Global Caching for Coalgebraic Description Logics [*]

Rajeev Goré[1] and Clemens Kupke[2] and Dirk Pattinson[2] and Lutz Schröder[3]

[1] Computer Science Laboratory, The Australian National University
[2] Department of Computing, Imperial College London
[3] DFKI Bremen and Department of Computer Science, Universität Bremen

Abstract. Coalgebraic description logics offer a common semantic umbrella for extensions of description logics with reasoning principles outside relational semantics, e.g. quantitative uncertainty, non-monotonic conditionals, or coalitional power. Specifically, we work in coalgebraic logic with global assumptions (i.e. a general TBox), nominals, and satisfaction operators, and prove soundness and completeness of an associated tableau algorithm of optimal complexity EXPTIME. The algorithm uses the (known) tableau rules for the underlying modal logics, and is based on on global caching, which raises hopes of practically feasible implementation. Instantiation of this result to concrete logics yields new algorithms in all cases including standard relational hybrid logic.

Introduction

Description Logics (DLs) [2], which may be regarded as notational variants of various extensions of modal logic, constitute one of the most important formalisms in the area of logic-based knowledge representation. Two key features of DLs are support for *nominals*, i.e. the ability to reason about particular individual states of the model rather than about subsets only, and support for reasoning using global assumptions, a so-called (general) TBox. The combination of these two features corresponds roughly to (i.e. is slightly less expressive than) reasoning with global assumptions in hybrid logic [1].

On the other hand, reasoning about real-life problems typically calls for modal principles beyond the standard existential and universal constraints that correspond in DL notation to the boxes and diamonds of modal logic. Some of these principles, such as qualified number constraints, are already incorporated in many DLs, while support for others such as quantitative uncertainty ('with likelihood at least p'), non-monotonic conditionals ('if a then normally b'), and strategic aspects ('coalition C of agents can force a') is less well-developed. In particular, the complexity of hybrid or description logics including these features was largely unknown until it was shown recently that TBox reasoning in such logics is, under weak assumptions, in EXPTIME and, hence, typically EXPTIME complete [20] The generic framework that makes results at this level of generality possible is *coalgebraic logic*, which relies on the principle of encapsulating the type of systems underlying the semantics of a given modal or hybrid logics (neighbourhood frames, Kripke frames, Markov chains, game frames etc.) as coalgebras for a set functor.

[*] The second and third author supported by EPSRC grant EP/F031173/1. The last author supported by BMBF grant FormalSafe (FKZ 01IW07002).

From a practical point of view, the generic algorithm presented in [20] has two drawbacks: it relies on decision procedures for infinite games, which guarantee EXPTIME complexity but also induce average-case exponential run time; and moreover it starts by guessing the theories of named states, which is practically infeasible. In the present work, we drastically improve on this by presenting an EXPTIME tableau algorithm that uses *global caching*. Tableau-based methods have been successful in providing efficient decision procedures for modal and description logics [12], and in particular are employed in the leading current DL reasoners. They are relatively easy to implement but often do not meet known upper complexity bounds for a given logic. For example, the traditional tableau algorithm for the $\mathcal{ALC}$ requires double exponential time in the worst case, and known alternative EXPTIME tableaux are highly complex [8]. Global caching has recently been proposed as a way of establishing tableau-based decision procedures that do meet known optimality bounds, while at the same time offering good practical efficiency and room for heuristic optimization. It has been successfully applied to the description logics $\mathcal{ALC}$ and $\mathcal{ALCI}$ [10,11] and, more recently, to coalgebraic modal logic with global assumptions [9].

The fundamental idea of global caching is that tableau sequents should be arranged in a directed graph rather than in a tree. In this way one is able to avoid unnecessary repetitions of calculations that would normally happen on various branches of a tree-shaped tableau. The challenge in extending global caching to hybrid logics is to manage the theories of named nodes in a consistent way across the tableau without resorting to backtracking, which is what global caching tries to avoid. Our algorithm achieves precisely this. It instantiates to new tableau-based decision procedures for a wide range of hybrid logics such as probabilistic and graded hybrid logics and hybrid coalition logic. Even its incarnation in the traditional relational realm seems to be a new decision procedure for global reasoning in hybrid K (or $\mathcal{ALCO}$ extended with satisfaction operators; see [4,3] for an overview of existing tableau systems for hybrid K), to our knowledge the first backtracking-free tableau-based procedure for hybrid K that matches the known EXPTIME bound [1]. Compared to other approaches in the literature [4] we note that our tableaux are *unlabelled* which allows us to avoid backtracking, as we do not need to pay special attention to labels, and in turn entails the optimal complexity bound.

1 Global Caching for Hybrid Logics, Informally

We proceed to give a brief motivation of the main ideas of our algorithm, using two very simple examples; the full description of the algorithm is given in Section 4. We feel entitled to work in the standard example of hybrid logic over Kripke frames, as our algorithm appears to be new even in this basic case, but we emphasize that our method applies in the much wider setting of coalgebraic hybrid logic. Recall that the main features that distinguish hybrid logic from plain modal logic (which comes with an operator $\Diamond$ ‘there exists a successor state such that’, corresponding to existential restrictions in description logic) are *nominals* which designate individual states, and *satisfaction operators* $@_i$ ‘state i satisfies …’. The main problem in designing tableaux calculi for hybrid logics that are amenable to global caching is that the satisfaction operators are global in nature, i.e. all states in the tableau have to agree on the truth values of formulas of the form $@_i A$. In complexity proofs, it is unproblematic to just guess these truth values before building the tableau [14,20], but of course this is not a feasible approach in the design of a reasoning algorithm that aims for efficient average-case behaviour. Standard tableau algorithms for hybrid logic (e.g. [3]) explore possible truth values for @-formulas via backtracking; however, the driving idea behind the global caching approach is precisely to minimize backtracking.

Our solution to this problem is roughly as follows. We view the construction of a tableau in the standard way as a game between two players, Eloise ($\exists$) and Abelard ($\forall$), where $\exists$ tries to prove satisfiability of the target formulas, and $\forall$ unsatisfiability. A typical move by $\exists$ in a standard purely modal tableau would be, e.g., to select a disjunct from a disjunction, while a typical move by $\forall$ would be, e.g., to pick a diamond formula $\Diamond A$ for which $\exists$ then has to establish satisfiability of a new state labelled A. In the hybrid setting, $\exists$ may stumble upon formulas $@_i\psi$ in this process; she may choose to just ignore these for the moment, but will later be forced to prove satisfiability of all such formulas that she runs across. Our main idea is now to collect such @-formulas along a potential winning strategy of $\exists$ and propagate this collection back through the tableau. Collections of @-formulas, which we call @-constraints, are attached to standard nodes via special links, along (one of) which $\exists$ may be forced to move to prove satisfiability.

As our first simple example, we have the following tableau for the sequent $\Diamond @_i p, \Diamond(\Diamond @_i q \vee \Diamond \neg p)$ (with the comma read conjunctively).

$$\begin{tikzcd}
\boxed{@_i p, @_i \neg p} & & \Diamond @_i p, \Diamond(\Diamond @_i q \vee \Diamond @_i \neg p) \arrow[ll, dotted, "\exists"'] \arrow[rrr, dotted, "\exists"] \arrow[dl, "\forall"'] \arrow[d, "\forall"] & & & \boxed{@_i p, @_i q} \arrow[d, "{\forall,\exists}"] \\
\boxed{@_i p} & \Diamond @_i p \arrow[l, dotted, "\exists"'] \arrow[d, "\forall"'] & \Diamond @_i q \vee \Diamond @_i \neg p \arrow[r, dotted, "\exists"] \arrow[drr, dotted, "\exists"] \arrow[ddl, "\exists"] \arrow[ddr, "\exists"'] & \boxed{@_i q} & & i, p, q, @_i p, @_i q \arrow[u, dotted] \\
\boxed{@_i p} & @_i p \arrow[l, dotted, "\exists"'] & & & \boxed{@_i \neg p} & \\
\boxed{@_i q} & \Diamond @_i q \arrow[l, dotted, "\exists"'] \arrow[d, "\forall"] & & \Diamond @_i \neg p \arrow[r, dotted, "\exists"] \arrow[d, "\forall"'] & \boxed{@_i \neg p} & \\
\boxed{@_i q} & @_i q \arrow[l, dotted, "\exists"'] & & @_i \neg p \arrow[r, dotted, "\exists"] & \boxed{@_i \neg p} &
\end{tikzcd}$$

