively by Condition 2 of Definition 19;

– the condition in the first "if" statement can be checked effectively by Condition 4 of Definition 19 and since f is a recursive function;
– the applicability of rules can be checked by the first part of Condition 3 of Definition 19;
– finally, that we can effectively take a rule and apply it to a node x follows from the second part of Condition 3 of Definition 19.

We now turn to termination, soundness, and completeness of the algorithm.

Lemma 3 (Termination). *Suppose the preconditions of Figure 2 are satisfied. Then the algorithm of Figure 2 terminates for any input $\Gamma \in \mathfrak{I}$.*

Proof. Let $\Gamma \in \mathfrak{I}$. The number of initial trees for Γ is finite and can be computed effectively. Hence, it is sufficient to show that the procedure $\mathsf{tableau}$ terminates on any initial tree for Γ. For each step in which the procedure does not immediately return true or false, a node is added to the tree or $n(x)$ properly increases for some node x (due to Condition 1 of admissibility). Hence, since $n(x) \subseteq \wp(\mathsf{nle}_S(\Gamma))$ for any node x and any tree constructed during a run of $\mathsf{tableau}$, it is sufficient to show that both the out-degree and the depth of the trees constructed is bounded. But the out-degree of the trees is bounded by $f(|\Gamma|)$ (more precisely, as soon as one rule application yields a tree with out-degree larger than $f(|\Gamma|)$, the algorithm returns false in the next step) and the length of E-paths does not exceed $2^{|\mathsf{nle}_S(\Gamma)|}$ since rules are not applied to blocked nodes. ☐

Lemma 4 (Soundness). *Suppose the preconditions of Figure 2 are satisfied. If the algorithm of Figure 2 returns* true *on input Γ, then $\Gamma \in \mathcal{P}$.*

Proof. Suppose the algorithm returns true on input Γ. Then the algorithm terminates with a clash-free S-tree $T = (V, E, n, \ell)$ whose out-degree does not exceed $f(|\Gamma|)$ and such that no rule is applicable to a non-blocked node in T. As S is sound for $\mathcal{P}$, it is sufficient to show that there exists a saturated and clash-free S-tree for Γ. To this end we construct a clash-free and saturated S-tree

$$T' = (V', E', n', \ell')$$

which is compatible with Γ (from which, by Lemma 1, we obtain a clash-free and saturated S-tree for Γ). Say that a node $x \in V$ is *directly blocked* if it is blocked but its predecessor is not blocked. If y is the (uniquely determined) node $y \neq x$ with yE^*x and $n(x) = n(y)$, then y is said to *block* x.

Now, V' consists of all non-empty sequences $\langle v_0, x_1, \ldots, x_n \rangle$, where v_0 is the root of V, the $x_1, \ldots, x_n \in V$ are directly blocked or not blocked, and $(x_i, x_{i+1}) \in E$ if x_i is not blocked or x_i is blocked by some $y \in V$ such that $(y, x_{i+1}) \in E$. Define E' by setting, for $\boldsymbol{x} = \langle v_0, x_1, \ldots, x_n \rangle \in V'$ and $\boldsymbol{y} \in V'$, $(\boldsymbol{x}, \boldsymbol{y}) \in E'$ iff there exists x_{n+1} such that $\boldsymbol{y} = \langle v_0, x_1, \ldots, x_n, x_{n+1} \rangle$. Define n' by setting $n'(\langle v_0, x_1, \ldots, x_n \rangle) = n(x_n)$. Finally, define ℓ' by

– $\ell'(\langle v_0, x_1, \ldots, x_n \rangle, \langle v_0, x_1, \ldots, x_n, x_{n+1} \rangle) = \ell(x_n, x_{n+1})$ if x_n is not blocked;
– $\ell'(\langle v_0, x_1, \ldots, x_n \rangle, \langle v_0, x_1, \ldots, x_n, x_{n+1} \rangle) = \ell(y, x_{n+1})$ if x_n is blocked and y blocks x_n.

