Optimizing Coalgebraic Modal Logic Reasoning

Daniel Hausmann*

Department of Computer Science, Universität Bremen
hausmann@informatik.uni-bremen.de

Abstract. The framework provided by coalgebraic modal logics offers broadly applicable coalgebraic semantics and an ensuing general treatment of modal sequent and tableau calculi while covering a wide variety of logics ranging from graded and probabilistic modal logic to coalition logic and conditional logics. Here we discuss generic optimisation strategies that may be employed to improve the performance of a global caching algorithm that decides the satisfiability of coalgebraic modal logics. Specifically, we discuss and show the admissability of generalisations of such established strategies as *semantic branching* and *dependency directed backtracking* to coalgebraic modal logics. As a more advanced consideration, the flattened representation of the involved proof graph by a *proof formula* is shown to be sound and complete; this separation of *proof structure* from the actual content of the proof does not only enhance the performance of the propagation process, it also allows for further optimisation techniques such as *proof graph simplification* to be applied. The presented optimisation techniques are implemented as part of the examplary implementation, the *Coalgebraic Logic Satisfiability Solver*(*CoLoSS*).

1 Introduction

Coalgebraic modal logic has been established as a unifying framework for many modal logics [6], allowing for generic and efficient decision of satisfiability (and provability) of those modal logics. In this context, the *Coalgebraic Logic Satisfiability Solver*(*CoLoSS*[1]) serves as an examplary implementation of the developed algorithms and as a testsuite for their optimisations [1, 3]. The broader focus introduces new difficulties to the process of proving satisfiablity (or provablility) of a formula, especially since the considered logics do not necessarily branch only on disjunctions (or conjunctions respectively) but may also introduce branching through their modal rules. Recent research in the area of coalgebraic modal logics has shown that the satisfiability (and provability) of coalgebraic logics with global assumptions may be decided in EXPTIME by means of an optimal global caching algorithm, thus enabling support for (basic) description logics in the coalgebraic setting [2]. The introduced algorithm is based on graph rewriting,

* Work forms part of DFG-project *Generic Algorithms and Complexity Bounds in Coalgebraic Modal Logic* (SCHR 1118/5-1)

[1] available under http://www.informatik.uni-bremen.de/cofi/CoLoSS/

however it has the weakness that the propagation of satisfiability through the graph is realized by computing fixpoints of the complete graph w.r.t specific functionals.

This work aims at improving the performance of said algorithm in two directions:

1. Established optimisation techniques for specific logics (such as **K**) are generalized and the generalizations are shown to be correct, thus allowing for these techniques to be applied for any coalgebraic modal logic.
2. *Proof formulas*, a flattened representation of the central data structure of the global caching algorithm (i.e. the proof graph) are introduced. The process of propagation may then be realized as a simple process of propositional simplification in the proof formula (thus making it feasable to propagate after each step of expansion).

The paper is structured as follows: We first give a short overview over coalgebraic modal logics. Then we briefly recall an optimal global caching algorithm that decides the provability of coalgebraic modal logics (with global assumptions). As the central contribution of the paper, we then turn our attention to the optimisation of said algorithm, focussing on three particular optimisation techniques: Generic semantic branching, generic dependency directed backtracking and efficient propagation by the use of proof formulas. The admissabilty of each of these generic optimization techniques is proved.

2 Coalgebraic Modal Logic

We quickly recall the needed notions from the domain of coalgebraic modal logic [6]:

2.1 Notation

Given an endofunctor on sets T, a T-coalgebra $\mathcal{C}$ consists of a carrier-set C and a function $\gamma : C \to T(C)$.

The semantics of modal operators $\heartsuit \in \Lambda$ is then defined by means of *predicate liftings* $[\![\heartsuit]\!] : 2^n \to 2 \circ T^{op}$ where $2 : \mathbf{Set} \to \mathbf{Set}^{op}$ denotes the contravariant powerset functor.

Formulas are of the shape

$$\mathcal{F}(\Lambda) \ni A_1, \ldots, A_n := p \mid \neg A_1 \mid A_1 \wedge A_2 \mid A_1 \vee A_2 \mid \heartsuit(A_1, \ldots, A_n)$$

where $\heartsuit \in \Lambda$ is an n-ary modal operator.

The semantics with respect to a model coalgebra $M = (C, \gamma, \pi)$ (where π denotes a valuation of propositional atoms) is fixed as follows:

$$[\![\heartsuit(A_1, \ldots, A_n)]\!]_M = \gamma^{-1} \circ [\![\heartsuit]\!]_C([\![A_1]\!]_M, \ldots, [\![A_n]\!]_M)$$

A (Λ-)*sequent* Γ is a set of formulas (from $\mathcal{F}(\Lambda)$). We refer to the set of all sequents of a logic as Seq and to the set of all sets of sequents of a logic as $Prem$.

A *sequent rule* consists of its *premise* Σ (a set of sequents) and its *conclusion* Γ (one sequent). A rule $R = (\Sigma = \{\Gamma_1, \ldots, \Gamma_n\}, \Gamma)$ is usually presented as follows:

$$\frac{\Gamma_1 \qquad \ldots \qquad \Gamma_n}{\Gamma}$$

A rule application of rule R to sequent Γ is an instantiation of R s.t. the conclusion of the rule is equal to Γ.

A formula ϕ is provable if there is a model coalgebra $M = (C, \gamma, \pi)$ with a state $x \in C$ s.t. $[\![\phi]\!] \ni x$. A sequent $\Gamma = \{\phi_1, \ldots, \phi_n\}$ is provable iff $\bigvee_{i=1}^{n} \phi_i$ is provable. A premise is said to be provable iff all its sequents are provable.