The solid arrows above indicate unfolding of tableau rules. At the root node, $\forall$ has two challenges, corresponding to the two $\Diamond$-formulas, that $\exists$ can answer in each case, and she can choose between two disjuncts at the node below the root. Ignoring @-constraints, both choices would demonstrate satisfiability of the root. The @-constraints are represented by dotted lines above, and are propagated from the bottom nodes that contain @-formulas. At the node below the root, $\exists$ can demonstrate satisfiability if she can satisfy *either* of the two constraints, which are propagated to the root node, and induce (stronger) constraints that also account for the the second (left) $\Diamond$-formula. At the root, each constraint represents a set of assumptions that need to be met in order for $\exists$ to demonstrate satisfiability, and the ability to establish satisfiability of the constraint on the right finally gives satisfiability of the root node. Note that we could have declared satisfiability even without unfolding the left disjunct at the node below the root, as the satisfiability of a node hinges on the satisfiability of only *one* constraint. The situation is different in the next example, a tableau for the sequent $\Diamond @_i \bot, \Diamond A$, where A is some complex formula:

$$\begin{tikzcd}
 & \Diamond @_i \bot, \Diamond A \arrow[r, dotted, "\exists"] \arrow[d, "\forall"] & \boxed{@_i \bot, \bullet} \arrow[d, "\forall"] \\
\boxed{@_i \bot} & @_i \bot \arrow[l, dotted, "\exists"'] & i, \bot, @_i \bot
\end{tikzcd}$$

Here, the root formula is clearly unsatisfiable, and $\forall$ can challenge the satisfiability of the root by requiring successors for each of the $\Diamond$-formulas. Unfolding the left $\Diamond$, this induces an unsatisfiable @-constraint that is propagated to the root, but as the right $\Diamond$ is not yet unfolded, this constraint is incomplete, indicated by $\bullet$ – further unfolding and propagation would still require satisfiability of $@_i \bot$. Challenging the satisfiability of this (incomplete) constraint, $\forall$ demonstrates unsatisfiability of the root node, even without fully unfolding the tableau.

2 Syntax and Semantics of Coalgebraic Hybrid Logic

To make our treatment parametric in the concrete syntax of any given modal logic, we fix a modal similarity type Λ consisting of modal operators with associated arities throughout. Throughout the paper, Prop and N denote two denumerable disjoint sets of *propositional variables* and *nominals*, respectively. We will only be considering formulas in negation normal form, and abbreviate $\overline{\mathsf{P}} = \{\overline{p} \mid p \in \mathsf{P}\}$ and $\overline{\Lambda} = \{\overline{\heartsuit} \mid \heartsuit \in \Lambda\}$ where the derived dual modal operator $\overline{\heartsuit}$ has the same arity as $\heartsuit$. The set $\mathcal{H}(\Lambda)$ of *hybrid Λ-formulas* is given by the grammar

$$\mathcal{H}(\Lambda) \ni A, B ::= x \mid A \wedge B \mid A \vee B \mid \heartsuit(A_1, \dots, A_n) \mid \overline{\heartsuit}(A_1, \dots, A_n) \mid @_i A$$

where $x \in \mathsf{P} \cup \overline{\mathsf{P}} \cup \mathsf{N} \cup \overline{\mathsf{N}}$ is a possibly negated propositional variable or nominal, $\heartsuit \in \Lambda$ is n-ary and $i \in \mathsf{N}$ is a nominal. We write

$$(\Lambda \cup \overline{\Lambda})(F) = \{\heartsuit(A_1, \dots, A_n) \mid \heartsuit \in \Lambda \cup \overline{\Lambda}\ n\text{-ary}, A_1, \dots, A_n \in F\}$$

for the set of all formulas that can be constructed by applying a (possibly dualised) modal operator to elements of a set F of formulas. A *Λ-tableau-sequent*, short *Λ-sequent* or just *sequent*, is a finite set of Λ-formulas that we read conjunctively, and we write $\mathcal{S}(\Lambda)$ for the set of Λ-sequents. A *Λ-state* is a Λ-sequent that neither contain a top-level propositional connective nor a pair $x, \overline{x}$ of complementary propositional variables or nominals, and we write $\mathsf{State}(\Lambda)$ for the set of Λ-states. We define the negation $\overline{A}$ of a formula $A \in \mathcal{H}(\Lambda)$ by $\overline{\overline{p}} = p$, $\overline{(A \wedge B)} = \overline{A} \vee \overline{B}$, $\overline{A \vee B} = \overline{A} \wedge \overline{B}$, $\overline{\heartsuit(A_1, \dots A_n)} = \overline{\heartsuit}(\overline{A}_1, \dots, \overline{A}_n)$, $\overline{\overline{\heartsuit}(A_1, \dots, A_n)} = \heartsuit(\overline{A}_1, \dots, \overline{A}_n)$ and $\overline{@_i A} = @_i \overline{A}$. We use the standard definitions for the other propositional connectives $\to, \leftrightarrow, \vee$. The set of nominals occurring in a formula A is denoted by $\mathsf{N}(A)$. This extends to sets of formulas, and $\mathsf{N}(\Gamma) = \bigcup\{\mathsf{N}(A) \mid A \in \Gamma\}$. A formula of the form $@_i A$ is called an *@-formula*, and a formula that does not begin with @ is called *plain*.

Semantically, nominals i denote individual states in a model, and an @-formula $@_i A$ stipulates that A holds at i.

To reflect parametricity in the particular underlying logic also semantically, we equip hybrid logics with a *coalgebraic semantics* extending the standard coalgebraic semantics of modal logics [15]: we fix an endofunctor $T : \mathsf{Set} \to \mathsf{Set}$ throughout that is equipped with an assigment of an *n-ary predicate lifting* $[\![\heartsuit]\!]$ for every n-ary modal operator $\heartsuit \in \Lambda$, i.e. a set-indexed family of mappings $([\![\heartsuit]\!]_X : \mathcal{P}(X)^n \to \mathcal{P}(TX))_{X \in \mathsf{Set}}$ that satisfies

$$[\![\heartsuit]\!]_X \circ (f^{-1})^n = (Tf)^{-1} \circ [\![\heartsuit]\!]_Y$$

for all $f : X \to Y$. In categorical terms, $[\![\heartsuit]\!]$ is a natural transformation $\mathcal{Q}^n \to \mathcal{Q} \circ T^{op}$ where $\mathcal{Q} : \mathsf{Set}^{op} \to \mathsf{Set}$ is the contravariant powerset functor.

In this setting, T-coalgebras play the roles of *frames*. A *T-coalgebra* is a pair (C, γ) where C is a set of *states* and $\gamma : C \to TC$ is the *transition function*. If clear from the context, we identify a T-coalgebra (C, γ) with its state space C. A *(hybrid) T-model* (C, γ, π) consists of a T-coalgebra (C, γ) together with a *hybrid valuation* π, i.e. a map $\mathsf{P} \cup \mathsf{N} \to \mathcal{P}(C)$ that assigns singleton sets to all nominals $i \in \mathsf{N}$. We often identify the singleton set $\pi(i)$ with its unique element.

The semantics of $\mathcal{H}(\Lambda)$ is a satisfaction relation $\models$ between states $c \in C$ in hybrid T-models $M = (C, \gamma, \pi)$ and formulas $A \in \mathcal{H}(\Lambda)$, inductively defined as follows. For $x \in \mathsf{N} \cup \mathsf{P}$ and $i \in \mathsf{N}$, put

$$c, M \models x \text{ iff } c \in \pi(x) \qquad c, M \models @_i A \text{ iff } \pi(i), M \models A.$$

Modal operators are interpreted using their associated predicate liftings, that is,

$$c, M \models \heartsuit(A_1, \dots, A_n) \iff \gamma(c) \in [\![\heartsuit]\!]_C([\![A_1]\!]_M, \dots, [\![A_n]\!]_M)$$

where $\heartsuit \in \Lambda$ n-ary and $[\![A]\!]_M = \{c \in C \mid M, c \models A\}$ denotes the truth-set of A relative to M and $[\![\overline{\heartsuit}]\!]_C(A_1, \dots, A_n) = C \setminus [\![\heartsuit]\!](C \setminus A_1, \dots, C \setminus A_n)$ for the case of dual operators. If Ξ is a set of formulas (the global assumptions, or TBox), we write $\mathsf{Mod}(\Xi)$ for the class of models $M = (C, \gamma, \pi)$ such that $M, c \models A$ for all $A \in \Xi$ and all $c \in C$. A formula A is *satisfiable in* $\mathsf{Mod}(\Xi)$ if it is satisfied in some state in some model of $\mathsf{Mod}(\Xi)$.

