CEGAR-Tableaux: Improved Modal Satisfiability via Modal Clause-Learning and SAT

Rajeev Goré[1(✉)] and Cormac Kikkert[2]

[1] Vienna University of Technology, Vienna, Austria [2] Australian National University, Canberra, Australia
cormac.kikkert@anu.edu.au

Abstract. We present CEGAR-Tableaux, a tableaux-like method for propositional modal logics utilising SAT-solvers, modal clause-learning and multiple optimisations from modal and description logic tableaux calculi. We use the standard Counter-example Guided Abstract Refinement (CEGAR) strategy for SAT-solvers to mimic a tableau-like search strategy that explores a rooted tree-model with the classical propositional logic part of each Kripke world evaluated using a SAT-solver. Unlike modal SAT-solvers and modal resolution methods, we do not explicitly represent the accessibility relation but track it implicitly via recursion. By using "satisfiability under unit assumptions", we can iterate rather than "backtrack" over the satisfiable diamonds at the same modal level (context) of the tree model with one SAT-solver. By keeping modal contexts separate from one another, we add further refinements for reflexivity and transitivity which manipulate modal contexts once only. Our solver CEGARBox is, overall, the best for modal logics K, KT and S4 over the standard benchmarks, sometimes by orders of magnitude.

1 Introduction

The TABLEAUX and DL communities have strived for thirty years to provide practical theorem provers for non-classical logics while the SAT community has moved from efficiently solving SAT-problems with tens of propositional variables to solving problems with hundreds of variables. The "silver bullet" was conflict driven clause-learning [7,10]. Following Claessen and Rosén [1], Fiorentini et al. [3] and Goré et al. [5], we give a tableaux-like calculus containing "modal clause learning" to handle modal satisfiability, where a main procedure explores a rooted tree-model with worlds evaluated via an "oracle" SAT-solver. Our implementation, `CEGARBox`, uses multiple optimisations and, overall, is the best over the standard benchmarks, sometimes by orders of magnitude.

Consider monomodal logic with modal operators $\Box$ and $\Diamond$ with formulae defined from atoms $p \in Atm$ by the BNF grammar $\varphi \ ::= \ \bot \mid \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \Box\varphi \mid \Diamond\varphi$. Define $(\varphi_1 \to \varphi_2) := (\neg\varphi_1 \vee \varphi_2)$ and $\varphi_1 \leftrightarrow \varphi_2 := ((\varphi_1 \to$

R. Goré—Work supported by the FWF projects I 2982 and P 33548.

© Springer Nature Switzerland AG 2021

$\varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1))$. We assume familiarity with the Kripke semantics for modal logics in which the modal logic K, KT, K4 and S4 are respectively characterised by all; reflexive; transitive; and reflexive-transitive frames.

We thank Steve Blackburn, Ullrich Hustadt and Daniel Le Berre.

2 Modal Clausal Tableaux

Following Goré and Nguyen [4], we define modal clausal tableaux as follows.

A *literal* is an atom p or its negation $\neg p$: we use a to f and l for literals. We use A, B, C and D for a set of literals. We define $\bar{l} := \neg p$ if $l = p$ and $\bar{l} := p$ if $l = \neg p$ so that $\bar{\bar{l}} = l$. A formula is in negation normal form (NNF) if it is implication-free and negations appear only in front of atomic formulae. A formula φ can be converted into an at most polynomially longer formula $nnf(\varphi)$ in NNF so that φ is logically equivalent to $nnf(\varphi)$. Let $\overline{\varphi} := nnf(\neg\varphi)$.

A *cpl-clause* is a disjunction of literals. A formula $(\neg a \vee \Box b)$ is a box-clause and $(\neg c \vee \Diamond d)$ is a dia-clause. We usually write box-clauses as $a \rightarrow \Box b$ and dia-clauses as $c \rightarrow \Diamond d$ to convey that the literal b is "boxed" while the literal d is "diamonded" and that these implications "fire" from left to right if their antecedents are true. For any set w_0 of these three types of clauses, let $\mathcal{C}^{cpl}(w_0)$ and $\mathcal{C}^{\rightarrow\Box}(w_0)$ and $\mathcal{C}^{\rightarrow\Diamond}(w_0)$ be, respectively, the set of cpl-clauses, box-clauses and dia-clauses from w_0.

A modal context is a possibly empty sequence of box-like modalities: formally $\Box^0\varphi := \varphi$ and $\Box^{i+1} := \Box^i\Box\varphi$. Every cpl-clause, box-clause and dia-clause is a modal clause, and if φ is a modal clause then so is $\Box^i\varphi, i \geq 1$. Using ";" for set-union, a set w_0 of modal clauses can be partitioned into separate modal contexts via: $w_0 = \Box^0\mathcal{C}_0(w_0) \; ; \; \Box^1\mathcal{C}_1(w_0) \; ; \; \cdots \; ; \; \Box^n\mathcal{C}_n(w_0)$ where each set $\mathcal{C}_i$ contains only cpl-clauses, box-clauses and dia-clauses so $\mathcal{C}_0(w_0) = \mathcal{C}^{cpl}(w_0) \; ; \; \mathcal{C}^{\rightarrow\Box}(w_0) \; ; \; \mathcal{C}^{\rightarrow\Diamond}(w_0)$. Letting $\mathcal{MC} := \Box^0\mathcal{C}_1; \cdots ; \Box^{k-1}\mathcal{C}_k$, we gather the non-zero modal contexts via $w_0 = \mathcal{C}^{cpl}(w_0); \mathcal{C}^{\rightarrow\Box}(w_0); \mathcal{C}^{\rightarrow\Diamond}(w_0); \Box\mathcal{MC}(w_0)$.