A sound and complete algorithm for the decision of provability (and satisfiability) of formula of the according coalgebraic logic may be obtained if the propositional rules from Figures 1 and 2 are utilized together with the appropriate modal rule(s). Some examplary modal rules for different coalgebraic modal logics are depicted in Figure 3 (for more details, refer to [6]) while the modal rules of some conditional logics are introduced in Figure 4 (more details about optimisation of coalgebraic conditional logics can be found in [5, 3]). The collection of all currently utilized rules is referred to as $\mathcal{R}$.

$$\boxed{(\neg\vee)\frac{\Gamma, A \qquad \Gamma, B}{\Gamma, \neg(\neg A \vee \neg B)}}$$

Fig. 1. Branching propositional sequent rule

$$\boxed{(\mathrm{T})\frac{}{\Gamma, \top} \qquad (\mathrm{At})\frac{}{\Gamma, p, \neg p} \qquad (\neg\neg)\frac{\Gamma, A}{\Gamma, \neg\neg A} \qquad (\vee)\frac{\Gamma, A, B}{\Gamma, (A \vee B)}}$$

Fig. 2. Linear or closing propositional sequent rules

Take note that the rules from Figure 2 may be integrated into the process of normalisation, saturation or simpification (see 3.1) due to their linearity.

We introduce the following abbreviations in order to allow for a swift presentation of the modal rules of propositional and graded modal logic: Given a formula ϕ_i and $r_i \in \mathbb{Z}$ for all $i \in I$ as well as $k \in \mathbb{Z}$,

$$\sum_{i \in I} r_i \phi_i \geq k \equiv \bigwedge_{J \subseteq I, r(J) < k} (\bigwedge_{j \in J} \phi_j \rightarrow \bigvee_{j \notin J} \phi_j),$$

Modal Logic (Feature)	Modal Rule(s)
$\mathbf{M}$ ($\Box_{\mathbf{M}}$)	$\dfrac{A \to B}{\Gamma, \Box_{\mathbf{M}} A \to \Box_{\mathbf{M}} B}$
$\mathbf{K}$ ($\Box_{\mathbf{K}}$)	$\dfrac{\bigwedge_{i=1}^{n} A_i \to B}{\Gamma, \bigwedge_{i=1}^{n} \Box_{\mathbf{K}} A_i \to \Box_{\mathbf{K}} B}$
$\mathbf{KD}$ ($\Box_{\mathbf{KD}}$)	$\dfrac{\bigwedge_{i=1}^{n} A_i \to B}{\Gamma, \bigwedge_{i=1}^{n} \Box_{\mathbf{KD}} A_i \to \Box_{\mathbf{KD}} B}$ $\qquad$ $\dfrac{\neg \bigwedge_{i=1}^{n} A_i}{\Gamma, \neg \bigwedge_{i=1}^{n} \Box_{\mathbf{KD}} A_i}$
Hennessy-Milner-Logic ($\Box^{a}_{\mathbf{HM}}$)	$\dfrac{\bigwedge_{i=1}^{n} A_i \to B}{\Gamma, \bigwedge_{i=1}^{n} \Box^{a}_{\mathbf{HM}} A_i \to \Box^{a}_{\mathbf{HM}} B}$
Coalition Logic ($\Box^{C}_{\mathbf{C}}$)	$\dfrac{\bigwedge_{i=1}^{n} A_i \to B \vee \bigvee_{j=1}^{m} C_j}{\Gamma, \bigwedge_{i=1}^{n} \Box^{C_i}_{\mathbf{C}} A_i \to \Box^{D}_{\mathbf{C}} B \vee \bigvee_{j=1}^{m} \Box^{N}_{\mathbf{C}} C_j}$ $m, n \geq 0$, C_i pairwise disjoint subsets of D.
Graded Modal Logic ($\Diamond^{i}_{\mathbf{G}}$)	$\dfrac{\sum_{i=1}^{n} A_i \leq \sum_{j=1}^{m} B_j}{\Gamma, \bigwedge_{i=1}^{n} \Diamond^{k_i}_{\mathbf{G}} A_i \to \bigvee_{j=1}^{m} \Diamond^{l_j}_{\mathbf{G}} B_j}$ $n, m \geq 0$, $\sum_{i=1}^{n} (k_i + 1) \geq 1 + \sum_{j=1}^{m} l_j$.
Probabilistic Modal Logic ($\Diamond^{p}_{\mathbf{P}}$)	$\dfrac{\sum_{i=1}^{n} A_i + u \leq \sum_{j=1}^{m} B_j}{\Gamma, \bigwedge_{i=1}^{n} \Diamond^{p_i}_{\mathbf{P}} A_i \to \bigvee_{j=1}^{m} \Diamond^{q_j}_{\mathbf{P}} B_j}$ $m, n \geq 0$, $m + n \geq 1$, $u \in \mathbb{Z}$, $\sum_{i=1}^{n} p_i + u \geq \sum_{j=1}^{m} q_j$ and $m = 0 \to \sum_{i=1}^{n} p_i + u > 0$.

Fig. 3. One-step complete, resolution closed modal rules for some coalgebraic modal logics

where $r(J) = \sum_{j \in J} r_j$. Furthermore, we use $\sum a_i \leq \sum b_j$ as an abbreviation for $\sum b_j - \sum a_i \geq 0$.

