Optimizing Conditional Logic Reasoning within CoLoSS

Daniel Hausmann[a,1] Lutz Schröder[a,b,2]

[a] *DFKI Bremen, SKS*

[b] *Department of Mathematics and Computer Science, Universität Bremen, Germany*

Abstract

The generic modal reasoner CoLoSS covers a wide variety of logics ranging from graded and probabilistic modal logic to coalition logic and conditional logics, being based on a broadly applicable coalgebraic semantics and an ensuing general treatment of modal sequent and tableau calculi. Here, we present research into optimisation of the reasoning strategies employed in CoLoSS. Specifically, we discuss strategies of memoisation and dynamic programming that are based on the observation that short sequents play a central role in many of the logics under study. These optimisations seem to be particularly useful for the case of conditional logics, for some of which dynamic programming even improves the theoretical complexity of the algorithm. These strategies have been implemented in CoLoSS; we give a detailed comparison of the different heuristics, observing that in the targeted domain of conditional logics, a substantial speed-up can be achieved.

Keywords: Coalgebraic modal logic, conditional logic, automated reasoning, optimisation, heuristics, memoizing, dynamic programming

1 Introduction

In recent decades, modal logic has seen a development towards semantic heterogeneity, witnessed by an emergence of numerous logics that, while still of manifestly modal character, are not amenable to standard Kripke semantics. Examples include probabilistic modal logic [4], coalition logic [11], and conditional logic [2], to name just a few. The move beyond Kripke semantics, mirrored on the syntactical side by the failure of normality, entails additional challenges for tableau and sequent systems, as the correspondence between tableaus and models becomes looser, and in particular demands created by modal formulas can no longer be discharged by the creation of single successor nodes.

This problem is tackled on the theoretical side by introducing the semantic framework of coalgebraic modal logic [8,12], which covers all logics mentioned above and many more. It turns out that coalgebraic modal logic does allow the design

[1] Email: Daniel.Hausmann@dfki.de

[2] Email: Lutz.Schroeder@dfki.de

of generic reasoning algorithms, including a generic tableau method originating from [15]; this generic method may in fact be separated from the semantics and developed purely syntactically, as carried out in [9,10].

Generic tableau algorithms for coalgebraic modal logics, in particular the algorithm described in [15], have been implemented in the reasoning tool CoLoSS [1] [3]. As indicated above, it is a necessary feature of the generic tableau systems that they potentially generate multiple successor nodes for a given modal demand, so that in addition to the typical depth problem, proof search faces a rather noticeable problem of breadth. The search for optimisation strategies to increase the efficiency of reasoning thus becomes all the more urgent. Here we present one such strategy, which is generally applicable, but particularly efficient in reducing both depth and branching for the class of conditional logics. We exploit a notable feature of this class, namely that many of the relevant rules rely rather heavily on premises stating equivalence between formulas; thus, conditional logics are a good candidate for memoising strategies, applied judiciously to short sequents. We describe the implementation of memoising and dynamic programming strategies within CoLoSS, and discuss the outcome of various comparative experiments.

2 Generic Sequent Calculi for Coalgebraic Modal Logic

Coalgebraic modal logic, originally introduced as a specification language for coalgebras, seen as generic reactive systems [8], has since evolved into a generic framework for modal logic beyond Kripke semantics [3]. The basic idea is to encapsulate the branching type of the systems relevant for the semantics of a particular modal logic, say probabilistic or game-theoretic branching, in the choice of a set functor, the signature functor (e.g. the distribution functor and the games functor in the mentioned examples), and to capture the semantics of modal operators in terms of so-called predicate liftings. For the purposes of the present work, details of the semantics are less relevant than proof-theoretic aspects, which we shall recall presently. The range of logics covered by the coalgebraic approach is extremely broad, including, besides standard Kripke and neighbourhood semantics, e.g. graded modal logic [5], probabilistic modal logic [4], coalition logic [11], various conditional logics equipped with selection function semantics [2], and many more.

Syntactically, logics are parametrised by the choice of a *modal similarity type* Λ, i.e. a set of modal operators with associated finite arities. This choice determines the set of formulas ϕ, ψ via the grammar

$$\phi, \psi ::= p \mid \phi \wedge \psi \mid \neg\phi \mid \heartsuit(\phi_1, \dots, \phi_n)$$

where $\heartsuit$ is an n-ary operator in Λ. Examples are $\Lambda = \{L_p \mid p \in [0,1] \cap \mathbb{Q}\}$, the unary operators L_p of probabilistic modal logic read 'with probability at least p'; $\Lambda = \{\Diamond_k \mid k \in \mathbb{N}\}$, the operators $\Diamond_k$ of graded modal logic read 'in more than k successors'; $\Lambda = \{[C] \mid C \subseteq N\}$, the operators $[C]$ of coalition logic read 'coalition C (a subset of the set N of agents) can jointly enforce that'; and, for our main

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

example here, $\Lambda = \{\Rightarrow\}$, the *binary* modal operator $\Rightarrow$ of conditional logic read e.g. 'if ... then normally ...'.