The distinguishing feature of the coalgebraic approach to hybrid and modal logics is the parametricity in both the logical language and the notion of frame: concrete instantiations of the general framework, in other words a choice of modal operators Λ, an endofunctor T and an assigment of predicate liftings, capture the semantics of a wide range of modal logics, as witnessed by the following examples.

Example 1. 1. The hybrid version of the modal logic K, *hybrid K* for short, has a single unary modal operator $\Box$, and we write $\overline{\Box} = \Diamond$. Hybrid K is interpreted over coalgebras for the powerset functor $\mathcal{P}$ that takes a set X to its powerset $\mathcal{P}(X)$ and $[\![\Box]\!]_X(A) = \{B \in \mathcal{P}(X) \mid B \subseteq A\}$. It is clear that $\mathcal{P}$-coalgebras $(C, \gamma : C \to \mathcal{P}(C))$ are in 1-1 correspondence with Kripke frames, and that the coalgebraic definition of satisfaction specialises to the usual semantics of the box operator.

2. *Graded hybrid logic* has modal operators $\Diamond_k$ 'in more than k successors, it holds that', where we write $\overline{\Diamond}_k = \Box_k$. It is interpreted over the functor $\mathcal{B}$ that takes a set X to the set $\mathcal{B}(X)$ of multisets over X, i.e. maps $B : X \to \mathbb{N} \cup \{\infty\}$, by $[\![\Diamond_k]\!]_X(A) = \{B \in \mathcal{B}(X) \mid \sum_{x \in A} B(x) > k\}$. This captures the semantics of graded modalities over *multigraphs* [7], which are precisely the $\mathcal{B}$-coalgebras. One can encode the description logic $\mathcal{ALCOQ}$ (which features *qualified number restrictions* $\geq nR$ and has a relational semantics) into multi-agent graded hybrid logic with multigraph semantics by adding formulas $\neg\Diamond_1 i$ for all occurring nominals i to the TBox.

3. *Probabilistic hybrid logic*, the hybrid extension of probabilistic modal logic [13], has modal operators L_p 'in the next step, it holds with probability at least p that', for $p \in [0,1] \cap \mathbb{Q}$. It is interpreted over the functor D_ω that maps a set X to the set of finitely-supported probability distributions on X by putting $[\![L_p]\!]_X(A) = \{P \in D_\omega(X) \mid PA \geq p\}$. Coalgebras for D_ω are just Markov chains.

3 Tableau Rules for Coalgebraic Logics

We now introduce the (type of) tableau rules we will be working with. Clearly, these rules have to relate syntax and semantics in an appropriate way, and we cannot expect to prove as much as soundness, let alone completeness, without the rules satisfying appropriate coherence conditions, which we introduce later. We begin with the propositional part of the calculus, for which it is convenient to unfold propositional connectives in a single step. This process is called *saturation* and is given by a map sat that is defined inductively by the clauses

$$\begin{aligned} \mathsf{sat}(\Delta') &= \{\Delta'\} & \mathsf{sat}(A \vee B, \Gamma) &= \mathsf{sat}(A, \Gamma) \cup \mathsf{sat}(B, \Gamma) \\ \mathsf{sat}(x, \overline{x}, \Gamma) &= \emptyset & \mathsf{sat}(A \wedge B, \Gamma) &= \mathsf{sat}(A, B, \Gamma) \end{aligned}$$

where $A, B \in \mathcal{H}(\Lambda)$ are formulas, Γ is a sequent, $x \in \mathsf{P} \cup \mathsf{N}$ is a propositional variable or a nominal and $\Delta' \in \mathsf{State}(\Lambda)$ is a state, i.e. contains neither complementary propositional variables nor top-level propositional connectives. As we interpret hybrid formulas over the class of *all* (coalgebraic) hybrid models, it suffices to use modal rules of a rather specific form where the premise contains only modalised formulas and the conclusion is purely propositional in terms of the arguments of the modalities. Rules of this type are called *one-step rules* and have been used in the context of tableau calculi in [9,6] and originate from the (dual) sequent rules of [19].

Definition 2. A *one-step tableau rule* over Λ is a tuple $(\Gamma_0, \Gamma_1, \ldots, \Gamma_n)$, written as $\Gamma_0/\Gamma_1 \ldots \Gamma_n$, where $\Gamma_0 \subseteq (\Lambda \cup \overline{\Lambda})(\mathsf{P} \cup \overline{\mathsf{P}})$ and $\Gamma_i \subseteq \mathsf{P} \cup \overline{\mathsf{P}}$ so that every variable that occurs in the conclusion $\Gamma_1 \ldots \Gamma_n$ also occurs in the premise Γ_0, and every propositional variable occurs at most once in the premise Γ_0.

We can think of one-step rules as a syntactic representation of the inverse image $\gamma^{-1} : \mathcal{P}(TC) \to \mathcal{P}(C)$ of a generic coalgebra map $\gamma : C \to TC$ in that the premise describes a property of successors, whereas the conclusion describes states. The requirement that propositional variables do not occur twice in the premise is a mere technical convenience, and can be met by introducing premises that state the equivalence of variables. While this rigid format of one-step rules suffices to completely axiomatise all coalgebraic logics [17], it does not accommodate frame conditions like transitivity ($\Box p \to \Box\Box p$), which require separate consideration.

Example 3. One-step rules that axiomatise the logics in Example 1 can be found (in the form of proof rules) in [16,19]. Continuing Example 1, we single out hybrid K and (hybrid) graded modal logic.

1. Hybrid K is axiomatised by the set $\mathcal{R}_K$ of one-step rules that contain
$$(K)\,\frac{\Diamond p_0, \Box p_1, \ldots, \Box p_n}{p_0, \ldots, p_n} \qquad \text{for all } n \geq 0.$$
2. The rules $\mathcal{R}_{\mathcal{B}}$ for graded modal logic contain
$$(G)\,\frac{\Diamond_{k_1} p_1, \ldots, \Diamond_{k_n} p_n, \Box_{l_1} q_1, \ldots, \Box_{l_m} q_m}{\sum_{j=1}^{m} s_j \overline{q}_j - \sum_{i=1}^{n} r_i p_i < 0}$$

where $n, m \in \mathbb{N}$ and $r_i, s_j \in \mathbb{N} \setminus \{0\}$ satisfy the side condition $\sum_{i=1}^{n} r_i(k_i + 1) \geq 1 + \sum_{j=1}^{m} s_j l_j$. The conclusion of (G) is to be read as arithmetic of characteristic functions, and expands into a disjunctive normal form with only positive literals [19].

While the above are examples of one-step rules, the generic treatment of a larger class of modal logic requires that we abstract away from concretely given rule sets. This is achieved by formalising *coherence conditions* that link the rules with the coalgebraic semantics. The following terminology is handy to formalise these conditions:

Definition 4. Suppose that X is a set, $\mathsf{P} \subseteq \mathsf{Prop}$ is a set of variables and $\tau : \mathsf{P} \to \mathcal{P}(X)$ is a valuation. The interpretation of a propositional sequent $\Gamma \subseteq \mathsf{P} \cup \overline{\mathsf{P}}$ relative to (X, τ) is given by $[\![\Gamma]\!]_{X,\tau} = \bigcap\{\tau(p) \mid p \in \Gamma\} \cap \bigcap\{X \setminus \tau(p) \mid \overline{p} \in \Gamma\} \subseteq X$. Modalised sequents $\Gamma \subseteq (\Lambda \cup \overline{\Lambda})(\mathsf{P} \cup \overline{\mathsf{P}})$ are interpreted, again relative to (X, τ), as subsets of TX by
$$[\![\Gamma]\!]_{TX,\tau} = \bigcap\{[\![\heartsuit]\!]_X([\![p_1]\!]_{X,\tau}, \ldots, [\![p_n]\!]_{X,\tau}) \mid \heartsuit(p_1, \ldots, p_n) \in \Gamma\}$$
where $p_1, \ldots, p_n \in \mathsf{P} \cup \overline{\mathsf{P}}$ and $\heartsuit \in \Lambda \cup \overline{\Lambda}$.