Modal Logic (Feature)	Modal Rule(s)
CK ($\Rightarrow_{\mathbf{CK}}$)	$\dfrac{A_0 \leftrightarrow \ldots \leftrightarrow A_n \qquad \neg B_1, \ldots, \neg B_n, B_0}{\Gamma, \bigwedge_{i=1}^{n} (A_i \Rightarrow_{\mathbf{CK}} B_i) \rightarrow (A_0 \Rightarrow_{\mathbf{CK}} B_0)}$
CK+CEM ($\Rightarrow_{\mathbf{CK_{CEM}}}$)	$\dfrac{A_0 \leftrightarrow \ldots \leftrightarrow A_n \qquad B_0, \ldots, B_k, \neg B_{k+1}, \ldots \neg B_n}{\Gamma, \bigwedge_{i=k+1}^{n} (A_i \Rightarrow_{\mathbf{CK_{CEM}}} B_i) \rightarrow \bigvee_{j=0}^{k} (A_j \Rightarrow_{\mathbf{CK_{CEM}}} B_j)}$
CK+CM ($\Rightarrow_{\mathbf{CK_{CM}}}$)	$\dfrac{\begin{array}{ll} A_i = A_j & \text{for } i,j \in l(v), v \in G \text{ initial} \\ \bigcup_{i \in l(v)} \{\neg B_i, \neg A_i\}, A_j & \text{for } v \in G \text{ and } j \in l(w) \\ & \text{for some successor } w \text{ of } v \\ \{\neg B_i \mid i \in I\}, D & \\ \bigcup_{i \in I} \{\neg B_i, \neg A_i\}, C & \\ \neg C, A_i & \text{for } i \in I \end{array}}{\Gamma, \bigwedge_{i \in I} (A_i \Rightarrow_{\mathbf{CK_{CM}}} B_i) \rightarrow (C \Rightarrow_{\mathbf{CK_{CM}}} D)}$ for an I-compatibility graph G
System S ($\Rightarrow_{SysS}$)	$\dfrac{\Delta_M(\nu(M)) \text{ for each } M \in \mathfrak{S}_{\Gamma_0}}{\underbrace{\Gamma, \bigwedge_{i \in I} (A_i \Rightarrow_{SysS} B_i) \rightarrow (A_0 \Rightarrow_{SysS} B_0)}_{\equiv: \Gamma_0}}$ where $\mathfrak{S}_{\Gamma_0}$ denotes all S-structures $(S, \preceq)$ over Γ_0, $\Delta_M([i]) \equiv \bigcup_{k \simeq i} \{\neg A_k, \neg B_k\}, \{A_j \mid i \prec j \vee j \notin S\}$, $\Delta_M([0]) \equiv \neg A_0, \bigcup_{0 \neq k \simeq 0} \{\neg A_k, \neg B_k\}, B_0, \{A_j \mid j \notin S\}$.

Fig. 4. One-step complete, resolution closed modal rules for some conditional logics

Furthermore, we allow for the treatment of global assumptions (which may be thought of as representing a TBox) by expanding the employed modal rules as follows: Given a set of global assumptions Δ and a modal rule $R = \Gamma_0, \ldots, \Gamma_n / \Gamma$, the expanded rule is defined as $R' = (\Delta \rightarrow \Gamma_0), \ldots, (\Delta \rightarrow \Gamma_n)/\Gamma$, i.e. we add the global assumptions to each newly constructed state.

The presence of global assumptions introduces the possibility of running into circularities during the course of of the proof so that a simple sequent calculus without blocking is no longer sufficient to solve the task (as it would possibly not terminate). Hence we introduce an algorithm that uses *global caching* in order to not only detect circular proof requirements but to also benefit from the improved performance that comes along with the use of (global) caching techniques.

3 Global Caching Algorithm for Coalgebraic Logics

We quickly recall the global caching algorithm that was introduced in [2] while treating provability of formulas here rather than satisfiability. However, a formula is satisfiable whenever its negation is not provable.

3.1 The proof graph and transitions

Given a sequent Γ, we define the set of all possible rule applications to Γ, $Rules(\Gamma) = \{(\Gamma, \Sigma) \mid \text{ is an instance of a rule } R\}$, the set of those sequents that appear in premises of rules that are applicable to Γ, $S(\Gamma) = \{\Gamma' \mid (\Gamma, \Sigma) \text{ is an instance of a rule } R \wedge \Gamma' \in \Sigma\}$, and the set of all selections $Selections(\Gamma) = \{(\Sigma, \Gamma') \mid (\Gamma, \Sigma) \in Rules(\Gamma) \wedge \Gamma' \in \Sigma\}$.

A *proof graph* is a tuple (A, E, U, X, L_1, L_2) where $A, E, U \subseteq Seq$ are sets of sequents, $X \subseteq Prems$ is a set of premises. The (partial) function L_1 assigns the set $Rules(\Gamma)$ to a sequent Γ whereas the (partial) function L_2 assigns the set $Selections(\Gamma)$ to a sequent Γ.

The sets A und E contain those sequents which are already known to be provable and improvable respectively. The set U contains the already expanded but yet undecided sequents whereas the set X keeps track of those sequents that have yet to be explored. The relations between sequents and premises are stored in the two functions L_1 and L_2.

Unexpanded sequents $\Gamma \in X$ (i.e. sequents to which no rule has been applied yet), are also called *open sequents*. A proof graph with $X = \emptyset$ is called *fully expanded*.

Any sequent to which either the rule (T) or the rule (At) may be applied is called a *successful leaf* of the proof graph.

Any sequent Γ to which no rule may be applied (i.e. Γ is not equal to the conclusion of any instance of any of the employed rules) is called an *unsuccessful leaf*.

For the sake of readability, we will sometimes show *simplified proof graphs* where only one rule application is considered for each sequent (such that only one – the interesting one – of the possible premises of each sequent is shown).