Coalgebraic modal logic was originally limited to so-called *rank*-1 *logics* axiomatised by formulas with nesting depth of modal operators uniformly equal to 1 [12]. It has since been extended to the more general non-iterative logics [14] and to some degree to iterative logics, axiomatised by formulas with nested modalities [13]. The examples considered here all happen to be rank-1, so we focus on this case. In the rank-1 setting, it has been shown [12] that all logics can be axiomatised by *one-step rules* ϕ/ψ, where ϕ is purely propositional and ψ is a clause over formulas of the form $\heartsuit(a_1, \dots, a_n)$, where the a_i are propositional variables. In the context of a sequent calculus, this takes the following form [9].

Definition 2.1 If S is a set (of formulas or variables) then $\Lambda(S)$ denotes the set $\{\heartsuit(s_1, \dots, s_n) \mid \heartsuit \in \Lambda \text{ is } n\text{-ary}, s_1, \dots, s_n \in S\}$ of formulas comprising exactly one application of a modality to elements of S. An *S-sequent*, or just a *sequent* in case that S is the set of all formulas, is a finite subset of $S \cup \{\neg A \mid A \in S\}$. Then, a *one-step rule* $\Gamma_1, \dots, \Gamma_n/\Gamma_0$ over a set V of variables consists of V-sequents $\Gamma_1, \dots, \Gamma_n$, the *premises*, and a $\Lambda(S)$-sequent Γ_0, the *conclusion*. A *goal* is a set of sequents, typically arising as the set of instantiated premises of a rule application.

A given set of one-step rules then induces an instantiation of the *generic sequent calculus* [9] which is given by a set of rules $\mathcal{R}_{sc}$ consisting of the finishing and the branching rules $\mathcal{R}^b_{sc}$ (i.e. rules with no premise or more than one premise), the linear rules $\mathcal{R}^l_{sc}$ (i.e. rules with exactly one premise) and the modal rules $\mathcal{R}^m_{sc}$, i.e. the given one-step rules. The finishing and the branching rules are presented in Figure 1 (where $\top = \neg\bot$ and p is an atom), the linear rules are shown in Figure 2. So far, all these rules are purely propositional. As an example for a set of modal one-step rules, consider the modal rules $\mathcal{R}^m_{sc}$ of the standard modal logic **K** as given by Figure 3.

$$(\neg F)\ \frac{}{\Gamma, \neg\bot} \qquad (Ax)\ \frac{}{\Gamma, p, \neg p} \qquad (\wedge)\ \frac{\Gamma, A \qquad \Gamma, B}{\Gamma, A \wedge B}$$

Fig. 1. The finishing and the branching sequent rules $\mathcal{R}^b_{sc}$

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

Fig. 2. The linear sequent rules $\mathcal{R}^l_{sc}$

This calculus has been shown to be complete under suitable coherence assumptions on the rule set and the coalgebraic semantics, provided that the set of rules *absorbs cut and contraction* in a precise sense [9]. We say that a formula is *provable* if

$$(\mathbf{K})\,\frac{\neg A_1, \dots, \neg A_n, A_0}{\Gamma, \neg\Box A_1, \dots, \neg\Box A_n, \Box A_0}$$

Fig. 3. The modal rule of **K**

it can be derived in the relevant instance of the sequent calculus; the algorithms presented below are concerned with the *provability problem*, i.e. to decide whether a given sequent is provable. This is made possible by the fact that the calculus does not include a cut rule, and hence enables automatic proof search. For rank-1 logics, proof search can be performed in *PSPACE* under suitable assumptions on the representation of the rule set [15,9]. The corresponding algorithm has been implemented in the system CoLoSS [1] which remains under continuous development. Particular attention is being paid to finding heuristic optimisations to enable practically efficient proof search, as described here.

Our running example used in the presentation of our optimisation strategies below is conditional logic as mentioned above. The most basic conditional logic is **CK** [2] (we shall consider a slightly extended logic below), which is characterised by assuming normality of the right-hand argument of the non-monotonic conditional $\Rightarrow$, but only replacement of equivalents in the left-hand arguments. Absorption of cut and contraction requires a unified rule set consisting of the rules depicted in Fig. 4 with $A = C$ abbreviating the pair of sequents $\neg A, C$ and $\neg C, A$. This illus-

$$(\mathbf{CK})\,\frac{A_0 = A_1; \dots; A_n = A_0 \qquad \neg B_1, \dots, \neg B_n, B_0}{\Gamma, \neg(A_1 \Rightarrow B_1), \dots, \neg(A_n \Rightarrow B_n), (A_0 \Rightarrow B_0)}$$

Fig. 4. The modal rule of **CK**

trates two important points that play a role in the optimisation strategies described below. First, the rule has a large branching degree both existentially and universally – we have to check that there *exists* a rule proving some sequent such that *all* its premises are provable, and hence we have to guess one of exponentially many subsequents to match the rule and then attempt to prove linearly many premises; compare this to linearly many rules and a constant number of premises per rule (namely, 1) in the case of **K** shown above. Secondly, many of the premises are short sequents. This will be exploited in our memoisation strategy. We note that *labelled* sequent calculi for conditional logics have been designed previously [7] and have been implemented in the CondLean prover [6]; contrastingly, our calculus is unlabelled, and turns out to be conceptually simpler. The comparison of the relative efficiency of labelled and unlabelled calculi remains an open issue for the time being.