The coherence conditions can be formulated solely in terms of (the interpretation of) propositional and modal sequents. In particular, we do not consider models for the logic under scrutiny.

Definition 5. Suppose that $\mathcal{R}$ is a set of one-step tableau rules. We say that $\mathcal{R}$ is *one-step tableau sound* (resp. *one-step tableau complete*) with respect to T if, for all $\mathsf{P} \subseteq \mathsf{Prop}$, all finite $\Gamma \subseteq (\Lambda \cup \overline{\Lambda})(\mathsf{P} \cup \overline{\mathsf{P}})$, all sets X and valuations $\tau : \mathsf{P} \to \mathcal{P}(X)$: $[\![\Gamma]\!]_{TX,\tau} \neq \emptyset$ only if (if) for all rules $\Gamma_0/\Gamma_1 \ldots \Gamma_n \in \mathcal{R}$ and all renamings $\sigma : \mathsf{P} \to \mathsf{P}$ (such that $A \neq B$ implies $A\sigma \neq B\sigma$ for all $A, B \in \Gamma_0$) with $\Gamma_0\sigma \subseteq \Gamma$, we have that $[\![\Gamma_i\sigma]\!]_{X,\tau} \neq \emptyset$ for some $1 \leq i \leq n$.

This means that a rule set is sound and complete if a modalised sequent is satisfiable iff every one-step rule applicable to it has at least one satisfiable conclusion. We note that the rule sets given in Example 3 are both one-step sound and one-step complete for their respective interpretations. This is argued, in the dual case of sequent rules, in [19].

4 Caching Graphs for Coalgebraic Hybrid Logic

Caching graphs address the problem of deciding the validity of a sequent $\varGamma_0$ under a finite set of global assumptions (TBox) $\varXi$ both of which we fix throughout. We write $\mathcal{C}$ for the *closure* of $\varGamma_0, \varXi$, i.e. the smallest set of formulas that contains $\varGamma, \varXi$ and is closed under taking subformulas, their (involutive) negations and prefixing of plain formulas (that do not begin with @) with $@_i$ where $i \in \mathsf{N}(\mathcal{C})$; we then work with *sequents over* $\mathcal{C}$, i.e. subsets of $\mathcal{C}$, but mostly omit explicit mention of $\mathcal{C}$.

For a given one-step sound and complete set $\mathcal{R}$ of one-step rules, this allows us to consider the set $T(\mathcal{R})$ that consists of the rule instances that are needed to expand sequents over $\mathcal{C}$.

Definition 6. The set $T(\mathcal{R})$ of *tableau rules relative to* $\mathcal{C}$ consists of the rules $\varGamma/\mathsf{sat}(\varGamma)$ for $\varGamma \in \mathcal{C}$ and the rules

$$\frac{\varGamma\sigma, \varGamma'}{\varDelta_1\sigma, \varXi \quad \dots \quad \varDelta_n\sigma, \varXi}$$

where $\varGamma/\varDelta_1, \dots, \varDelta_n \in \mathcal{R}$, $\sigma : \mathsf{P} \to \mathcal{C}$ is a substitution such that $\varGamma\sigma \subseteq \mathcal{C}$ and σ does not identify elements of $\varGamma$, and $\varGamma' \subseteq \mathcal{C}$.

In other words, the set $T(\mathcal{R})$ of tableau rules relative to $\mathcal{C}$ consists of all substitution instances of one-step rules where we allow an additional sequent $\varGamma'$ in the premise to absorb weakening, and the set $\varXi$ of global assumptions is added to every conclusion. Informally, the conclusions of a modal rule specify properties of successor states, and adding $\varXi$ to each conclusion ensures that successor states also validate $\varXi$, leading to a model that globally validates the TBox.

While the tableau rules are used to expand sequents, a second type of sequent is needed to deal with the @-formulas: since @-formulas are either globally true or globally false, they need to be propagated across the tableau, and their validity needs to be ascertained. This is the role of @-constraints that we now introduce, together with rules that govern their expansion.

Definition 7 (@-Constraints). An *@-constraint* over $\mathcal{C}$ is a finite set of @-formulas in $\mathcal{C}$ that may additionally include the symbol $\bullet$. The expansion of @-constraints is governed by the rules $T(@)$ that contain, for each @-constraint $\varUpsilon$ over $\mathcal{C}$, the following *@-expansion rules*

$$\frac{\varUpsilon}{i, \varUpsilon/@_i, \varUpsilon \setminus \{\bullet\}, \varXi}$$

where $i \in \mathsf{N}(\varUpsilon)$ is a nominal occurring in $\varUpsilon$ and $\varUpsilon/@_i = \{A \mid @_i A \in \varUpsilon\}$.

The role of @-constraints is to record those formulas that are required to be *globally* valid (and therefore need to satisfy the global assumptions $\varXi$)to guarantee the satisfiability of a particular sequent. To check whether a particular @-constraint is satisfiable, we therefore need to check, for each applicable @-expansion rule, the consistency of the conclusion. The role of $\bullet$ as an element of an @-constraint is to denote an (as yet) unknown constraint to be induced by a sequent that is yet to be expanded. We next introduce caching graphs, which provide the fundamental data structure that allows for the propagation of these constraints.

Definition 8. A *caching graph* over $\mathcal{C}$ is a tuple $G = (S, C, L_M, L_@, \lambda_S, \lambda_C)$ where

- S and C are sets of sequents and @-constraints respectively
- $L_M = (L_M^\forall, L_M^\exists)$ is a pair of relations with $L_M^\forall \subseteq S \times \mathcal{P}(S)$ and $L_M^\exists \subseteq \mathcal{P}(S) \times S$
- $L_@ = (L_@^\forall, L_@^\exists)$ is a pair of relations with $L_@^\forall \subseteq C \times S$ and $L_@^\exists \subseteq S \times C$; we require that $L_@^\exists$ is *upclosed*, i.e. $(\Gamma, \Upsilon) \in L_@^\exists$ and $\Upsilon \subseteq \Upsilon'$ imply $(\Gamma, \Upsilon') \in L_@^\exists$
- $\lambda_S : S \to \{A, E, U, X\}$ and $\lambda_C : C \to \{T, D\}$ are labelling functions.

We denote the *upclosure* (under $\subseteq$) of a set $\mathcal{B}$ of @-constraints by $\uparrow \mathcal{B}$.

We think of L_M as the "modal links" where $L_M^\forall$ links sequents to sets of conclusions of modal rules, and $L_M^\exists$ links conclusions (sets of sequents) to their individual elements. This reflects the fact that to declare a sequent Γ satisfiable, we need to select, for all rules applicable to this sequent (all $(\Gamma, \Psi) \in L_M^\forall$) one conclusion $\Delta \in \Psi$ (there exists $(\Psi, \Delta) \in L_M^\exists$) which is recursively satisfiable. Similarly, $L_@$ encodes the global constraints (which we later propagate) that ensure that a sequent is satisfiable. For example for the sequent $\Gamma = @_i A \vee @_j B$ to be satisfiable, *either* the @-constraint $@_i A$ *or* the @-constraint $@_j B$ needs to be satisfiable, so that $\{(\Gamma, @_i A), (\Gamma, @_i B)\} \subseteq L_@^\exists$, and we ask for the existence of a $L_@^\exists$-successor of Γ. The required upclosure corresponds to the fact that – in order to satisfy an @-constraint – it suffices to satisfy any larger constraint. Technically, requiring upclosure simplifies the definition of @-propagation below. To guarantee the satisfiability of @-constraints, we link every @-constraint to a set of sequents (one for each nominal) that stipulate the validity of these constraints. For example, the @-constraint $\Upsilon = @_i A, @_j B$ requires that A holds at i and B holds at j which stipulates that both $\Upsilon_i = i, A, \Upsilon$ and $\Upsilon_j = j, B, \Upsilon$ should be satisfiable. This is represented by stipulating that $\{(\Upsilon, \Upsilon_i), (\Upsilon, \Upsilon_j)\} \subseteq L_@^\forall$ where again $\forall$ indicates universal choice.