We show that T' is a clash-free and saturated S-tree which is compatible with Γ. Compatibility is readily checked using the definition of T'. Since T is clash-free and no rule is applicable to a non-blocked node of T, we can prove clash-freeness and saturatedness of T' by showing that any S-pattern P that matches $T', \boldsymbol{x}$ for some node $\boldsymbol{x}$ in T' also matches a T, x for some non-blocked node x in T. Thus, assume that $P \sim_\tau T', \langle v_0, x_1, \ldots, x_n \rangle$, for some bijection τ. If x_n is not blocked, then $P \sim_{\tau'} T, x_n$, where τ' is obtained from τ by composing τ with the mapping that assigns x_n to $\langle v_0, x_1, \ldots, x_n \rangle$ and x_{n+1} to each successor $\langle v_0, x_1, \ldots, x_n, x_{n+1} \rangle$ of $\langle v_0, x_1, \ldots, x_n \rangle$. Similarly, if x_n is blocked by y, then $P \sim_{\tau'} T, y$, where τ' is obtained from τ by composing τ with the mapping that assigns y to $\langle v_0, x_1, \ldots, x_n \rangle$ and x_{n+1} to each successor $\langle v_0, x_1, \ldots, x_n, x_{n+1} \rangle$ of $\langle v_0, x_1, \ldots, x_n \rangle$. □

Lemma 5 (Completeness). *Suppose the preconditions of Figure 2 are satisfied. If $\Gamma \in \mathcal{P}$, then the algorithm of Figure 2 returns* true *on input Γ.*

Proof. Suppose $\Gamma \in \mathcal{P}$. Since S is f-complete for $\mathcal{P}$, there exists a clash-free and saturated S-tree $T = (V, E, n, \ell)$ for Γ whose out-degree does not exceed $f(|\Gamma|)$. We use T to "guide" the algorithm to an S-tree of out-degree at most $f(|\Gamma|)$ in which no clash-trigger applies and no rule is applicable to a non-blocked node. This will be done in a way such that all constructed S-trees T' satisfy $T' \precsim T$.

For the start, we need to choose an appropriate initial S-tree T_1. Let v_0 be the root of T. Since S-trees for Γ are also compatible with Γ, the definition of compatibility implies that there exists $\Lambda \in \mathsf{ini}_S(\Gamma)$ such that $\Lambda \subseteq n(v_0)$. Define T_1 to be the initial S-tree $(\{v_0\}, \emptyset, \{v_0 \mapsto \Lambda\}, \emptyset)$. Clearly, $T_1 \precsim T$. We start the procedure tableau with the tree T_1.

Now suppose that tableau is called with some S-tree T' such that $T' \precsim T$. If no rule is applicable to a non-blocked node in T', we are done: since $T' \precsim T$ and T is clash-free and of out-degree at most $f(|\Gamma|)$, the same holds for T'. Now suppose that a rule is applicable to a non-blocked node in T'. Assume that the tableau procedure has chosen the rule $P \rightarrow_{\mathcal{R}} \{P_1, \ldots, P_k\}$ with $P \sim T', x$. Since $T' \precsim_\tau T$ for some τ, we have $P \precsim T, \tau(x)$. Since T is saturated, $T, \tau(x)$ is saturated. By Condition 2 of admissibility, we have $P_j \precsim T, \tau(x)$ for some $j, 1 \leq j \leq k$. So we "guide" the tableau procedure to continue exploring the S-tree T'_j obtained from T' by applying the rule $P \rightarrow_{\mathcal{R}} \{P_1, \ldots, P_k\}$ such that $P_j \sim T'_j, x$. Now, $P_j \precsim T, \tau(x)$ implies $T'_j \precsim T$.

Since the tableau procedure terminates on any input, the "guidance" process will also terminate and thus succeeds in finding an S-tree of out-degree at most $f(|\Gamma|)$ in which no clash-trigger applies and no rule is applicable to a non-blocked node. Hence, $\mathsf{tableau}(T_1)$ returns true. □

The three lemmas just proved imply that we have succeeded in converting the tableau system S into a decision procedure for $\mathcal{P}$.

Theorem 2. *Suppose the preconditions of Figure 2 are satisfied. Then the algorithm of Figure 2 effectively decides $\mathcal{P}$.*

5 A Tableau System for $\mathcal{ALCQI}$

As an example for a more expressive DL that can be treated within our framework, we consider the DL $\mathcal{ALCQI}$, which extends $\mathcal{ALC}$ with qualifying number restrictions and inverse roles. Qualified number restrictions ($(\geqslant m\, r.C)$ and $(\leqslant m\, r.C)$) can be used to state constraints on the number of r-successors belonging to a given concept C, and the inverse roles allow us to use both a role r and its inverse r^- when building a complex concept.