3 The Algorithm

According to the framework introduced above, we now devise a generic algorithm to decide the provability of formulas, which is easily instantiated to a specific logic by just implementing the relevant modal rules.

Algorithm 1 makes use of the sequent rules in the following manner: In order to show the *provability* of a formula ϕ, the algorithm starts with the sequent $\{\phi\}$ and just keeps trying to apply all of the sequent rules in $\mathcal{R}_{sc}$ to it, giving precedence to the linear rules. Below, we refer to a sequent as *open* if it has not yet been checked for provability. Under suitable tractability assumptions on the rule set as in [15,9], this algorithm realises the theoretical upper bound *PSPACE*. It is the starting point of the proof search algorithms employed in CoLoSS, being essentially a sequent reformulation of the algorithm described in [1], where it is easily verified that correctness and termination of the algorithm are preserved by this reformulation; optimisations of this algorithm are the subject of the present work.

Algorithm 1 *(Decide provability of a sequent Γ)*

1. Try all possible applications of rules from $\mathcal{R}_{sc}$ to Γ, giving precedence to linear rules. For every such rule application, perform the following steps, and answer 'provable' in case these steps succeed for one of the rule applications.
2. Let Λ denote the set of premises arising from the rule application.
3. Check recursively that every sequent in Λ is provable.

3.1 The conditional logic instance

The genericity of the introduced sequent calculus allows us to easiliy create instantiations of Algorithm 1 for a large variety of modal logics:
For instance it has been shown in [10] that the complexity of **CKCEM** is *coNP*, using a dynamic programming approach in the spirit of [16]; in fact, this was the original motivation for exploring the optimisation strategies pursued here. Due to this reason, we restrict ourselves to the examplary conditional logic **CKCEM** for the remainder of this section; slightly adapted versions of the optimisation will work for other conditional logics. **CKCEM** is characterised by the additional axioms of *conditional excluded middle* $(A \Rightarrow B) \vee (A \Rightarrow \neg B)$, which to absorb cut and contraction is integrated in the rule for **CK** as shown in Figure 5.

$$(\mathbf{CKCEM})\frac{A_0 = A_1; \dots ; A_n = A_0 \qquad B_0, \dots , B_j, \neg B_{j+1}, \dots , \neg B_n}{\Gamma, (A_0 \Rightarrow B_0), \dots , (A_j \Rightarrow B_j), \neg(A_{j+1} \Rightarrow B_{j+1}), \dots , \neg(A_n \Rightarrow B_n)}$$

Fig. 5. The modal rule **CKCEM** of conditional logic

In the following, we use the notions of *conditional antecedent* and *conditional consequent* to refer to the parameters of the modal operator of conditional logic.

In order to decide using Algorithm 1 whether there is an instance of the modal rule of conditional logic which can be applied to the actual current sequent, it is necessary to create a preliminary premise for each possible combination of equalities of all the premises of the modal operators in this sequent. This results in $2^n - 1$ new premises for a sequent with n top-level modal operators.

Example 3.1 For the sequent $\Gamma = \{(A_0 \Rightarrow B_0), (A_1 \Rightarrow B_1), (A_2 \Rightarrow B_2)\}$, there are $2^3 = 8$ possible instances of the rule to be tried, corresponding to the non-empty subsequents of the goal; the premise to be checked for $I \subseteq \{1, 2, 3\}$ consists of $A_i = A_j$ for $i, j \in I$, and $\{B_i \mid i \in I\}$.

It seems to be a more intelligent approach to first partition the set of all antecedents of the top-level modal operators in the current sequent into equivalence classes with respect to logical equality. This partition allows for a significant reduction both of the number of rules to be tried and of the number of premises to be actually proved for each rule.

Example 3.2 Consider again the sequent from Example 3.1. By using the examplary knowledge that $A_0 = A_1$, $A_1 \neq A_2$ and $A_0 \neq A_2$, it is immediate that there are just two reasonable instations of the modal rule, leading to the two premises $\{\{B_0, B_1\}\}$ and $\{\{B_2\}\}$. For the first of these two premises, note that it is not necessary to show the equivalence of A_0 and A_1 again.