We now fix two effectively computable transitions between proof graphs:

- *Expand:* Given a sequent Γ from X, this transition leads from a graph g to the graph g' that is obtained from g by expanding Γ. To this end, X is extended by $S(\Gamma)$ and Γ is removed from X ($X' = (X \cup S(\Gamma)) \setminus \Gamma$). Furthermore, L_1 is extended by all possible rule applications for Γ ($L_1' = L_1 \cup AR(\Gamma)$). Finally, L_2 is extended by all possible selections for rule applications to Γ ($L_2' = L_2 \cup AS(\Gamma)$).
  In conclusion, this procedure applies all applicable rules to Γ, marks all resulting premises as not yet expanded, marks Γ as expanded and stores the implied transitions in L_1 and L_2.
  For a fixed sequent Γ and a graph g, we denote the expansion of Γ in g by $exp(g, \Gamma)$. If $g' = exp(g, \Gamma)$, we also write $g \xrightarrow{\Gamma}_E g'$.
- *Propagate:* This procedure evaluates the given graph as far as yet possible: The sets X, L_1 and L_2 remain unchanged. The set of provable sequents however is extended by the least fixpoint of a function M^L ($A' = A \cup \mu(M^L)$). The set of improvable sequents is extended by the greatest fixpoint of a function W^L ($E' = E \cup \nu(W^L)$). Finally, those sequents that are known to be either provable or not provable are removed from the set of undecided

sequents ($U' = U \setminus (A' \cup E')$).

The functions which are used for the fixpoint computations are defined as follows:

$$M^L(X) = \{\Gamma \in U \mid \exists(\Gamma, \Sigma) \in L_1.\forall(\Sigma, \Gamma') \in L_2.\Gamma' \in X \cup A\}$$
$$W^L(X) = \{\Gamma \in U \mid \forall(\Gamma, \Sigma) \in L_1.\exists(\Sigma, \Gamma') \in L_2.\Gamma' \in X \cup E\}$$

Given a graph g, we denote the graph obtained by propagating in g by $prp(g)$. If $g' = prp(g)$, we also write $g \rightarrow_P g'$.

3.2 The Algorithm

In order to show the provability of a formula ϕ, we start off with the sequent $\Gamma_0 = \{\phi\}$:

Algorithm: (Show provability of Γ_0)
1. Initialize: $A_0 = E_0 = \emptyset$, $U_0 = \{\Gamma_0\}$, $X_0 = S(\Gamma_0)$,
   $g = (A_0, E_0, U_0, X_0 Rules(\Gamma_0), Selections(\Gamma_0))$.
2. If $X = \emptyset$, return the current graph as result.
3. Otherwise select any sequent Σ from X. Compute $g' = exp(g, \Gamma)$ s.t. $g \stackrel{\Gamma}{\rightarrow}_E g'$.
   (Optionally: Compute $g'' = prp(g')$ s.t. $g' \rightarrow_P g''$;
   If $\Gamma_0 \in A'' \cup E''$, break and return the current graph as result.)
4. Set the current graph to g' (or g'' if a propagation step took place).
   Continue with 2.

After the loop has finished, propagate one final time and then let A (E) denote the set of provable (improvable) sequents of the resulting graph.
If $\Gamma_0 \in A$, return `True`, if $\Gamma_0 \in E$, return `False`, otherwise return `Undecided`.

Fig. 5. The global caching algorithm

In [2], this algorithm has been shown to be

a) sound and complete w.r.t. provability of formulas of the logic that is represented by the utilized set of rules $\mathcal{R}$,
b) of complexity EXPTIME, if the underlaying ruleset allows for it.

4 Optimisations

Even though the complexity of deciding provability (satisfiability) of modal formulas using the above algorithm is exponential, it is still possible to achieve reasonable performance by making use of several optimisation techniques, such as described for specific logics in [4]. Since the above algorithm makes use of

global caching in order to recognize circular proof requirements, it is already optimized in so far as that the provability (improvability) of any sequent Γ has to be shown only once during the course of a proof (as the status of this sequent is stored and may be looked up upon future occurances of Γ). We continue by describing further methods to modify the introduced algorithm such that even complex formulas may be efficiently treated:

First of all, we assume all appearing formulas to be normalized to truth, negation and generalized disjunction (where the collection of all disjuncts is treated as a set in order to remove duplicates and ordering) as indicated by the rules shown in Figure 6.

Syntactic construct	Normalisation
$\bot$	$\neg\top$
$\neg\neg\phi$	ϕ
$\phi \wedge \psi$	$\neg(\neg\phi \vee \neg\psi)$
$\phi \rightarrow \psi$	$\neg\phi \vee \psi$
$\phi \leftrightarrow \psi$	$(\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$
$\phi \vee \psi$	$\cup\{\phi, \psi\}$
$\cup\{\phi, (\cup\Gamma)\}$	$\cup(\{\phi\} \cup \Gamma)$

Fig. 6. Normalisation rules

Furthermore, the input formula is assumed to already contain only the standard modalities for each feature (e.g. $\Box_K\phi$ is assumed to be represented by $\neg\Diamond_K\neg\phi$). As another convention, we assume that sequents are always being *satured* (i.e. top-level disjunctions $\cup S$ are replaced by just the set S).
As usual, there is a list of all appearing formulas and a sequent is encoded as a string of bits, where the bit at position i indicates whether the sequent contains the i-th element of the list of formulas or not.
Another improvement over the algorithm from [2] is based on the following observation: If a sequent Γ has been shown to be provable, all sequents containing Γ will be provable as well. Thus it is admissable to soften the requirement in the functionals which are used to compute the fixpoints:

$$M^L(X)' = \{\Gamma \in U \mid \exists(\Gamma, \Sigma) \in L_1.\forall(\Sigma, \Gamma') \in L_2.\exists\Gamma'' \subseteq \Gamma'.\Gamma'' \in X \cup A\}$$
$$W^L(X)' = \{\Gamma \in U \mid \forall(\Gamma, \Sigma) \in L_1.\exists(\Sigma, \Gamma') \in L_2.\exists\Gamma'' \subseteq \Gamma'.\Gamma'' \in X \cup E\}$$

4.1 Simplification

As a first basic optimisation, simplifaction makes use of specific (propositional or modal) tautologies in order to deterministically rewrite any appearing formulas in a proof. The application of simplification does not invoke any branching and is thus a linear process.
A non-exhaustive list of some important simplification rules is shown in Figure 7.