The role of the labelling functions is essentially for bookkeeping. The label $\lambda_S(\Gamma)$ of a sequent Γ indicates whether the sequent is satisfiable (E: a winning position for the existential player), unsatisfiable (A: a winning position for the universal player), unknown (U) or unexpanded (X). Similarly, the label of an @-constraint Υ indicates that this constraint is expanded (D for done) or unexpanded (T for todo).

Remark 9. In a concrete implementation of caching graphs, it is sufficient to represent the upwards closed sets $L_@^\exists(\Gamma) = \{\Upsilon \mid (\Gamma, \Upsilon) \in L_@^\exists\}$ by means of a set of generators, which dramatically reduces the size of caching graphs.

We now introduce a set of transitions between caching graphs that correspond to expansion of both sequents and @-constraints, propagation of @-constraints and updating of winning positions. We begin with sequent expansion.

Definition 10 (Sequent Expansion). Suppose $G = (S, C, L_M, L_@, \lambda_C, \lambda_S)$ is a caching graph. We put $G \to_E G'$ and say that G' arises from G through *sequent expansion* if there is an unexpanded sequent $\Gamma \in S$ (i.e. $\lambda_S(\Gamma) = X$), and G' arises from G by inserting all relations (Γ, Ψ) where $\Gamma/\Psi \in T(\mathcal{R})$ is a rule into $L_M^\forall$, all ensuing relations (Ψ, Δ) with $\Delta \in \Psi$ to $L_M^\forall$, updating S to contain new sequents Δ that have been encountered in this procedure and setting their status to unexpanded (i.e. $\lambda_S(\Delta) = X$), equipping them with all @-constraints that contain $\bullet$ and finally marking Γ as unknown ($\lambda_S(\Gamma) = U$).

The situation is somewhat dual for @-constraints, where the universal links are added by a simple expansion process, but the existential links arise via propagation. Given that satisfiability of a sequent is conditional on the satisfiability of one of the associated @-constraints, we need a mechanism to check their satisfiability. In a nutshell, for an @-constraint to be satisfiable, we need to check, for each nominal, that the formulas deemed to be valid at this nominal are jointly satisfiable. While this results in an (ordinary) sequent, this process may uncover more constraints, and we therefore need to remember the set of @-constraints that we started out with. Formally, expansion of @-constraints takes the following form:

Definition 11 (@-Expansion). Suppose $G = (S, C, L_M, L_@, \lambda_S, \lambda_C)$ is a caching graph. We put $G \rightarrow_{@E} G'$ and say that G' arises from G through *@-expansion* if there exists a 'todo'-constraint $\Upsilon \in C$ (i.e. $\lambda_C(\Upsilon) = T$ and G' arises from G by inserting all sequents Γ for which $\Upsilon/\Gamma \in T(@)$ to $L^{\exists}_{\forall}(\Upsilon)$ and adding all new sequents to S, marking them as unexpanded ($\lambda_C(\Gamma) = X$, equipping new sequents with all @-constraints containing $\bullet$, and finally marking Υ as done ($\lambda_C(\Upsilon) = D$).

Informally, every @-constraint Υ specifies, for each nominal i, a set of formulas that are to be valid at i, which are collected in the @-demands. As the expansion of these formulas may unearth further @-formulas (possibly involving nominals distinct from i), the original @-constraint Υ is remembered in the @-demand. For the existential @-links the situation is more complicated, as @-links emanating from a sequent describe constraints (sets of @-prefixed formulas) that need to be satisfied for the sequent to be satisfiable. As @-prefixed formulas are either globally true or globally false, these constraints must hold at *all* points of a putative model. This necessitates distributing those constraints from one node of a tableau graph to the others. In general, every tableau node comes with a finite number of @-constraints where each particular constraint represents one requirement under which the associated sequent is satisfiable, such $@_i A$ or $@_j B$ for the above-mentioned example sequent $@_i A \vee @_j B$. As a consequence, we need to analyse the universal / existential branching structure of the caching graph during the propagation phase. As we are dealing with a possibly circular graph (due to the global assumptions), propagation is formalised as a *greatest* fixpoint computation.

Definition 12 (@-propagation). Let $\mathcal{A}$ denote the set of all @-constraints that can be formed in the closed set $\mathcal{C}$. Suppose $G = (S, C, L_M, L_@, \lambda_S, \lambda_C)$ is a caching graph and $R \subseteq S \times \mathcal{A}$. The set $C_R(\Gamma)$ of *R-constraints* of Γ consists of all @-constraints of the form $\Upsilon_1, \dots, \Upsilon_k$ such that for some $(\Delta_1, \dots, \Delta_k) \in \prod_{(\Gamma,\Psi) \in L^{\forall}_M} \Psi$, $(\Delta_i, \Upsilon_i) \in R$ and $\lambda_S(\Delta_i) \neq A$ for $i = 1, \dots, k$. In other words, an R-constraint of Γ collects, for every rule applicable to Γ, one constraint of one rule conclusion. In particular, $C_R(\Gamma) = \emptyset$ if $(\Gamma/\emptyset) \in L^{\forall}_M$ (i.e. Γ is inconsistent) and $C_R(\Gamma) = \{\emptyset\}$ in case no rule is applicable to Γ. Recall that $L^{\exists}_@$ is maintained as an upclosed relation of type $S \times \mathcal{A}$. Thus, let $\mathcal{R} = \{R \subseteq L^{\exists}_@ \mid R \text{ upclosed}\}$, and define a monotone operator $W_@ : \mathcal{R} \to \mathcal{R}$ by

$$W_@(R)(\Gamma) = \begin{cases} \uparrow \{\{\bullet\}\} & (\lambda_S(\Gamma) = X) \\ \uparrow \{@\Gamma, \Theta_1, \Theta_2, \mid \Theta_1 \in R(\Gamma), \Theta_2 \in C_R(\Gamma)\} & (\text{otherwise}) \end{cases}$$

where $@\Gamma = \{@_i A \mid i, A \subseteq \Gamma \text{ and } A \text{ not an @-formula}\} \cup \{A \in \Gamma \mid A \text{ an @-formula}\}$ so that to an expanded sequent, we associate its own @-formulas together with one previously computed constraint and one constraint that is propagated upwards from its children. We put $G \rightarrow_{@P} G'$ if $G' = (S, C', L_M, L'_@, \lambda_S, \lambda'_C)$ where

- $C' = C \cup \{\Upsilon \mid \exists \Gamma \in S\ ((\Gamma, \Upsilon) \in \nu W_@)\}$,
- $L^{\exists\,\prime}_@ = \nu W_@$ and $L^{\forall\,\prime}_@ = L^\forall_@$
- $\lambda'_C(\Upsilon) = \lambda_C(\Upsilon)$ if $\Upsilon \in C$ and $\lambda'_C(\Upsilon) = T$, otherwise

and say that G' arises from G through @*-propagation*.

Some comments are in order regarding the above definition of @-propagation. Updating $L^\exists_@$ requires us to compute an upclosed relation $R \subseteq S \times \mathcal{A}$; because constraints grow monotonically (due to sequent expansion), we will have $R \subseteq L^\exists_@$ (possibly throwing out some smaller constraints). To propagate @-constraints from the children of a sequent Γ up to Γ itself, note that Γ is satisfiable if for all applicable rules Γ/Ψ, there exists at least one satisfiable conclusion in $\Delta \in \Psi$. In particular, one of the @-constraints associated with Δ needs to be satisfiable. In other words, for Γ to be satisfiable it is necessary to be able to simultaneously select one @-constraint $\Upsilon \in R(\Delta)$ from one of the conclusions $\Delta \in \Psi$ for each of the rules Γ/Ψ that are applicable to Γ. If we think of R as defining an over-approximation of @-constraints, we keep for each Γ only those constraints that contain the @-formulas of Γ and one of the R-constraints of Γ. This is precisely the effect of one application of $W_@$, and we compute the greatest fixpoint of $W_@$ to propagate this information across cycles in the tableau graph.

The final crucial step is the updating of winning positions. Here, the intuition is that a given sequent is satisfiable if we can select a *complete* set of @-constraints so that all @-demands of this set are satisfiable – i.e. for each of the @-demands, we must be able to (recursively) pinpoint a complete set of @-constraints for which the same condition holds. We call a set of @-constraints complete if it represents full information, that is, collects all constraints that ensure that – if these constraints are satisfied – the sequent under consideration does not have a closed tableau. This is where $\bullet$ comes in: @-constraints that do not include $\bullet$ are complete. On the other hand, $\forall$ can win from a given sequent if all of the (possibly still incomplete) @-constraints (recursively) have at least one unsatisfiable @-demand.