In the case of conditional logic, observe the following: Since the modal antecedents that appear in a formula are not being changed by any rule of the sequent calculus, it is possible to extract all possibly relevant antecedents of a formula even *before* the actual sequent calculus is applied. This allows us to first compute the equivalence classes of all the relevant antecedents and then feed this knowledge into the actual sequent calculus, as illustrated next.

4 The Optimisation

Definition 4.1 A conditional antecedent of *modal nesting depth* i is a conditional antecedent which contains at least one antecedent of modal nesting depth $i-1$ and which contains no antecedent of modal nesting depth greater than $i-1$. A conditional antecedent of nesting depth 0 is an antecedent that does not contain any further modal operators. Let a_i denote the set of all conditional antecedents of modal nesting depth i. Further, let $\mathit{prems}(n)$ denote the set of all conditional antecedents of modal nesting depth at most n (i.e. $\mathit{prems}(n) = \bigcup_{j=1..n} a_j$). Finally, let $\mathit{depth}(\phi)$ denote the maximal modal nesting in the formula ϕ.

Definition 4.2 A set $\mathcal{K}$ of sequents together with a function $\mathit{eval} : \mathcal{K} \to \{\top, \bot\}$ is called a *knowledge base*.

We may now construct an optimized algorithm which allows us to decide provability (and satisfiability) of formulas more efficiently in some cases. The optimized algorithm is constructed from two functions (namely from the actual proving function and from the so-called *pre-proving* function):

Algorithm 2 *(Decide provability of ϕ using the knowledge base $(\mathcal{K}, \mathit{eval})$)*

1. If $\Gamma \in \mathcal{K}$, answer 'provable' if $\mathit{eval}(\Gamma) = \top$, else 'unprovable'. Otherwise:

2. Try all possible applications of rules from $\mathcal{RO}_{sc}$ to Γ, giving precedence to linear rules, where $\mathcal{RO}_{sc}$ is an optimised set of rules taking into account the knowledge base, as explained below. For every such rule application, perform the following steps, and answer 'provable' in case these steps succeed for one of the rule applications.
3. Let Λ denote the set of premises arising from the rule application.
4. Check recursively that every sequent in Λ is provable.

Algorithm 2 is very similar to Algorithm 1 but relies on the knowledge base passed to it and moreover uses a modified set of rules $\mathcal{RO}_{sc}$. The set of rules $\mathcal{RO}_{sc}$ makes appropriate use of the knowledge base. It is obtained from $\mathcal{R}_{sc}$ by replacing the modal rule from Figure 5 with the modified modal rule from Figure 6. The point is that the premises $A_0 = \cdots = A_n$ are replaced by side conditions representing lookup in the knowledge base. This improves on standard memoising as embodied by the lookup operation in step 1 of the above algorithm in that existential branching over potentially applicable rules is reduced: the rule does not even match the target sequent unless the equivalence premises are already in the knowledge base. This is still a complete system due to the way memoising is organised, as explained below.

$$(\mathbf{CKCEM}^m)\frac{B_0, \ldots, B_j, \neg B_{j+1}, \ldots, \neg B_n}{\Gamma, (A_0 \Rightarrow B_0), \ldots, (A_j \Rightarrow B_j), \neg(A_{j+1} \Rightarrow B_{j+1}), \ldots, \neg(A_n \Rightarrow B_n)}$$

$$(\textstyle\bigwedge_{i,j\in\{1..n\}} eval(A_i = A_j) = \top)$$

Fig. 6. The modified modal rule $\mathbf{CKCEM}^m$ of conditional logic

The knowledge base used in Algorithm 2 is computed in Algorithm 3. The algorithm proceeds by dynamic programming with stages corresponding to modal nesting depth, in the spirit of [16]. Thus, in order to show the equivalence of two conditional antecedents of nesting depth at most i, we assume that the equivalences $\mathcal{K}_i$ between modal antecedents of nesting depth less than i have already been computed and the result is stored in $eval_i$; hence, two antecedents are equal, if their equivalence is provable by Algorithm 2 using only the knowledge base $(\mathcal{K}_i, eval_i)$.

Algorithm 3 Step 1: Take a formula ϕ as input. Set $i = 0$, $\mathcal{K}_0 = \emptyset$, $eval_0 = \emptyset$.
Step 2: Generate the set $prems_i$ of all conditional antecedents of ϕ of nesting depth at most i. If $i < depth(\phi)$ continue with Step 3, else set $\mathcal{K} = \mathcal{K}_{i-1}$, $eval = eval_{i-1}$ and continue with Step 4.
Step 3: Let eq_i denote the set of all equalities $A_a = A_b$ for different formulas $A_a, A_b \in prems_i$. Compute Algorithm 2 $(\psi, (\mathcal{K}_i, eval_i))$ for all $\psi \in eq_i$. Set $\mathcal{K}_{i+1} = eq_i$, set $i = i + 1$. For each equality $\psi \in eq_i$, set $eval_{i+1}(\psi) = \top$ if the result of Algorithm 2 was 'provable' and $eval_{i+1}(\psi) = \bot$ otherwise. Continue with Step 2.
Step 4: Call Algorithm 2 $(\phi, (\mathcal{K}, eval))$ and return its return value as result.