Operator	Propositional Tautology
$\vee$	$\cup\{\top, ...\} \rightarrow \top$ $\cup\{\neg\top, \phi_1, \ldots, \phi_n\} \rightarrow \cup\{\phi_1, \ldots, \phi_n\}$ $\cup\{\phi, \neg\phi, \ldots\} \rightarrow \top$
$\vee\wedge$	$\cup\{\neg\phi, \neg(\neg\phi \vee \neg\psi)\} \rightarrow \psi$
$\Box_{\mathbf{K}}$	$\Box_{\mathbf{K}}\top \rightarrow \top$ $(\Box_{\mathbf{K}}\phi \wedge \Box_{\mathbf{K}}\psi \rightarrow \Box_{\mathbf{K}}(\phi \wedge \psi))$
$\Box_{\mathbf{KD}}$	$\Box_{\mathbf{KD}}\top \rightarrow \top$ $(\Box_{\mathbf{KD}}\phi \wedge \Box_{\mathbf{KD}}\psi \rightarrow \Box_{\mathbf{KD}}(\phi \wedge \psi))$ $\neg\Box_{\mathbf{KD}}\bot \rightarrow \top$
$\Box^a_{\mathbf{HM}}$	$\Box^a_{\mathbf{HM}}\top \rightarrow \top$ $(\Box^a_{\mathbf{HM}}\phi \wedge \Box^a_{\mathbf{HM}}\psi \rightarrow \Box^a_{\mathbf{HM}}(\phi \wedge \psi))$
$\Rightarrow_{\mathbf{CK}}$	$(\phi \Rightarrow_{\mathbf{CK}} \top) \rightarrow \top$
$\Rightarrow_{\mathbf{CK_{CEM}}}$	$(\phi \Rightarrow_{\mathbf{CK_{CEM}}} \top) \rightarrow \top$

Fig. 7. Examplary tautologies used for simplification

4.2 Generalized Semantic Branching

The technique of semantic branching [4] allows to ensure that created sub-proofs are strictly disjoint. This prevents duplicate proofs a formulas which might otherwise appear in both sub-proofs.

Propositional Semantic Branching In the propositional case, there is just one branching rule, the conjunction rule. A semantic version of this rule is shown in Figure 8. The justification for replacing the standard conjunction rule by the rule for semantic conjunction branching is simple:

Lemma 1. *A sequent is provable by syntactic propositional branching iff it is provable by semantic propositional branching.*

Proof. This fact follows from the propositional tautology

$$(A \wedge B) \leftrightarrow (A \wedge (A \rightarrow B)).$$

Syntactic branching	Semantic branching
$\dfrac{\Gamma, A \qquad \Gamma, B}{\Gamma, A \wedge B}$	$\dfrac{\Gamma, A \qquad \Gamma, \neg A, B}{\Gamma, A \wedge B}$

Fig. 8. Syntactic vs. Semantic branching on conjunctions

Modal Semantic Branching The general principle that can be extracted from propositional semantic branching is a follows:

A syntactic branching rule

$$\frac{\Gamma_1 \quad \ldots \quad \Gamma_i \quad \ldots \quad \Gamma_n}{\Gamma}$$

may be (repeatedly) replaced by a 'more semantic' branching rule

$$\frac{\Gamma_1 \quad \ldots \quad \Gamma_i' \quad \ldots \quad \Gamma_n}{\Gamma}$$

where $\Gamma_i' = \bigwedge_{t\in T} \Gamma_t \rightarrow \Gamma_i$ for some $T \subseteq \{1,\ldots,n\} \setminus \{i\}$.

The justification for this generalized semantic branching is as simple as in the propositional case:

Lemma 2. *A sequent is provable by syntactic modal branching iff it is provable by semantic modal branching.*

Proof. This fact follows from the propositional tautology

$(\bigwedge_{t\in T} \Gamma_t \wedge \Gamma_i) \leftrightarrow (\bigwedge_{t\in T} \Gamma_t \wedge (\bigwedge_{t\in T} \Gamma_t \rightarrow \Gamma_i))$.

Feature	Semantic Branching
$\Rightarrow_{\mathbf{CK}}$	$\dfrac{A_0 \leftrightarrow \ldots \leftrightarrow A_n \qquad \neg(A_0 \leftrightarrow \ldots \leftrightarrow A_n), \neg B_1, \ldots, \neg B_n, B_0}{\Gamma, \bigwedge_{i=1}^{n} (A_i \Rightarrow_{\mathbf{CK}} B_i) \rightarrow (A_0 \Rightarrow_{\mathbf{CK}} B_0)}$
$\Rightarrow_{\mathbf{CK_{CEM}}}$	$\dfrac{A_0 \leftrightarrow \ldots \leftrightarrow A_n \qquad \neg(A_0 \leftrightarrow \ldots \leftrightarrow A_n), B_0, \ldots, B_k, \neg B_{k+1}, \neg B_n}{\Gamma, \bigwedge_{i=k+1}^{n} (A_i \Rightarrow_{\mathbf{CK_{CEM}}} B_i) \rightarrow \bigvee_{j=0}^{k} (A_j \Rightarrow_{\mathbf{CK_{CEM}}} B_j)}$

Fig. 9. Specific semantic branching for examplary modal logics

4.3 Generalized Dependency Directed Backtracking

The technique of *controlled backtracking* (also known as dependency directed backtracking [4]) is in fact a special instance of caching: When this optimisation is employed, each sequent Γ depends on a set $\sharp(\Gamma)$ of sequents (namely on those sequents which introduced any formula $\phi \in \Gamma$ into the proof tree). Whenever a sequent turns out to be provable, its provability depends on possibly several sequents. These sequents are called the respective *dependency set* of $\{\top\}$, denoted by $\sharp(\{\top\})$ (these sequents are also known as the *promotors* of the successful leaf).