Definition 21 ($\mathcal{ALCQI}$ Syntax and Semantics). *Let $\mathsf{N_C}$ and $\mathsf{N_R}$ be pairwise disjoint and countably infinite sets of concept and role names. The set of $\mathcal{ALCQI}$-*roles *is defined as $\mathsf{ROL}_{\mathcal{ALCQI}} := \mathsf{N_R} \cup \{r^- \mid r \in \mathsf{N_R}\}$.*

The set of $\mathcal{ALCQI}$-concepts $\mathsf{CON}_{\mathcal{ALCQI}}$ is the smallest set such that

- *every concept name is a concept, and*
- *if C and D are $\mathcal{ALCQI}$-concepts and $r \in \mathsf{ROL}_{\mathcal{ALCQI}}$ is a role, then $\neg C, C \sqcap D, C \sqcup D, (\leqslant m\, r.C)$ and $(\geqslant m\, r.C)$ are also $\mathcal{ALCQI}$-concepts.*

TBoxes are defined as in the case of $\mathcal{ALC}$, i.e., they are finite sets of GCIs $C \sqsubseteq D$ where $C, D \in \mathsf{CON}_{\mathcal{ALCQI}}$.

The semantics of $\mathcal{ALCQI}$ is defined as for $\mathcal{ALC}$, where the additional constructors are interpreted as follows:

$$
\begin{aligned}
(r^-)^{\mathcal{I}} &:= \{(y, x) \mid (x, y) \in r^{\mathcal{I}}\}, \\
(\leqslant m\, r.C)^{\mathcal{I}} &:= \{d \in \Delta^{\mathcal{I}} \mid \#\{y \mid (d, y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \leq m\}, \\
(\geqslant m\, r.C)^{\mathcal{I}} &:= \{d \in \Delta^{\mathcal{I}} \mid \#\{y \mid (d, y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \geq m\},
\end{aligned}
$$

where $\#S$ denotes the cardinality of the set S.

Although the constructors $\exists r.C$ and $\forall r.C$ are not explicitly present in $\mathcal{ALCQI}$, they can be simulated by $(\geqslant 1\, r.C)$ and $(\leqslant 0\, r.\neg C)$, respectively. Thus, $\mathcal{ALCQI}$ really extends $\mathcal{ALC}$.

The definition of negation normal form (and thus of the function nnf) can easily be extended to $\mathcal{ALCQI}$ (see, e.g., [14]), and the same is true for the function sub. In addition, we define the *closure* of the $\mathcal{ALCQI}$ concept C and the TBox $\mathcal{T}$ as

$$
\mathsf{cl}(C, \mathcal{T}) := sub(C, \mathcal{T}) \cup \{\mathsf{nnf}(\neg D) \mid D \in sub(C, \mathcal{T})\}.
$$

In order to simplify the treatment of inverse roles, we denote *the inverse* of the $\mathcal{ALCQI}$-role r by $\overline{r}$, i.e., $\overline{r} = r^-$ if r is a role name, and $\overline{r} = s$ if $r = s^-$ for a role name s.

Completion trees for $\mathcal{ALCQI}$ look like completion trees for $\mathcal{ALC}$, with the only difference that edges may also be labeled with inverse roles. In fact, to handle a number restriction of the form $(\geqslant 1\, r^-.C)$ (which corresponds to the existential restriction $\exists r^-.C$ in $\mathcal{ALC}$) in the label of the node v, the tableau algorithm may introduce an r^--successor of v.

The fact that edges can also be labeled by inverse roles complicates the treatment of number restrictions by tableau rules. For a given node, we need to count the number of other nodes it is related to via the role r. Without inverse roles, this is quite easy: we just take the direct successors reached by an edge labeled with r. With inverse roles, we must also count the direct predecessor if the corresponding edge is labeled with $\overline{r}$. However, our framework only allows for patterns of depth one, and thus the rules cannot simultaneously look at a node together with its direct predecessor *and* its direct successors. To overcome this problem, we introduce new concept names $M_{r,C}$ for every pair r, C appearing in a number restriction $(\geqslant m\, r.C)$ or $(\leqslant m\, r.C)$. The intuitive meaning of these "marker concepts" is the following: if the label of node v contains $M_{r,C}$, then the edge leading to v from its direct predecessor u is labeled with $\overline{r}$ and the label of u contains C. Since the root node of the tree does not have a predecessor, these concepts are not allowed to appear in the root node. We will ensure this by enforcing that the root label contains $\neg M_{r,C}$. Given a concept C and a TBox $\mathcal{T}$, the set of necessary marker concepts is

$$\mathsf{M}_{\mathsf{C},\mathcal{T}} := \{M_{r,C}, \neg M_{r,C} \mid \{(\geqslant m\, r.C), (\leqslant m\, r.C)\} \cap \mathsf{cl}(C,\mathcal{T}) \neq \emptyset \text{ for some } m\}.$$