Definition 13 (Position Propagation). Suppose that $G = (S, C, L_M, L_@, \lambda_S, \lambda_C)$ is a caching graph. By abuse of notation, write

$$U = \lambda_S^{-1}(U) \quad E = \lambda_S^{-1}(E) \text{ and } A = \lambda_S^{-1}(A)$$

for the sets of sequents that are labelled with U, E and A respectively. Define two monotone operators $M, W : \mathcal{P}(U) \to \mathcal{P}(U)$ by

$$M(X) = \{\Gamma \in U \mid \forall (\Gamma, \Upsilon) \in L^\exists_@\ \exists (\Upsilon, \Delta) \in L^\forall_@\ (\Delta \in X \cup A)\}$$
$$W(X) = \{\Gamma \in U \mid \exists (\Gamma, \Upsilon) \in L^\exists_@\ (\bullet \notin \Upsilon) \text{ and } \forall (\Upsilon, \Delta) \in L^\forall_@\ (\Delta \in X \cup E)\}.$$

We put $G \rightarrow_P G'$ if $G' = (S, C, L_M, L_@, \lambda'_S, \lambda_C)$ where $\lambda'_S(\Gamma) = A$ if $\Gamma \in A \cup \mu M$, $\lambda'_S(\Gamma) = E$ if $\Gamma \in E \cup \nu W$ and $\lambda'_S(\Gamma) = \lambda_S(\Gamma)$, otherwise, and say that G' arises from G through *position propagation*.

We now have all ingredients in place to describe the algorithm for deciding the satisfiability of a sequent $\Gamma \subseteq \mathcal{H}(\Lambda)$. This algorithm non-deterministically applies expansion, propagation and update steps until the initial sequent is either marked A (unsatisfiable) or E (satisfiable).

Algorithm 14. Decide whether Γ_0 is satisfiable in $\mathsf{Mod}(\Xi)$.

```
1. Initialise: put $G = (\{\Gamma_0, \Xi\}, \uparrow \{\{\bullet\}\}, \emptyset, L_@, \lambda_S, \lambda_C)$ where
   – $\lambda_S(\Gamma_0, \Xi) = X$, and $\lambda_C(\Upsilon) = T$ everywhere;
   – $L_@^\exists$ is total and $L_@^\forall = \emptyset$.
2. While $(\lambda_S(X)^{-1} \neq \emptyset)$ or $(\lambda_C^{-1}(T) \neq \emptyset)$ do
   (a) choose $G'$ with $G \to_E G'$ or $G \to_{@E} G'$ and let $G := G'$;
   (b) (optional) choose $G'$ with $G \to_{@P} G'$ and let $G := G'$;
   (c) (optional)
         – choose $G'$ with $G \to_P G'$ and let $G := G'$;
         – return 'yes' if $\lambda_S(\Gamma) = E$ and 'no' if $\lambda_S(\Gamma) = A$.
3. Find $G'$ with $G \to_{@P} G'$, let $G := G'$, and continue with Step 2.
4. Find $G'$ with $G \to_P G'$ and let $G := G'$.
5. Return 'yes' if $\lambda_S(\Gamma) = E$ and 'no' if $\lambda_S(\Gamma) = A$.
```

In the above formulation, the algorithm nondeterministically expands sequents or @-constraints and interleaves @-propagation and position update. Since @-propagation may create new @-constraints, we need to make sure that all @-constraints are eventually created, which is ensured by going back to Step 2 after @-propagation in Step 3. This procedure terminates, after at most exponentially many steps, as there are at most exponentially many @-constraints and sequents (measured in the size of the initial sequent Γ_0 and the TBox Ξ), and the final position update ensures that all sequents are marked accordingly. Note that we may terminate at any time after the initial sequent has been marked as either satisfiable or unsatisfiable after a position update.

5 Correctness and Completeness

We begin by showing that a sequent marked as satisfiable by Algorithm 14 is indeed satisfiable. This necessitates the construction of a satisfying model, which is based on a *named tableau graph*. Simply put, a named tableau graph consists of sequents Γ so that for every rule applicable to Γ, one of the conclusions occurs in the tableau graph, and is connected to Γ. In order to also satisfy @-formulas, we require that the tableau graph be *named*, as introduced next.

Definition 15. A *tableau graph* over a finite set S of sequents is a graph $G_T = (S, L)$ where $L \subseteq (S \times \mathcal{P}(S)) \cup (\mathcal{P}(S) \times S)$ is such that

- for all $\Gamma \in S$ and all $\Gamma/\Psi \in T(\mathcal{R})$ there exists $\Delta_{(\Gamma,\Psi)} \in \Psi$ such that $L = \{(\Gamma, \Psi) \mid \Gamma/\Psi \in T(\mathcal{R})\} \cup \{(\Psi, \Delta_{(\Gamma,\Psi)}) \mid \Gamma/\Psi \in T(\mathcal{R})\}$.

We say that (S, L) is a *named* tableau graph if additionally

- for each $i \in N$, there exists exactly one $\Gamma_i \in S$ with $i \in \Gamma$, and

– for all $\Gamma \in S$ and all $@_i A \in \Gamma$ we have $A \in \Gamma_i$.

The crucial stepping stone in the correctness proof for Algorithm 14 is the fact that we can construct a satisfying model based on a named tableau graph.

Lemma 16 (Model Existence Lemma). *If G_T is a named tableau graph over a set S of sequents, there exists a coalgebra structure $\sigma : W \to TW$ on the set of states contained in S and a hybrid valuation $\pi : \mathsf{N} \cup \mathsf{P} \to \mathcal{P}(W)$ such that $\Gamma \in [\![A]\!]_{(W,w,\pi)}$ for all $A \in \Gamma$.*

In order to be marked as satisfiable by a position update step in Algorithm 14, we need to be able to select a $\bullet$-free @-constraint for every satisfiable sequent. This entails the existence of a tableau graph for this sequent, and we will later merge these graphs.

Lemma 17. *Suppose that during the execution of Algorithm 14, $(\Gamma, \Upsilon) \in L_{@}^{\exists}$ with $\bullet \notin \Upsilon$ for a caching graph G. Then there exists a (not necessarily named) tableau graph (S, T) with $\Gamma \in S$.*

The first part of the correctness assertion can now be established as follows:

Lemma 18 (Completeness). *Every sequent marked 'satisfiable' by Algorithm 14 is satisfiable in* $\mathsf{Mod}(\Xi)$.

The second half of the correctness of Algorithm 14 needs the following preliminary lemma that shows that satisfiable sequents have satisfiable @-constraints.

Lemma 19. *Throughout the construction of the caching graph $G = (S, C, L_M, L_@, \lambda_S, \lambda_C)$ by Algorithm 14, it holds that for every satisfiable sequent $\Gamma \in S$ there exists an @-constraint $\Upsilon \in C$ such that $(\Gamma, \Upsilon) \in L_{@}^{\exists}$, $@\Gamma \subseteq \Upsilon$, and $\Upsilon \setminus \{\bullet\}$ is satisfiable.*

With the help of the last lemma, the second half of correctness of Algorithm 14 can now be established as follows:

Lemma 20. *Every sequent marked 'unsatisfiable' by Algorithm 14 is unsatisfiable in* $\mathsf{Mod}(\Xi)$.

Finally, we need to establish that Algorithm 14 in fact marks the initial sequent Γ_0, Ξ as either satisfiable or unsatisfiable, which only requires proof if Algorithm 14 terminates in Step 5. This rests on the final position update, and we first show that every @-constraint is 'morally' complete, that is, can be turned into a complete @-constraint by removing $\bullet$. This trivialises the condition $\bullet \notin \Upsilon$ in the definition of position update, which is used in the following lemma.

Lemma 21. *If Algorithm 14 terminates in Step 5, then the following holds for all $\Gamma \in S$: If $(\Gamma, \Upsilon) \in L_{@}^{\exists}$ then $(\Gamma, \Upsilon \setminus \{\bullet\}) \in L_{@}^{\exists}$.*

Finally, we show that Algorithm 14 always delivers a result (which is correct by Lemma 20 and Lemma 18), and thus finally confirm correctness.