4.1 Treatment of Requisite Equivalences Only

Since Algorithm 3 tries to show the logical equivalence of any combination of two conditional antecedents that appear in ϕ, it will have worse completion time than Algorithm 1 on many formulas:

Example 4.3 Consider the formula

$$\phi = (((p_0 \Rightarrow p_1) \Rightarrow p_2) \Rightarrow p_4) \vee (((p_5 \Rightarrow p_6) \Rightarrow p_7) \Rightarrow p_8).$$

Algorithm 3 will not only try to show the necessary equivalences between the pairs $(((p_0 \Rightarrow p_1) \Rightarrow p_2), ((p_5 \Rightarrow p_6) \Rightarrow p_7))$, $((p_0 \Rightarrow p_1), (p_5 \Rightarrow p_6))$ and (p_0, p_5), but it will also try to show equivalences between any two conditional antecedents (e.g. $(p_0, (p_5 \Rightarrow p_6)))$, even though these equivalences will not be needed during the execution of Algorithm 2.

Based on this observation it is possible to assign a category to each pair of antecedents that appear in it:

Definition 4.4 The *paths (in ϕ)* to a conditional antecedent ψ describe the orders of modal arguments through which ψ is reached, when starting from the root of ϕ: The path to a top-level antecedent is just $\{1\}$. If ψ does not appear as antecedent on the topmost level of ϕ, the path to it is $\{1\}$ prepended to the set of paths to ψ in any top-level conditional antecedent of ϕ together with $\{0\}$ prepended to the set of paths to ψ in any top-level conditional consequent of ϕ.

Example 4.5 Consider the formula $\phi = (p_0 \Rightarrow p_2) \Rightarrow ((p_0 \Rightarrow p_1) \Rightarrow p_3)$. Then the path to $(p_0 \Rightarrow p_2)$ is $\{1\}$, whereas the path to $(p_0 \Rightarrow p_1)$ is $\{01\}$. The paths to p_0 are $\{11, 011\}$.

Definition 4.6 Let A and B be two conditional antecedents. A and B are called *connected (in ϕ)* if at least one path to A is also a path to B (and hence vice-versa). If no path to A is a path to B, the two antecedents are said to be *independent*.

Since two independent conditional antecedents will never appear in the scope of the same application of the modal rule, it is in no case necessary to show (or refute) the logical equivalence of independent conditional antecedents. Hence it suffices to focus our attention to the connected conditional antecedents. It is then obvious that any possibly requisite equivalence and its truth-value are allready included in (K, eval) when the main proving is induced. On the other hand, we have to be aware that it may be the case, that we show equivalences of antecedents which are in fact not needed (since antecedents may indeed be connected and still it is possible that they never appear together in an application of the modal rule - this is the case whenever two preceding antecedents are not logically equivalent).

As result of these considerations, we devise Algorithm 4, an improved version of Algorithm 3. The only difference is that before proving any equivalence, Algorithm 4 checks whether the current pair of conditional antecedents is actually connected; only then does it treat the equivalence. Hence independent pairs of antecedents remain untreated

Algorithm 4 Step 1: Take a formula ϕ as input. Set $i = 0$, $\mathcal{K}_0 = \emptyset$, $\mathit{eval}_0 = \emptyset$.
Step 2: Generate the set prems_i of all conditional antecedents of ϕ of nesting depth

at most i. If $i < depth(\phi)$ continue with Step 3, else set $\mathcal{K} = \mathcal{K}_{i-1}$, $eval = eval_{i-1}$ and continue with Step 4.
Step 3: Let eq_i denote the set of all equalities $A_a = A_b$ for different and not independent pairs of formulas $A_a, A_b \in prems_i$. Compute Algorithm 2 (ψ, ($\mathcal{K}_i, eval_i$)) for all $\psi \in eq_i$. Set $\mathcal{K}_{i+1} = eq_i$, set $i = i + 1$. For each equality $\psi \in eq_i$, set $eval_{i+1}(\psi) = \top$ if the result of Algorithm 2 was 'provable' and $eval_{i+1}(\psi) = \bot$ otherwise. Continue with Step 2.
Step 4: Call Algorithm 2 (ϕ, ($\mathcal{K}, eval$)) and return its return value as result.

5 Implementation

The proposed optimized algorithms have been implemented (using the programming language Haskell) as part of the generic coalgebraic modal logic satisfiability solver (CoLoSS [4]). CoLoSS provides the general coalgebraic framework in which the generic sequent calculus is embedded. It is easily possible to instantiate this generic sequent calculus to specific modal logics, one particular example being conditional logic. The matching function for conditional logic in CoLoSS was hence adapted in order to realize the different optimisations (closely following Algorithms 1, 3 and 4), so that CoLoSS now provides an efficient algorithm for deciding the provability (and satisfiability) of conditional logic formulas.