Definition 22 ($S_{\mathcal{ALCQI}}$). *The tableau system $S_{\mathcal{ALCQI}}$ is defined as follows: $\mathsf{NLE} := \mathsf{CON}_{\mathcal{ALCQI}}$ is the set of all $\mathcal{ALCQI}$-concepts, $\mathsf{EL} := \mathsf{ROL}_{\mathcal{ALCQI}}$ is the set of all $\mathcal{ALCQI}$-roles, and the function $\cdot^{S_{\mathcal{ALCQI}}}$ assigns to any input pair $(C, \mathcal{T})$ the following tuple $(\mathsf{nle}_{S_{\mathcal{ALCQI}}}, \mathsf{el}_{S_{\mathcal{ALCQI}}}, \mathsf{ini}_{S_{\mathcal{ALCQI}}})$:*

$$\begin{aligned}
\mathsf{nle}_{S_{\mathcal{ALCQI}}}(C,\mathcal{T}) &:= \mathsf{cl}(C,\mathcal{T}) \cup \mathsf{M}_{\mathsf{C},\mathcal{T}},\\
\mathsf{el}_{S_{\mathcal{ALCQI}}}(C,\mathcal{T}) &:= \{r, r^- \mid r \in \mathsf{N_R} \textit{ occurs in } C \textit{ or } \mathcal{T}\},\\
\mathsf{ini}_{S_{\mathcal{ALCQI}}}(C,\mathcal{T}) &:= \{\{\mathsf{nnf}(C)\} \cup \{\neg M_{r,C} \mid \neg M_{r,C} \in \mathsf{M}_{\mathsf{C},\mathcal{T}}\}\}.
\end{aligned}$$

The rules and clash-triggers of $S_{\mathcal{ALCQI}}$ are introduced in the next two definitions.

In order to define the rules and clash-triggers, we need to count the r-neighbors of a given node v in an $S_{\mathcal{ALCQI}}$-tree, i.e., the nodes that are either r-successors of v or the (unique) predecessor of v in case v is an $\overline{r}$-successor of this predecessor. The problem is that we must do this in a pattern, where the predecessor is not explicitly present. Instead, we use the presence of the marker concepts $M_{r,D}$ in the label of v.[10] Let P be a pattern with root v_0. We say that v_0 *has k r-neighbors containing D* iff

- either $M_{r,D}$ is not in the label of v_0 and v_0 has exactly k r-successors whose labels contain D;
- or $M_{r,D}$ is in the label of v_0 and v_0 has exactly $k-1$ r-successors whose labels contain D.

[10] The rules and clash-triggers are defined such that the presence of the marker concept $M_{r,D}$ in the label of a node v in a saturated and clash-free $S_{\mathcal{ALCQI}}$-tree implies that v is an $\overline{r}$-successor of its father w and that the label of w contains D.

In addition to the rules handling conjunctions, disjunctions, and the TBox axioms, $S_{\mathcal{ALCQI}}$ has three rules[11] that treat number restrictions. Before introducing them formally, we give brief intuitive explanations of these rules:

R$\geqslant$ To satisfy an at-least restriction $({\geqslant}m\, r.C)$, the rule creates the necessary neighbors one-by-one. In a single step, it adds $M_{r,C}$ to the label of the root of the pattern, or it adds C to the label of an existing r-successor of the root, or it creates a new r-successor of the root with label $\{C\}$.

RC If the root label of the pattern contains the at-most restriction $({\leqslant}m\, r.C)$, then this so-called *choose-rule* adds either the concept C or the concept $\mathsf{nnf}(\neg C)$ to the label of all r-successors of the root. In addition, this rule also takes the (not explicitly present) predecessor node into account by "guessing" whether the given node is an $\overline{r}$-successor of its predecessor and whether the label of this predecessor contains C. This is done by adding either $M_{r,C}$ or $\neg M_{r,C}$ to the label of the root.