Lemma 22. *When Algorithm 14 terminates in Step 5, each sequent is either marked satisfiable or unsatisfiable.*

Correctness of Algorithm 14 is a consequence of Lemma 18, Lemma 20 and Lemma 22.

Theorem 23. *For any given sequent Γ, Algorithm 14 delivers the answer 'yes' if Γ is satisfiable in* $\mathsf{Mod}(\Xi)$ *and the answer 'no' otherwise.*

6 Complexity

We proceed to analyse the runtime of the global caching algorithm, under suitable sanity assumptions on the set of modal rules. Specifically, in order to ensure that executions of the algorithm run in exponential time, we need to assume, as in [9,20], that our set $\mathcal{R}$ of one-step rules is EXPTIME-*tractable* in the following sense. To begin, a *demand* of a sequent Δ is a sequent $\Gamma_i\sigma$, $i \geq 1$, where $\Gamma_0/\Gamma_1 \dots \Gamma_n \in \mathcal{R}$ is a modal rule and σ is a substitution such that $\Gamma_0\sigma \subseteq \Delta$ and σ does not identify any two formulas in Γ_0. In this case, σ *matches* $\Gamma_0/\Gamma_1 \dots \Gamma_n$ to Δ. Then, we say that $\mathcal{R}$ is EXPTIME-tractable if there exists a coding of the rules such that all demands of a sequent can be generated by rules with codes of polynomially bounded size, validity of codes and membership of a sequent in the set of premises of a coded rule are decidable in EXPTIME, and matching substitutions for a given rule code/sequent pair can be enumerated in exponential time.

Theorem 24. *If the given set $\mathcal{R}$ of one-step rules is* EXPTIME*-tractable, then every execution of the global caching algorithm (Algorithm 14) runs in* EXPTIME.

We emphasize explicitly that, although the global caching algorithm is non-deterministic, it does not have any inherent non-determinism: *every* terminating execution yields the correct answer, so that the non-determinism works in favour of the implementer, who now has a chance to achieve improved average case behaviour by using suitable heuristics in his strategy for choosing expansion, propagation, and update steps. In particular, the above theorem does reprove the tight EXPTIME bound from [20].

7 Conclusions

We have presented an optimal tableau algorithm for hybrid modal logic over arbitrary TBoxes that is applicable to all (hybrid) logics with coalgebraic semantics. Instantiated to the modal logic K or a multi-modal variant, such as the description logic $\mathcal{ALCO}$, this provides, to our knowledge, the first purely syntax driven and backtracking-free tableau algorithm that realizes optimal (EXPTIME) complexity bounds. However, the scope of the coalgebraic framework is much broader, and the built-in parametricity uniformly provides us with optimal tableau-based decision procedures, e.g., for hybrid graded modal logic (or the description logic $\mathcal{ALCOQ}$), hybrid probabilistic modal logic, or hybrid logics for coalitional power in games. The compositionality of coalgebraic logics [18] in particular allows us to obtain optimal tableau algorithms for logics that mix the above features (see [20] for examples). The most pressing research concern at this point is, of course, experimental evaluation, which is the subject of ongoing work, and we plan to extend the CoLoSS system [5] that already implements the (modal) proof rules for a large variety of logics.

References

1. C. Areces and B. ten Cate. Hybrid logics. In P. Blackburn, J. van Benthem, and F. Wolter, eds., *Handbook of Modal Logic*. Elsevier, 2007.

2. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, eds. *The Description Logic Handbook*. Cambridge University Press, 2003.
3. T. Bolander and P. Blackburn. Termination for hybrid tableaus. *J. Log. Comput.*, 17:517–554, 2007.
4. T. Bolander and T. Braüner. Tableau-based decision procedures for hybrid logic. *J. Log. Comput.*, 16(6):737–763, 2006.
5. G. Calin, R. Myers, D. Pattinson, and L. Schröder. CoLoSS: The coalgebraic logic satisfiability solver (system description). In *Methods for Modalities, M4M-5*, ENTCS. Elsevier, 2008. To appear.
6. C. Cîrstea, C. Kupke, and D. Pattinson. EXPTIME tableaux for the coalgebraic μ-calculus. In *Computer Science Logic, CSL 09*, vol. 5771 of *LNCS*, pp. 179–193. Springer, 2009.
7. G. D'Agostino and A. Visser. Finality regained: A coalgebraic study of Scott-sets and multisets. *Arch. Math. Logic*, 41:267–298, 2002.
8. F. M. Donini and F. Massacci. EXPTIME tableaux for ALC. *Artif. Intell.*, 124:87–138, 2000.
9. R. Gore, C. Kupke, and D. Pattinson. Optimal tableau algorithms for coalgebraic logics. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS 10*, LNCS. Springer, 2010. To appear.
10. R. Goré and L. Nguyen. EXPTIME tableaux for $\mathcal{ALC}$ using sound global caching. In *Description Logics, DL 07*, vol. 250 of *CEUR Workshop Proceedings*, 2007.
11. R. Goré and L. Nguyen. EXPTIME tableaux with global caching for description logics with transitive roles, inverse roles and role hierarchies. In *Automated Reasoning with Analytic Tableaux and Related Methods, TABLEAUX 07*, vol. 4548 of *LNCS*, pp. 133–148. Springer, 2007.
12. I. Horrocks and P. F. Patel-Schneider. Optimising description logic subsumption. *J. Logic Comput.*, 9:267–293, 1999.
13. K. Larsen and A. Skou. Bisimulation through probabilistic testing. *Inf. Comput.*, 94:1–28, 1991.
14. R. Myers, D. Pattinson, and L. Schröder. Coalgebraic hybrid logic. In *Foundations of Software Science and Computation Structures, FoSSaCS 2009*, vol. 5504 of *LNCS*, pp. 137–151. Springer, 2009.
15. D. Pattinson. Coalgebraic modal logic: Soundness, completeness and decidability of local consequence. *Theoret. Comput. Sci.*, 309:177–193, 2003.
16. D. Pattinson and L. Schröder. Cut elimination in coalgebraic logics. *Inf. Comput.* To appear.
17. L. Schröder. A finite model construction for coalgebraic modal logic. *J. Log. Algebr. Prog.*, 73:97–110, 2007.
18. L. Schröder and D. Pattinson. Modular algorithms for heterogeneous modal logics. In *Automata, Languages and Programming (ICALP 07)*, vol. 4596 of *LNCS*, pp. 459–471. Springer, 2007.
19. L. Schröder and D. Pattinson. PSPACE bounds for rank-1 modal logics. *ACM Trans. Comput. Log.*, 10:13:1–13:33, 2009. Earlier version in LICS 06.
20. L. Schröder, D. Pattinson, and C. Kupke. Nominals for everyone. In *International Joint Conferences on Artificial Intelligence, IJCAI 09*, pp. 917–922. AAAI Press, 2009.
21. W. Thomas. On the synthesis of strategies in infinite games. In *Symposium on Theoretical Aspects of Computer Science, STACS 95*, vol. 900 of *LNCS*, pp. 1–13. Springer, 1995.

A Appendix: Omitted Proofs

Proof of the model existence lemma (Lemma 16)

One first constructs a coherent coalgebra structure $w : W \to TW$ on W where coherence is as in [9]. This structure is then equipped with a coherent valuation $\pi : \mathsf{N} \cup \mathsf{P} \to \mathcal{P}(W)$ so that $\pi(i) = \Gamma_i$. The claim follows by induction on the structure of formulas as in *op.cit.*. □

Proof of Lemma 17

This follows by analysing the individual steps of the algorithm, where the step of interest is @-propagation. Note that Γ has a tableau graph if $\Gamma \in \nu W_T$ where $W_T : \mathcal{P}(S) \to \mathcal{P}(S)$ and

$$W_T(X) = \{\Gamma \in S \mid \forall \Gamma/\Psi \in T(\mathcal{R})\ \exists \Delta \in \Psi\ (\Delta \in X)\}.$$

We thus have to show that

$$X = \{\Gamma \in S \mid \exists (\Gamma, \Upsilon) \in L^{\exists}_{@}\ (\bullet \notin \Upsilon)\} \subseteq \nu W_T$$

which follows, by coinduction, if we can show that $X \subseteq W_T(X)$. The latter inclusion follows from the definition of W_T and $W_@$ by observing that $X = \{\Gamma \in S \mid \exists (\Gamma, \Upsilon) \in \nu W_@\ (\bullet \notin \Upsilon)\}$ since $\nu W_@ = L^{\exists}_{@}$. □