5.1 Comparing the Proposed Algorithms

In order to show the relevance of the proposed optimisations, we devise several classes of conditional formulas. Each class has a characteristic general shape, defining its complexity w.r.t. different parts of the algorithms and thus exhibiting specific advantages or disadvantages of each algorithm:

- The formula `bloat`(i) is a full binary tree of depth i (containing 2^i pairwise logically inequivalent atoms and $2^i - 1$ independent modal antecedents):
  $\texttt{bloat}(i) = (\texttt{bloat}(i-1)) \Rightarrow (\texttt{bloat}(i-1))$
  $\texttt{bloat}(0) = p_{rand}$
  Formulas from this class should show the problematic performance of Algorithm 3 whenever a formula contains many modal antecedents which appear at different depths. A comparison of the different algorithms w.r.t. formulas `bloat`(i) is depicted in Figure 7. Since Algorithm 3 does not check whether pairs of modal antecedents are independent or connected, it performs considerably worse than Algorithm 4 which only attempts to prove the logical equivalence of formulas which are not independent. Algorithm 1 has the best performance in this extreme case, as it only has to consider pairs of modal antecedents which actually appear during the course of a proof. This is the price to pay for the optimisation by dynamic programming.

- The formula `conjunct`(i) is just an i-fold conjunction of a specific formula A:

[4] As already mentioned above, more information about CoLoSS, a web-interface to the tool and the tested benchmarking formulas can be found at `http://www.informatik.uni-bremen.de/cofi/CoLoSS/`

i	Algorithm 1	Algorithm 3	Algorithm 4
1	0.007s	0.007s	0.007s
2	0.007s	0.007s	0.007s
3	0.007s	0.008s	0.008s
4	0.008s	0.017s	0.012s
5	0.009s	0.112s	0.048s
6	0.010s	1.154s	0.416s
7	0.012s	11.998s	4.116s

Fig. 7. Results for bloat(i)

`conjunct(`i`)` $= A_1 \wedge \ldots \wedge A_i$
$A = (((p_1 \vee p_0) \Rightarrow p_2) \vee ((p_0 \vee p_1) \Rightarrow p_2)) \vee \neg(((p_0 \vee p_1) \Rightarrow p_2) \vee ((p_1 \vee p_0) \Rightarrow p_2)))$
This class consists of formulas which contain logically (but not sytactically) equivalent antecedents. As i increases, so does the amount of appearances of identical modal antecedents in different positions of the considered formula. A comparison of the different algorithms w.r.t. formulas `conjunct(`i`)` is depicted in Figure 8. It is obvious that the optimized algorithms perform considerably better than the unoptimized Algorithm 1. The reason for this is that Algorithm 1 repeatedly proves equivalences between the same pairs of modal antecedents. The optimized algorithms on the other hand are equipped with knowledge about the modal antecedents, so that these equivalences have to be proved only once. However, even the runtime of the optimized algorithms is exponential in i, due to the exponentially increasing complexity of the underlying propositional formula. Note that the use of propositional tautologies (such as $A \leftrightarrow (A \wedge A)$ in this case) would help to greatly reduce the computing time for `conjunct(`i`)`. Optimisation of propositional reasoning is not the scope of this paper though, thus we devise the following examplary class of formulas (for which propositional tautologies would not help):

- The formula `explode(`i`)` contains equivalent but not syntactically equal and interchangingly nested modal antecedents of depth at most i:
  `explode(`i`)` $= X_1^i \vee \ldots \vee X_i^i$
  $X_1^i = (A_1^i \Rightarrow (\ldots(A_i^i \Rightarrow (c_1 \wedge \ldots \wedge c_i))\ldots))$
  $X_j^i = (A_{j \bmod i}^i \Rightarrow (\ldots(A_{(j+(i-1)) \bmod i}^i \Rightarrow \neg c_j)\ldots))$
  $A_j^i = p_{j \bmod i} \wedge \ldots \wedge p_{(j+(i-1)) \bmod i}$
  This class contains complex formulas for which the unoptimized algorithm should not be efficient any more: Only the combined knowledge about all appearing modal antecedents A_j^i allows the proving algorithm to reach all modal consequents c_n, and only the combined sequent $\{(c_1 \wedge \ldots \wedge c_i), \neg c_1, \ldots, \neg c_i\}$ (containing every appearing consequent) is provable. For formulas from this class (specifically designed to show the advantages of optimization by dynamic programming), the optimized algorithms vastly outperform the unoptimized algorithm (see Figure 9).

i	Algorithm 1	Algorithm 3	Algorithm 4
1	0.008s	0.008s	0.008s
2	0.009s	0.009s	0.009s
3	0.011s	0.010s	0.010s
4	0.058s	0.011s	0.011s
5	1.335s	0.014s	0.015s
6	42.885s	0.038s	0.040s
7	>600.000s	0.220s	0.232s
8	$\gg$600.000s	1.944s	2.019s
9	$\gg$600.000s	17.826s	18.790s