R$\uparrow$ This rule propagates the information contained in the marker concepts to the predecessor node, i.e., if the label of an $\overline{r}$-successor of the root contains $M_{r,C}$, then we add C to the label of the root; if the label of an $\overline{r}$-successor of the root contains $\neg M_{r,C}$, then we add $\neg C$ to the label of the root.

Definition 23 (The rules of $S_{\mathcal{ALCQI}}$). *Let $P = (V, E, n, \ell)$ be a pattern with root v_0. Then $\mathcal{R}(P)$ is the smallest set of finite sets of patterns that contains all the sets of patterns required by the* R$\sqcap$, R$\sqcup$, *and* R$\mathcal{T}$ *rules, and in addition the following sets:*

R$\geqslant_1$ *if the root label $n(v_0)$ contains the concept $({\geqslant}m\, r.C)$ as well as the concept $M_{r,C}$, and there are less than $m-1$ nodes v for which $\ell(v_0, v) = r$ and $C \in n(v)$, then $\mathcal{R}(P)$ contains the set $\{P_0, P_1, \ldots, P_t\}$, where $\{u_1, \ldots, u_t\}$ consists of all sons of v_0 with $\ell(v_0, u_i) = r$ and $C \notin n(u_i)$ and*

1. *$P_0 = (V_0, E_0, n_0, \ell_0)$, where $u_0 \notin V$, $V_0 = V \cup \{u_0\}$, $E_0 = E \cup \{(v_0, u_0)\}$, $n_0 = n \cup \{u_0 \mapsto \{C\}\}$, and $\ell_0 = \ell \cup \{(v_0, u_0) \mapsto r\}$,*
2. *for $1 \leq i \leq t$, $P_i = (V, E, n_i, \ell)$, where $n_i(v) = n(v)$ for all $v \in V \setminus \{u_i\}$ and $n_i'(u_i) = n_i(u_i) \cup \{C\}$;*

R$\geqslant_2$ *if the root label $n(v_0)$ contains the concept $({\geqslant}m\, r.C)$, but not the concept $M_{r,C}$, and if there are less than m nodes v for which $\ell(v_0, v) = r$ and $C \in n(v)$, then $\mathcal{R}(P)$ contains the set $\{P_{-1}, P_0, P_1, \ldots, P_t\}$, where t and $P_0, \ldots, P_t$ are defined as in the* R$\geqslant_1$*-rule, and*

3. *$P_{-1} = (V, E, n_{-1}, \ell)$, where $n_{-1}(v) = n(v)$ for all $v \in V \setminus \{v_0\}$ and $n_{-1}(v_0) = n(v_0) \cup \{M_{r,C}\}$;*

RC$_1$ *if the root label $n(v_0)$ contains the concept $({\leqslant}m\, r.C)$ and $\ell(v_0, v_1) = r$ for some $v_1 \in V$ with $n(v_1) \cap \{C, \mathsf{nnf}(\neg C)\} = \emptyset$, then $\mathcal{R}(P)$ contains the set $\{(V, E, n', \ell), (V, E, n'', \ell)\}$, where $n'(v_1) = n(v_1) \cup \{C\}$, $n''(v_1) = n(v_1) \cup \{\mathsf{nnf}(\neg C)\}$, and $n'(v) = n''(v) = n(v)$ for all $v \in V \setminus \{v_1\}$;*

[11] For better readability, each rule will be split into two sub-rules.

RC_2 *if the root label* $n(v_0)$ *contains the concept* $(\leqslant m\, r.C)$, *but neither* $M_{r,C}$ *nor* $\neg M_{r,C}$, *then* $\mathcal{R}(P)$ *contains the set* $\{(V, E, n', \ell), (V, E, n'', \ell)\}$, *where* $n'(v_0) = n(v_0) \cup \{M_{r,C}\}$, $n''(v_0) = n(v_0) \cup \{\neg M_{r,C}\}$ *and* $n'(v) = n''(v) = n(v)$ *for all* $v \in V \setminus \{v_0\}$;