We have yet to formally define dependency sets, but we will rely on the following crucial fact:

Lemma 3. *Given a dependency set $\sharp(\{\top\})$ of a succesfull leaf, the first sequent Γ s.t. $\Gamma \in \sharp(\{\top\})$, is provable.*

Due to Lemma 3, it will not be necessary to treat any open sequents between the current successful node in the proof graph and the first node which is contained in the according dependency set.

To achieve this, a set of *current* dependency sequents is used, and whenever a sequent is provable, the current set of dependency sequents is defined as the set of dependency sequents of $\{\top\}$. Now all those open sequents which appear during the following backtracking through the proof tree and which are created by a sequent which is *not* in the current set of dependency states, may be directly set to *provable*.

If however a node in the proof graph which is contained in the current set of dependency states is reached, the current set of dependency states is set to $\emptyset$ and the proof may continue as usual.

Definition 1. *Depending on the shape of a proof tree, $\sharp(\Gamma)$ is defined as follows:*

- *If Γ is the root of the proof tree, $\sharp(\Gamma) = \Gamma$.*
- *If Γ is obtained from Γ' by means of normalisation, saturation or simplification, then $\sharp(\Gamma) := \sharp(\Gamma')$.*
- *If Γ is the only sequent in the premise of a rule application of a non-branching rule to Γ', then $\sharp(\Gamma) := \sharp(\Gamma')$*
- *If $\Gamma_1, \ldots, \Gamma_n$ are n sequents from the premise of the rule application of a branching rule to $\Gamma = \Gamma', \Gamma''$, s.t. $\Gamma_1, \ldots, \Gamma_n$ is constructed from Γ'', the dependancy set of any formula $\phi \in \bigcup_{i\in\{1,\ldots,n\}} \Gamma_i$ is defined as follows: if $\forall i \in \{1, \ldots, n\}.\phi \in \Gamma_i$, then $\sharp(\{\phi\}) := \sharp(\Gamma'')$, else $\sharp(\{\phi\}) := \Gamma$.*
- *Finally, $\sharp(\Gamma_1 \cup \Gamma_2) := \sharp(\Gamma_1) \cup \sharp(\Gamma_2)$*

Feature	Modal Rule	Dependencies
$\Box_{\mathbf{K}}$	$\dfrac{\overbrace{\bigwedge_{i=1}^{n} A_i \to B}^{\equiv:\Gamma_1}}{\Gamma, \underbrace{\bigwedge_{i=1}^{n} \Box_{\mathbf{K}} A_i \to \Box_{\mathbf{K}} B}_{\equiv:\Gamma'}}$	$\sharp(\Gamma_1) := \sharp(\Gamma')$
$\Rightarrow_{\mathbf{CK}}$	$\dfrac{\overbrace{A_0 \leftrightarrow \ldots \leftrightarrow A_n}^{\equiv:\Gamma_1} \quad \overbrace{\neg B_1, \ldots \neg B_n, B_0}^{\equiv:\Gamma_2}}{\Gamma, \underbrace{\bigwedge_{i=1}^{n} (A_i \Rightarrow_{\mathbf{CK}} B_i) \to (A_0 \Rightarrow_{\mathbf{CK}} B_0)}_{\equiv:\Gamma'}}$	$\sharp(\Gamma_1, \Gamma_2) := \Gamma, \Gamma'$

Fig. 10. Propagation of dependencies over some examplary modal features

Inituitively, a single application of for instance the modal rule of **K** does not *choose* any subformulas of its conclusionary sequent to generate new sequents

(this is already a consequence of the fact that this rule does not branch). Hence any introduced formulas will depend on those formulas on which the conclusionary sequent already depended. On the other hand, the modal rule of for instance **CK** selects some subformulas (the antecedents) from its conclusionary sequent to be in the first sequent of the premise and others (the consequents) to be in the second sequent of the premise, thus a choice is made and the introduced formulas will directly depend on the conclusionary sequent.

The following Lemmas establish that dependency directed backtracking as defined above is a sound optimisation technique for any coalgebraic model logic:

Lemma 4. *Assume an application of any (possibly branching) rule:*

$$\frac{\Gamma_1 \quad \dots \quad \Gamma_n}{\Gamma}$$

Assume a sequent Γ' such that $\Gamma \notin \sharp(\Gamma')$, but $\Gamma' \subseteq \Gamma_i$ for any $0 < i \leq n$. Then provability of Γ' implies the provability of all sequents Γ_i of the premise of the rule.

Proof. Let Γ' be provable. Since $\Gamma \notin \sharp(\Gamma')$, we have a non-selecting rule application (w.r.t. Γ'). For non-branching rules, the implication is trivial. For a branching rule, it follows directly from the definition of $\sharp(\_)$ and the fact that one Γ_j contains Γ', that all Γ_j contain Γ'. Hence all the sequents in the premise of the rule are provable.