Fig. 8. Results for conjunct(i)

i	Algorithm 1	Algorithm 3	Algorithm 4
1	0.007s	0.007s	0.007s
2	0.007s	0.007s	0.007s
3	0.009s	0.009s	0.009s
4	0.017s	0.010s	0.010s
5	0.133s	0.012s	0.012s
6	0.305s	0.015s	0.016s
7	430.684s	0.018s	0.022s
8	$\gg$600.000s	0.023s	0.029s
9	$\gg$600.000s	0.029s	0.044s

Fig. 9. Results for explode(i)

The tests were conducted on a Linux PC (Dual Core AMD Opteron 2220S (2800MHZ), 16GB RAM). It is obvious that a significant increase of performance may be obtained through the proposed optimisations. In general, the performance of the implementation of the proposed algorithms in the generic reasoner CoLoSS is comparable to dedicated conditional logic provers such as CondLean; a direct comparison is presently made difficult by the fact that the benchmarking formulas used to evaluate CondLean are not listed explicitly in [6].

6 Generalized Optimisation

As previously mentioned, the demonstrated optimisation is not restricted to the case of conditional modal logics.

Definition 6.1 If Γ is a sequent, we denote the set of all arguments of top-level modalities from Γ by $arg(\Gamma)$. A *short sequent* is a sequent which consists of just one formula which itself is a propositional formula over a fixed maximal number of modal arguments from $arg(\Gamma)$. In the following, we fix the maximal number of modal arguments in short sequents to be 2.

The general method of the optimisation then takes the following form: Let $S_1, \ldots, S_n$ be short sequents and assume that there is a (w.r.t the considered modal logic) sound instance of the generic rule which is depicted in Figure 10 (where $\mathcal{S}$ is a set of any sequents).

$$(\mathbf{Opt})\ \frac{S_1 \qquad \ldots \qquad S_n \qquad \mathcal{S}}{\Gamma}$$

Fig. 10. The general rule-scheme to which the optimisation may be applied

Then we devise a final version (Algorithm 5) of the optimized algorithm: Instead of considering only equivalences of conditional antecedents for pre-proving, we now extend our attention to any short sequents over any modal arguments.

Algorithm 5 Step 1: Take a formula ϕ as input. Set $i = 0$, $\mathcal{K}_0 = \emptyset$, $eval_0 = \emptyset$.
Step 2: Generate the set $args_i$ of all modal arguments of ϕ which have nesting depth at most i. If $i < depth(\phi)$ continue with Step 3, else set $\mathcal{K} = \mathcal{K}_{i-1}$, $eval = eval_{i-1}$ and continue with Step 4.
Step 3: Let seq_i denote the set of all short sequents of form S_i (where S_i is a sequent from the premise of rule (**Opt**)) over at most two formulas $A_a, A_b \in args_i$. Compute Algorithm 2 (ψ, ($\mathcal{K}_i$, $eval_i$)) for all $\psi \in seq_i$. Set $\mathcal{K}_{i+1} = seq_i$, set $i = i + 1$. For each short sequent $\psi \in seq_i$, set $eval_{i+1}(\psi) = \top$ if the result of Algorithm 2 was 'provable' and $eval_{i+1}(\psi) = \bot$ otherwise. Continue with Step 2.
Step 4: Call Algorithm 2 (ϕ, ($\mathcal{K}$, $eval$)) and return its return value as result.

This new Algorithm 5 may then be used to decide provability of formulas, where the employed ruleset has to be extended by the generic modified rule given by Figure 11.

$$(\mathbf{Opt}^m)\ \frac{\mathcal{S}}{\Gamma}\qquad (\textstyle\bigwedge_{i\in\{1..n\}} eval(S_i) = \top)$$

Fig. 11. The general optimized rule

Example 6.2 The following two cases are instances of the generic optimisation:

(i) (Classical modal Logics / Neighbourhood Semantics) Let $\Gamma = \{\Box A = \Box B\}$, $n = 1$, $S_1 = \{A = B\}$ and $\mathcal{S} = \emptyset$. Algorithm 5 may be then applied whenever the following congruence rule is sound in the considered logic:

$$(\mathbf{Opt}_{Cong})\ \frac{A = B}{\Box A = \Box B}$$

The according modified version of this rule is as follows:

$$(\mathbf{Opt}^m_{Cong})\ \frac{}{\Box A = \Box B}$$

with the side-condition $eval(A = B) = \top$.