$\mathsf{R}{\uparrow}_1$ *if there is a son* v_1 *of the root* v_0 *with* $M_{r,C} \in n(v_1)$ *and* $\ell(v_0, v_1) = \overline{r}$, *but* $C \notin n(v_0)$, *then* $\mathcal{R}(P)$ *contains the singleton set* $\{P'\}$, *where* $P' = (V, E, n', \ell)$ *and* $n'(v) = n(v)$ *for all* $v \in V \setminus \{v_0\}$ *and* $n'(v_0) = n(v_0) \cup \{C\}$;

$\mathsf{R}{\uparrow}_2$ *if there is a son* v_1 *of the root* v_0 *with* $\neg M_{r,C} \in n(v_1)$ *and* $\ell(v_0, v_1) = \overline{r}$, *but* $\mathsf{nnf}(\neg C) \notin n(v_0)$, *then* $\mathcal{R}(P)$ *contains the singleton set* $\{P'\}$, *where* $P' = (V, E, n', \ell)$ *and* $n'(v) = n(v)$ *for all* $v \in V \setminus \{v_0\}$ *and* $n'(v_0) = n(v_0) \cup \{\mathsf{nnf}(\neg C)\}$.

In $S_{\mathcal{ALCQI}}$, we also need two additional clash-triggers. First, we have a clash whenever the label of the node v contains a marker concept that is in conflict with the actual label of the edge connecting the predecessor of v with v. Second, we need a clash-trigger that detects that an at-most restriction is violated.

Definition 24. *The set of clash-triggers* $\mathcal{C}$ *contains all the clash triggers of* $S_{\mathcal{ALC}}$, *and additionally*

- *all patterns* (V, E, n, ℓ) *such that there exists an edge* $(v, w) \in E$, *roles* r, s, *and a concept* C *with* $\ell(v, w) = r$, $M_{s,C} \in n(w)$, *and* $r \neq \overline{s}$;
- *all patterns* (V, E, n, ℓ) *with root* v_0 *such that* $(\leqslant m\, r.C) \in n(v_0)$ *and* v_0 *has more than* m r*-neighbors containing* C.

Admissibility, ExpTime-admissibility, and recursive admissibility of $S_{\mathcal{ALCQI}}$ can be shown as for $S_{\mathcal{ALC}}$. The proof of soundness and completeness is similar to known soundness and completeness proofs for tableau algorithms for DLs containing qualified number restrictions and inverse roles (see, e.g., [18]). In order to have p-completeness for an appropriate polynomial p, we must assume that numbers in number restrictions are given in unary coding, i.e., the number m really contributes with m to the size of the input. As an immediate consequence of Theorem 1, we obtain the following upper-bound for the satisfiability problem in $\mathcal{ALCQI}$.

Corollary 2. $\mathcal{ALCQI}$*-concept satisfiability w.r.t. TBoxes is in ExpTime.*

6 Variants and Extensions

When defining the abstract notion of a tableau system, we had several degrees of freedom. The decisions we made were motivated by our desire to stay as close as possible to the "usual" tableau-based algorithms for DLs while at the same time obtaining a notion that is as general as possible. While writing the paper, we have noticed that several decisions could have been made differently. In the following, we mention three alternative decisions, one leading to a restricted variant and two leading to extensions of the framework. Embedding the two extensions into our framework is the subject of future work.

6.1 Changing the Definition of Subpatterns

Recall that our treatment of existential restrictions in the tableau system $S_{\mathcal{ALC}}$ differs from the usual treatment in tableau-based algorithms for $\mathcal{ALC}$ in that it leads to a non-deterministic rule, which chooses between generating a new r-successor or re-using an old one. In contrast, the usual rules treating existential restrictions always generate a new successor.

Why could we not employ the usual rule for handling existential restrictions? The reason is that then the tableau system would not be admissible. In fact, the proof that Condition 2 of Definition 7 is satisfied for $S_{\mathcal{ALC}}$ (given below Definition 7) strongly depends on the fact that r-successors can be re-used. To be more precise, assume that P is a pattern whose root label consists of $\exists r.A$ for a concept name A, and whose root has exactly one successor u_1, which is an r-successor with an empty label. Let P' be the pattern that is obtained from P by adding A to the label of u_1. Obviously, $P \precsim P'$ and P' is saturated. However, if we consider the pattern P_1 that is obtained from P by adding a new r-successor with label $\{A\}$, then $P_1 \not\precsim P'$. Thus, the deterministic rule $P \to_{\mathcal{R}} \{P_1\}$ does not satisfy Condition 2 of Definition 7.