Lemma 5. *Let the following be a (not yet fully explored) proof tree for Γ (where Γ^i_j denotes the sequent number j on level i; in case that $j > 1$ and $i > 1$, Γ^i_j is assumed to be an open sequent):*

$$\dfrac{\dfrac{\dfrac{\dfrac{\checkmark}{\Gamma^n_1}}{\dots}}{\Gamma^2_1} \quad \dots \quad \Gamma^2_i}{\Gamma^1_1} \quad \dots \quad \Gamma^1_j}{\Gamma}$$

Further, let $\sharp(\{\top\})$ denote the dependency set of the successful leaf. If for all Γ^i_1 with $0 < i \leq n$, $\sharp(\{\top\}) \not\subseteq \Gamma^i_1$, and if $\sharp(\{\top\}) \subseteq \Gamma$ (i.e. Γ is the first sequent contained in the dependency set of the successful leaf), then Γ^1_1 is provable (since any Γ^i_j with $i > 1$ is provable).

Proof. Lemma 4 ensures, that the provability of the successful leaf propagates to all the open sequents Γ^i_j for $i, j > 1$ (since $\sharp(\{\top\}) \not\subseteq \Gamma^{i-1}_j$). Thus we are not able to find any sequent of level > 1 that is not provable.

To summarize, once we found a successful leaf Γ^n_1, it is not necessary to treat the whole sub-proof which is induced by Γ^1_1, since Γ^1_1 is always provable in the above situation.

4.4 Proof Formulas and efficient propagation

Due to the inherent inefficiency of computing the propagation of provable sequents A (improvable sequents E) through the proof graph (all undecided sequents have to be traversed in order to compute the necessary fix-points), we devise the following more efficient method:

Definition 2. *A* position $p \in Pos = \{c_1 \dots c_n \mid c_1, \dots, c_n \in \mathbb{N}\} \cup \{[]\}$ *is a finite list of natural numbers.*

A proof formula *is defined as follows (for* $j, p \in Pos$*):*

$$\mathcal{PF} \ni \psi_1, \dots, \psi_n := a_j \mid \wedge\{\psi_1, \dots, \psi_n\} \mid \vee\{\psi_1, \dots, \psi_n\} \mid l_p$$

The subformula $\phi|_p$ *of a proof formula* ϕ *at position* $p = c_1 \dots c_n$ *is defined recursively. If* $n = 0$*, then* $\phi|_p = \phi$*. If* $n = 1$ *(i.e.* $p = c_1$*), then* $p_2 = []$*, otherwise (i.e.* $n > 1$*,* $p = c_1 c_2 \dots c_n$*),* $p_2 = c_2 \dots c_n$*.*

- *If* $n > 0$*,* $a_j|_p$ *and* $l_i|_p$ *are undefined.*
- *Let* $\phi = \wedge\{\psi_1, \dots, \psi_n\}$ *or* $\phi = \vee\{\psi_1, \dots, \psi_n\}$*. If* $0 < c_1 \leq n$*,* $\phi|_p = \psi_{c_1}|_{p_2}$*; if* $c_1 > n$*,* $\phi|_p$ *is undefined.*

The partial indexing function $(\!|\_|\!) : Seq + Prem \to Pos$ *maps sequents* $\Gamma \in Seq$ *or premises* $\Sigma \in Prem$ *from the defined subset of its domain to a position.*

We also introduce sets of provable positions $A_p \subseteq Pos$ *and non-provable positions* $E_p \subseteq Pos$ *as well as the set tcs of positions that have been expanded since the last propagation step and that are relevant to the propagation.*

When constructing the proof graph g for a formula ϕ, we build up a proof formula $pf(g)$ in parallel, beginning with $pf(g_0) = a_{(\!|\{\phi\}|\!)}$, $(\!|\_|\!)$ undefined except for $(\!|\{\phi\}|\!) = []$ and $A_p = E_p = tcs = \emptyset$.

Expanding the proof formula: Say we expand sequent Γ, s.t. $g \xrightarrow{\Gamma}_E g'$ for $g' = exp(g, \Gamma) = (A', E', U', X', L'_1, L'_2)$. This means that there are n premises to Γ (i.e. $|AP(\Gamma)| = n$) and for each $\Sigma_i \in AP(\Gamma)$, $(\Gamma, \Sigma_i) \in L'_1$. Also, for each $\Sigma_i \in AP(\Gamma)$, there is an m_i s.t. $|\{\Gamma_j \mid (\Sigma_i, \Gamma_j) \in AS(\Gamma)\}| = m_i$ and each such (Σ, Γ_j) is contained in L'_2. In other words, Γ has n premises, each premise consists of m_i sequents and the according transitions are stored in L'_1 and L'_2.

The according changes to the proof formula and the indexing function are as follows:
First we expand the sequent Γ:

- Replace $a_{(\!|\Gamma|\!)}$ with $\vee\{b_1, \dots, b_n\}$.
- For $0 < i \leq n$, if $(\!|\Sigma_i|\!)$ is defined (i.e. the premise was encountered before), replace b_i by $l_{(\!|\Sigma_i|\!)}$. If $(\!|\Sigma_i|\!) \in A_p \cup E_p$, add $(\!|\Gamma|\!)$ to tcs.
- For each remaining b_i (i.e. for each new premise), set $(\!|\Sigma_i|\!) = (\!|\Gamma|\!)i$.
- *Optional:* If $n = 1$, then $\vee\{b_1\} = b_1$ and $(\!|\Sigma_1|\!) = (\!|\Gamma|\!)$.
- If $n = 0$, then add $(\!|\Gamma|\!)$ to tcs.