(ii) (Monotone modal logics) By setting $\Gamma = \{\Box A \to \Box B\}$, $n = 1$, $S_1 = \{A \to B\}$ and $\mathcal{S} = \emptyset$, we may instantiate the generic algorithm to the case of modal logics which are monotone w.r.t. their modal operator. So assume the following rule to be sound in the considered modal logic:

$$(\mathbf{Opt}_{Mon})\ \frac{A \to B}{\Box A \to \Box B}$$

The according modified version of this rule is as follows:

$$(\mathbf{Opt}^m_{Mon})\ \frac{}{\Box A \to \Box B}$$

with the side-condition $eval(A \to B) = \top$.

In the case that $(\mathbf{Opt}_{Mon})$ is the only modal rule in the considered logic (i.e. the case of plain monotone modal logic), all the prove-work which is connected to the modal operator is shifted to the pre-proving process. Especially matching with the modal rules $\mathcal{RO}^m_{sc}$ becomes a mere lookup of the value of *eval*. This means, that all calls of the sequent algorithm Algorithm 2 correspond in complexity just to ordinary sat-solving of propositional logic. Furthermore, Algorithm 2 will be called $|\phi|$ times. This observation may be generalized:

Remark 6.3 In the case that all modal rules of the considered logic are instances of the generic rule (**Opt**) with $P = \emptyset$ (as seen in Example 6.2), the optimisation does not only allow for a reduction of computing time, but it also allows us to effectively reduce the sequent calculus to a sat-solving algorithm. Furthermore, the optimized algorithm will always be as efficient as the original one in this case (since every occurence of short sequents over $arg(\Gamma)$ which accord to the current instantiation of the rule (**Opt**) will have to be shown or refuted even during the course of the original algorithm).

7 Conclusion

We presented (from a practical point of view) two optimisations for reasoning in conditional logic:

- The first optimisation makes use of the concept of dynamic programming in order to separate the two tasks that showing validity of formulas in conditional logic consists of: The first task of proving equivalences of antecedents and the second task of ordinary sequent proving. The use of dynamic programming substantially decreases the branching breadth of the resulting sequent calculus.
- The second proposed optimisation introduces a strategy to reduce the amount of pairs of antecedents whose equivalence has to be considered. This is achieved by distinguishing between connected and independent pairs of modal arguments.

When both optimisations are applied at the same time, a significant increase in performance of the sequent algorithm for conditional logic can be observed. This was shown in Section 5.1 by considering the results of benchmarking a Haskell implementation (which forms part of the generic proving tool CoLoSS) of the optimised algorithms.

It remains as an open question whether the gain in perfomance which is obtained by optimising the algorithm for conditional logic may be transferred to other logics by making use of the generic optimisation strategy as described in the last section.

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] B. Chellas. *Modal Logic*. Cambridge University Press, 1980.

[3] C. Cirstea, A. Kurz, D. Pattinson, L. Schröder, and Y. Venema. Modal logics are coalgebraic. *The Computer Journal*, 2009. In print.

[4] R. Fagin and J. Y. Halpern. Reasoning about knowledge and probability. *J. ACM*, 41:340–367, 1994.

[5] K. Fine. In so many possible worlds. *Notre Dame J. Formal Logic*, 13:516–520, 1972.

[6] N. Olivetti and G. L. Pozzato. CondLean: A theorem prover for conditional logics. In *Automated Reasoning with Analytic Tableaux and Related Methods, TABLEAUX 2003*, vol. 2796 of *LNCS*, pp. 264–270. Springer, 2003.

[7] N. Olivetti, G. L. Pozzato, and C. Schwind. A sequent calculus and a theorem prover for standard conditional logics. *ACM Trans. Comput. Logic*, 8(4:22):1–51, 2007.

[8] D. Pattinson. Coalgebraic modal logic: Soundness, completeness and decidability of local consequence. *Theoret. Comput. Sci.*, 309:177–193, 2003.

[9] 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.

[10] D. Pattinson and L. Schröder. Generic modal cut elimination applied to conditional logics. In *Automated Reasoning with Analytic Tableaux and Related Methods, TABLEAUX 2009*, vol. 5607 of *LNCS*. Springer, 2009.

[11] M. Pauly. A modal logic for coalitional power in games. *J. Logic Comput.*, 12:149–166, 2002.

[12] L. Schröder. Expressivity of coalgebraic modal logic: The limits and beyond. *Theoret. Comput. Sci.*, 390:230–247, 2008.

[13] L. Schröder and D. Pattinson. How many toes do I have? Parthood and number restrictions in description logics. In *Principles of Knowledge Representation and Reasoning, KR 2008*, pp. 307–218. AAAI Press, 2008.

[14] L. Schröder and D. Pattinson. Shallow models for non-iterative modal logics. In *Advances in Artificial Intelligence, KI 2008*, vol. 5243 of *LNAI*, pp. 324–331. Springer, 2008.

[15] L. Schröder and D. Pattinson. Pspace bounds for rank-1 modal logics. *ACM Trans. Comput. Logic*, 10(2:13):1–33, 2009.

[16] M. Vardi. On the complexity of epistemic reasoning. In *Logic in Computer Science*, pp. 243–251. IEEE, 1989.