Could we change the framework such that the usual deterministic rule for handling existential restrictions becomes admissible? One way to achieve this would be to change the definition of the subpattern relation $\precsim$ (see Definition 6) by removing the requirement that π be injective. In fact, with this new definition, we would have $P_1 \precsim P'$ in the example above. By consistently replacing the old version of $\precsim$ with this new version, we would obtain a framework where all the results of Sections 3 and 4 still hold, and where the usual deterministic rule for handling existential restrictions in $\mathcal{ALC}$ is admissible.

Why did we not use this modified framework? Intuitively, if we use a non-injective mapping π in the definition of $\precsim$, then the actual number of r-successors of a given node is irrelevant as long as we have one successor of each "type." Thus, a clash-trigger that fires if a certain number of successors is exceeded (like the one used in Section 5) does not make sense. In fact, with the modified definition of $\precsim$, a pattern $P \in \mathcal{C}$ having at least m successors of the root node could be a subpattern of a pattern T, x where x has only one successor. Thus, the modified framework could not treat a DL like $\mathcal{ALCQI}$, where the number of successors (and not just their type) counts. For DLs like $\mathcal{ALC}$, where the number of successors of a given type is irrelevant,[12] the modified framework could be used, and would probably lead to simpler rules. However, we think that number restrictions are important enough in DLs to justify the use of a framework that can handle them, even if this leads to a somewhat more complex treatment of other constructors.

[12] This follows from the bisimulation invariance of $\mathcal{ALC}$, which is an immediate consequence of bisimulation invariance of its syntactic variant, multi-modal K_m [5].

6.2 Using Larger Patterns

In our current framework, patterns (the clash-triggers and left-hand sides of rules) are trees of depth at most one, i.e., we consider one node and its direct successors when defining rules and clash-triggers. In some cases, it would be more convenient to have larger patterns available. A case in point are DLs with inverse roles (like $\mathcal{ALCQI}$), where it would be more convenient to have not only the direct successors of a node available, but also its direct predecessor. In our definition of the tableau system for $\mathcal{ALCQI}$, we had to employ special markers to memorize whether the predecessor belongs to a certain concept. Though this works, it is not very natural, and it leads to rather complicated rules. Thus, a natural extension motivated by $\mathcal{ALCQI}$ and similar DLs is to consider patterns consisting of a node together with its direct predecessor and its direct successors. This would yield a new framework that is close to two-way automata [28].

Why have we not made this extension? Including the predecessor of a node in the definition of patterns is an extension that appears to be tailored to the treatment of DLs with inverse roles. Thus, it has the flavor of an ad-hoc extension, with the clear danger that adding another constructor may motivate yet another extension of the framework.

Is there a more general extension? Instead of restricting patterns to being certain trees of depth 2, a more general extension would be to use as patterns trees of some fixed depth k or patterns whose depth is bounded by some function of the input size. We conjecture that it is possible to extend our framework in this direction while retaining the results shown in this paper. In contrast to the extension of patterns by predecessor nodes, this appears to require some more work, though.

6.3 Allowing for Global Information

In the present framework, rules are local in that they consider only one node and its direct successors. The extension mentioned in the previous subsection extends the scope of rules but leaves it still local (bounded by the depth of patterns). In some cases, it would be convenient to be able to access global information that can influence the behavior of rules and can also be changed by rules.

Is such global information useful? A typical example where it would be convenient to allow for global information are DLs with so-called nominals, i.e., concept names that must be interpreted as singletons, and thus stand for a single element of the interpretation domain. Assume that N is such a nominal. If N occurs in the label of two different nodes of a completion tree, then this means that these nodes represent the same individual in the corresponding model, and thus the whole label sets of these nodes must coincide in a saturated and clash-free completion tree. Thus, rules and clash-triggers that are designed to realizing this are concerned with information about nodes that may be quite far apart from each other in the tree. One way of ensuring this could be to have, in addition to the completion tree with its local node labels, a global book-keeping component that contains information about the labels of all nominals. A rule that encounters the nominal N in the label of node v may then use the information in the book-keeping component for nominal N to extend the label of v, but it may also extend the book-keeping component based on what is found in the label of v. Thus, through this book-keeping component, information can be passed between nodes that are far apart from each other in the tree.

Is this extension too general? We believe that this extension is harmless as long as the number of possible "states" of the book-keeping component is appropriately bounded by the size of the input. Of course, this depends on the exact definition of the book-keeping component and its interaction with rules and clash-triggers. The integration of such a book-keeping component into our framework and the proof that the results shown in the present paper still hold in this extended framework is a subject of future research.