Then we expand all the premises Σ_i:

– Replace any b_i with $\wedge\{a_1, \ldots, a_{m_i}\}$.
– For $0 < j \leq m_i$, if $(\!|\Gamma_j|\!)$ is defined (i.e. the sequent was encountered before), replace a_j by $l_{(\!|\Gamma_j|\!)}$. If $(\!|\Gamma_j|\!) \in A_p \cup E_p$, add $(\!|\Sigma_i|\!)$ to tcs.
– For each remaining a_j (i.e. for each new sequent), set $(\!|\Gamma_j|\!) = (\!|\Sigma_i|\!)j$.
– *Optional:* If $m_i = 1$, then $\wedge\{a_1\} = a_1$ and $(\!|\Gamma_1|\!) = (\!|\Sigma_i|\!)$.
– If $m_i = 0$, then add $(\!|\Sigma_i|\!)$ to tcs.
– Replace any a_j with $a_{(\!|\Gamma_j|\!)}$.

Propagation in the proof formula: Propagation of A (E) through the proof formula is realized by iterative approximation of the fixpoints. Any of the rules from Figure 11 is applied repeatedly for each $pj \in tcs$ until no rule may be applied for this specific pj any more, pj is removed from tcs afterwards.

Condition(s)	Changes to A_p (E_p)	Changes to tcs
$\phi\|_{pj} = \wedge\emptyset$	add pj	
$\phi\|_{pj} = \vee\emptyset$	(add pj)	
$\exists q \in Pos.\phi\|_q = l_{pj}$, $pj \in A_p$	add q	add q
$\exists q \in Pos.\phi\|_q = l_{pj}$, $pj \in E_p$	(add q)	add q
$\phi\|_p = \wedge\{\psi_1, \ldots, \psi_n\}$, $\forall i \in \{1 \ldots n\}.pi \in A_p$	add p	add p
$\phi\|_p = \vee\{\psi_1, \ldots, \psi_n\}$, $\exists i \in \{1 \ldots n\}.pi \in A_p$	add p	add p
$\phi\|_p = \wedge\{\psi_1, \ldots, \psi_n\}$, $\exists i \in \{1 \ldots n\}.pi \in E_p$	(add p)	add p
$\phi\|_p = \vee\{\psi_1, \ldots, \psi_n\}$, $\forall i \in \{1 \ldots n\}.pi \in E_p$	(add p)	add p

Fig. 11. The simplification rules for propagation in proof formulas

Propagation adds the positions of all newly added empty conjunctions (and disjunctions) to A_p (E_p) and propagates them through the formula. The set tcs is empty after each completed propagation step.

The set of all atoms a_j that appear in $pf(g)$ is the relevant subset of the set X of unexpanded sequents of the proof graph.

Lemma 6. $\{\Gamma \in Seq \mid (\!|\Gamma|\!) \in A_p\} = A$.

Proof. The constructed proof formula is a direct representation of the proof graph. Repeated application of the rules from Figure 11 iteratively approximates the collection of fixpoints A and stores the according positions in A_p.

It follows that ϕ is provable iff $[] \in A_p$ and that ϕ is not provable otherwise. The improved efficiency of the process of propagating in the proof formula turns propagation after each expansion step into a feasible option.

5 Conclusion

We have applied the well known optimisation techniques of semantic branching and dependency directed backtracking to coalgebraic logics and we have shown

that the techniques generalize to all coalgebraic modal logics in a natural way. This is especially interesting in the case of those modal logics whose modal rule introduces additional branching (in contrast to more classic modal logics which only branch on conjunctions).

As a second contribution, we have established a minimal representation of the structure of satisfiability proofs of coalgebraic modal logics; this is achieved by introducing and justifying the usage of proof formulas as a representation of the proof graph. Hence basic propagation becomes a mere process of simplification of the respective proof formula starting at the successful or unsucessful leaves and only traversing the needed parts of the proof structure.

The presented algorithm and the proposed optimisations have been successfully implemented as a part of the experimental prover *CoLoSS*. Additionally, the following optimization techniques have been implemented:

- restriction to maximal application of modal rules (when admissable),
- irrelevance check for subformulas of polarity preserving modal logics,
- heuristics (such as *"most-appearing-first"* or *"oldest-first"*).

The implementation shows reasonable performance especially for logics for which few optimisations apart from those presented here exist. However, detailled and relevant benchmarking of the examplary implementation in *CoLoSS* is a complex task (due to the amount and nature of different logics and optimisation techniques which are involved) and subject to ongoing research.

References

1. G. Calin, R. Myers, D. Pattinson, and L. Schröder. Coloss: The coalgebraic logic satisfiability solver (system description). In *Methods for Modalities, M4M-5*, vol. 231 of *ENTCS*, pp. 41–54. Elsevier, 2009.
2. R. Goré, C. Kupke, and D. Pattinson. Optimal tableau algorithms for coalgebraic logics. In R. Majumdar and J. Esparza, eds., *Proc. TACAS 2010*, Lecture Notes in Comp. Sci., 2010.
3. D. Hausmann and L. Schröder. Optimizing conditional logic reasoning within coloss (system description). In *Methods for Modalities, M4M-6*, vol. 262 of *ENTCS*, pp. 157–171. Elsevier, 2010.
4. I. Horrocks and P. Patel-Schneider. Optimizing description logic subsumption. *Journal of Logic and Computation*, 9(3):267–293, 1999.
5. D. Pattinson and L. Schröder. Admissibility of cut in coalgebraic logics. In *Coalgebraic Methods in Computer Science, CMCS 08*, vol. 203 of *ENTCS*, pp. 221–241. Elsevier, 2008.
6. L. Schröder and D. Pattinson. Pspace bounds for rank-1 modal logics. *ACM Trans. Comput. Logic*, 10(2:13):1–33, 2009.