Proof of the completeness lemma (Lemma 18)

We show that whenever the algorithm marks a sequent Γ as satisfiable ($\lambda(\Gamma) = E$) then there exists a named tableau graph for Γ by analysing the different steps of the algorithm, and the claim then follows from Lemma 16. The step of interest here is position update, so let $G = (S, C, L_M, L_@, \lambda_S, \lambda_C)$ be a caching graph that has been constructed during the run of Algorithm 14 and suppose that $G \to_U G'$ and $G' = (S, C, L_M, L_@, \lambda'_S, \lambda_C)$. We may assume inductively that $\lambda_C(\Gamma) = E$ implies that Γ has a named tableau graph. Now consider $\Gamma \in \lambda_S^{-1}(U)$ such that $\lambda'_S(\Gamma) = E$; we need to show that Γ has a named tableau graph. In this case, $\Gamma \in \nu W$, with $W : \mathcal{P}(U) \to \mathcal{P}(U)$ as in Definition 13. We consider the following two-player game, played on G with initial position Γ: the existential player (Eloise) may move from a sequent Δ to an @-constraint Υ with $\bullet \notin \Upsilon$ if $(\Delta, \Upsilon) \in L^{\exists}_{@}$, and the universal player (Abelard) may move from an @-constraint Υ to a sequent Δ if $(\Upsilon, \Delta) \in L^{\forall}_{@}$. Plays are lost by the player who cannot move, and Eloise wins all infinite games. The fact that $\Gamma \in \nu W$ entails that Eloise has a winning strategy in the above game. We know by Lemma 17 that each sequent encountered along this winning strategy has a tableau graph. These tableau graphs can be merged to form a named tableau graph. □

Proof of Lemma 19

Induction over the steps of the algorithm, where all induction steps are trivial except the one for @-propagation. Thus, let $G = (S, C, L_M, L_@, \lambda_S, \lambda_C)$ be the caching graph before application of an @-propagation step. Let $\mathit{\Gamma}$ be satisfiable, and let $\mathit{\Upsilon}$ be the $@\mathcal{C}$-theory of a model containing a state that satisfies $\mathit{\Gamma}$, so that in particular $@\mathit{\Gamma} \subseteq \mathit{\Upsilon}$. Put

$$R(\mathit{\Delta}) = \{\mathit{\Upsilon} \cup \{\bullet\}\}$$

in case $\mathit{\Delta}$ is satisfied in some state of M, and $R(\mathit{\Delta}) = \emptyset$ otherwise. We will be done once we show that in the notation of Definition 12, we have $R \subseteq \nu W_@$. By coinduction, it suffices to prove $R \subseteq W(R)$, i.e. $R(\mathit{\Delta}) \subseteq W_@(R)(\mathit{\Gamma})$ for all $\mathit{\Delta}$. This is trivial if $\mathit{\Delta}$ is not satisfied in M. Thus let $\mathit{\Delta}$ be satisfied in a state of M, so that $R(\mathit{\Delta}) = \{\mathit{\Upsilon} \cup \{\bullet\}\}$. If $\lambda_s(\mathit{\Gamma}) = X$ then trivially $\mathit{\Upsilon} \cup \{\bullet\} \in W(R)(\mathit{\Gamma})$. Otherwise, because $\mathit{\Delta}$ is satisfiable in M, we have $(\mathit{\Xi}_1, \dots, \mathit{\Xi}_k) \in \prod_{(\mathit{\Delta},\mathit{\Psi}) \in L_M^{\forall}} \mathit{\Psi}$ such that $\mathit{\Xi}_i$ is satisfiable in M for $i = 1, \dots, n$, and hence $R(\mathit{\Xi}_i) = \mathit{\Upsilon} \cup \{\bullet\}$. Moreover, $@\mathit{\Delta} \subseteq \mathit{\Upsilon}$, so that indeed $\mathit{\Upsilon} \cup \{\bullet\} \in W_@(R)(\mathit{\Delta})$. □

Proof of Lemma 20

Let $\mathit{\Gamma}$ be marked A in an update step from $G = (S, C, L_M, L_@, \lambda_S, \lambda_C)$ to $G' = (S, C, L_M, L_@, \lambda'_S, \lambda_C)$ as in Definition 13. By induction over the number of steps in the algorithm, we assume that in G, A consists only of unsatisfiable sequents. We proceed by a further induction over the least fixed point μM, and hence have that for all $\mathit{\Upsilon}$ such that $(\mathit{\Gamma}, \mathit{\Upsilon}) \in L_@^{\exists}$, there exists $(\mathit{\Upsilon}, \mathit{\Delta}) \in L_@^{\forall}$ such that $\mathit{\Delta}$ is unsatisfiable which, by definition of @-expansion, just means that $\mathit{\Upsilon} \setminus \{\bullet\}$ is unsatisfiable. By Lemma 19, it follows that $\mathit{\Gamma}$ is unsatisfiable. □

Proof of Lemma 21

Let $G = (S, C, L_M, L_@, \lambda_S, \lambda_C)$ be the caching graph reached upon termination. Put

$$R = \{(\mathit{\Gamma}, \mathit{\Upsilon} - \{\bullet\}) \mid (\mathit{\Gamma}, \mathit{\Upsilon}) \in L_@^{\exists}\}.$$

In the notation of Definition 12, we are done once we show that $R \subseteq L_@^{\exists} = \nu W_@$. By coinduction, it suffices to show $R \subseteq W_@(R)$. Thus let $(\mathit{\Gamma}, \mathit{\Upsilon}) \in L_@^{\exists}$; we have to show $\mathit{\Upsilon} \setminus \{\bullet\} \in W(R)(\mathit{\Gamma})$. Now by the fixed point property of $\nu W_@$, we have a decomposition

$$\mathit{\Upsilon} = @\mathit{\Gamma}, \mathit{\Theta}_1, \mathit{\Theta}_2, \mathit{\Theta}_3$$

where $\mathit{\Theta}_1 \in L_@^{\exists}(\mathit{\Gamma})$, $\mathit{\Theta}_2 \in C_{L_@^{\exists}}(\mathit{\Gamma})$, and $\mathit{\Theta}_3 \in \mathcal{A}$. Then we have

$$\mathit{\Upsilon} \setminus \{\bullet\} = @\mathit{\Gamma}, \mathit{\Theta}_1 \setminus \{\bullet\}, \mathit{\Theta}_2 \setminus \{\bullet\}, \mathit{\Theta}_3 - \{\bullet\}$$

where now $\mathit{\Theta}_1 \setminus \{\bullet\} \in R(\mathit{\Gamma})$, $\mathit{\Theta}_2 \setminus \{\bullet\} \in C_R(\mathit{\Gamma})$, and $\mathit{\Theta}_3 \setminus \{\bullet\} \in \mathcal{A}$, so that indeed $\mathit{\Upsilon} \setminus \{\bullet\} \in W_@(R)(\mathit{\Gamma})$ as required. □

Proof of Lemma 22

Let $G = (S, C, L_M, L_@, \lambda_S, \lambda_C)$ be the caching graph reached upon termination. In the notation of Definition 13, we have to show that $U = \emptyset$. Since sequent expansion is not applicable to G, we have $S = E \cup A \cup U$. By Lemma 21, the fixed points μM and νN determine the positions in U for which $\forall$ and $\exists$, respectively, have history-free winning strategies in an infinite reachability game played on $S \cup C$, where positions in S belong to $\exists$ and positions in C belong to $\forall$. Positions in A and E are winning positions for $\forall$ and $\exists$, respectively, and do not have any moves. Other positions have moves determined by $L_@$. All infinite games are won by $\exists$. By the history-free determinacy of such games [21], U is the disjoint union of μM and νN, and because position update does not apply to G, both these fixed points must be empty. $\Box$

Proof of Theorem 24

Using rule codes in a polynomial-size representation of sets of rule conclusions, we can ensure that the caching graph has only exponentially many candidate nodes (sequents, @-constraints, and sets of rule conclusions), and hence takes at most exponentially many expansion, propagation, or update steps. All fixed point computations that occur are on complete orders of at most exponential size, and hence all steps can be performed in EXPTIME. $\Box$