References

1. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003.
2. F. Baader, E. Franconi, B. Hollunder, B. Nebel, and H.-J. Profitlich. An empirical analysis of optimization techniques for terminological representation systems or: Making KRIS get a move on. *Applied Artificial Intelligence. Special Issue on Knowledge Base Management*, 4:109–132, 1994.
3. F. Baader and W. Nutt. Basic description logics. In *[1]*, pages 43–95. 2003.
4. F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001.
5. P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 2001.
6. R. J. Brachman and H. J. Levesque, editors. *Readings in Knowledge Representation.* Morgan Kaufmann, Los Altos, 1985.
7. D. Calvanese and G. DeGiacomo. Expressive description logics. In *[1]*, pages 178–218. 2003.
8. G. De Giacomo and M. Lenzerini. TBox and ABox reasoning in expressive description logics. In L. C. Aiello, J. Doyle, and S. C. Shapiro, editors, *Proc. of the 5th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'96)*, pages 316–327. Morgan Kaufmann, Los Altos, 1996.
9. F. Donini. Complexity of reasoning. In *[1]*, pages 96–136. 2003.
10. F. Donini and F. Massacci. EXPTIME tableaux for $\mathcal{ALC}$. *Acta Informatica*, 124(1):87–138, 2000.
11. F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proc. of the 2nd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'91)*, pages 151–162. Morgan Kaufmann, Los Altos, 1991.
12. F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. Tractable concept languages. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, pages 458–463, Sydney (Australia), 1991.
13. V. Haarslev and R. Möller. RACER system description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, 2001.

14. B. Hollunder and F. Baader. Qualifying number restrictions in concept languages. In *Proc. of the 2nd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'91)*, pages 335–346, 1991.
15. I. Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 636–647, 1998.
16. I. Horrocks. Implementation and optimization techniques. In *[1]*, pages 306–346. 2003.
17. I. Horrocks and U. Sattler. A description logic with transitive and inverse roles and role hierarchies. *J. of Logic and Computation*, 9(3):385–410, 1999.
18. I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for very expressive description logics. *J. of the Interest Group in Pure and Applied Logic*, 8(3):239–264, 2000.
19. R. MacGregor. The evolving technology of classification-based knowledge representation systems. In J. F. Sowa, editor, *Principles of Semantic Networks*, pages 385–400. Morgan Kaufmann, Los Altos, 1991.
20. E. Mays, R. Dionne, and R. Weida. K-REP system overview. *SIGART Bull.*, 2(3), 1991.
21. M. Minsky. A framework for representing knowledge. In J. Haugeland, editor, *Mind Design*. The MIT Press, 1981. A longer version appeared in *The Psychology of Computer Vision* (1975). Republished in [6].
22. R. Möller and V. Haarslev. Description logic systems. In *[1]*, pages 282–305. 2003.
23. P. F. Patel-Schneider, D. L. McGuiness, R. J. Brachman, L. Alperin Resnick, and A. Borgida. The CLASSIC knowledge representation system: Guiding principles and implementation rational. *SIGART Bull.*, 2(3):108–113, 1991.
24. C. Peltason. The BACK system — an overview. *SIGART Bull.*, 2(3):114–119, 1991.
25. M. R. Quillian. Word concepts: A theory and simulation of some basic capabilities. *Behavioral Science*, 12:410–430, 1967. Republished in [6].
26. K. Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, pages 466–471, 1991.
27. M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.
28. M. Y. Vardi. Reasoning about the past with two-way automata. In *Proc. of the 25th Int. Coll. on Automata, Languages and Programming (ICALP'98)*, volume 1443 of *Lecture Notes in Computer Science*, pages 628–641. Springer-Verlag, 1998.
29. M. Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *J. of Computer and System Sciences*, 32:183–221, 1986. A preliminary version appeared in *Proc. of the 16th ACM SIGACT Symp. on Theory of Computing (STOC'84)*.
30. W. A. Woods and J. G. Schmolze. The KL-ONE family. In F. W. Lehmann, editor, *Semantic Networks in Artificial Intelligence*, pages 133–178. Pergamon Press, 1992. Published as a special issue of *Computers & Mathematics with Applications*, Volume 23, Number 2–9.