A formula can be put into modal clausal form (or Mints [11] normal form) in linear time and space wrt length [4]. The resulting modal clauses are K-satisfiable iff the original formula is K-satisfiable [4].

We assume familiarity with the standard tableau calculi for modal logics K, KT, K4 and S4 using NNF. These calculi will also work for formulae in modal clausal form. In the modal rules, L is a finite set of literals, while X, Y, and Z are possibly empty sets of modal clauses:

$$(T)\ \frac{\Box\varphi; X}{\varphi; \Box\varphi; X} \qquad (K)\ \frac{\Diamond\varphi; \Box X; \Diamond Y; L}{\varphi; X} \qquad (K4)\ \frac{\Diamond\varphi; \Box X; \Diamond Y; L}{\varphi; X; \Box X}$$

Suppose we want to test the formula φ_0 for validity. We negate it and put the negation into nnf to obtain $\overline{\varphi_0} := nnf(\neg\varphi_0)$. We then put $\overline{\varphi_0}$ into modal clausal form to obtain w_0. Thus w_0 is the modal clausal form of $nnf(\neg\varphi_0)$. We then use the rules shown above to try to find a closed tableau, as usual. But there is an alternative which builds-in some aspects of modus ponens as explained next.

Given an example root node $w_0 := L\ ;\ \{c \to \Diamond d, c_1 \to \Diamond d_1\}\ ;\ \{a_1 \to \Box b_i, a_2 \to \Box b_2\}\ ;\ \Box\mathcal{MC}(w_0)$, where L is a set of literals and $\Box\mathcal{MC}(w_0)$ is arbitrary, consider the (transitional-but-modus-ponens-like) KE-rule [2] instance:

$$(jump)\ \frac{L\ ;\ c \to \Diamond d\ ;\ a_1 \to \Box b_1, a_2 \to \Box b_2\ ;\ \Box\mathcal{MC}(w_0)}{d\ ;\ b_1\ ;\ \mathcal{MC}(w_0)}$$

Proposition 1. *If* $(\{c, a_1, \overline{a_2}\} \subseteq L$, *then this (jump) rule instance is derivable.*

Instead of a derivation of (jump), we simply show how it mimics (K) viz:

L: the set of literals as in (K)
$\Diamond\varphi$: a principal diamond $(c \to \Diamond d) \in \mathcal{C}^{\to\Diamond}(w_0)$ which fires giving $\Diamond d$ if $c \in L$
$\Diamond Y$: non-principal dia-clauses $\{c_1 \to \Diamond d_1\} = \mathcal{C}^{\to\Diamond}(w_0) \setminus \{c \to \Diamond d\}$
$\Box X$: box-clauses $(a_1 \to \Box b_1) \subseteq \mathcal{C}^{\to\Box}(w_0)$ giving $\{\Box b_1\} \subseteq \Box X$ if $a_1 \subseteq L$
$\Box X$: the non-empty modal contexts $\Box\mathcal{MC}(w_0) \subseteq \Box X$ and
none: box-clauses $a_2 \to \Box b_2$ which are dormant if $\overline{a_2} \subseteq L$ and have no counterpart in the original (K) rule.

We will replace the tableaux rules for cpl with a SAT-solver (oracle) and replace (K) with a generalised variant of (jump) called (jump)/(restart) which uses modal clause-learning. We first explain SAT-solvers and the CEGAR procedure.

3 SAT-solvers and the CEGAR Procedure

A formula is in conjunctive normal form (CNF) if it is a conjunction of cpl-clauses. SAT-solvers are extremely efficient algorithms for determining the satisfiability of a set of formulae of classical propositional logic in CNF [16].

Incremental SAT-solvers are solvers which allow alternating between adding a clause to the SAT-solver and testing for satisfiability. Further, modern SAT-solvers, such as MiniSAT [14], allow testing for *Satisfiability Under Unit Assumptions*, with a set of literals $A = \{l_1, \cdots, l_n\}$ called unit assumptions. That is, if the SAT-solver is in some state σ after loading a set S of cpl-clauses into it, we can now query whether or not $S \cup A$ is classically satisfiable. Moreover, after computing the un/satisfiability of $S \cup A$, such a SAT-solver will "undo" its actions to return to its previous state σ. Using this feature, we can use one single SAT-solver in state σ to *iteratively* test the classical satisfiability of many different extensions $S \cup \{A_1\}, S \cup \{A_2\}, \cdots, S \cup \{A_m\}$ of a given S without their interfering with one another, as long as each A_i is a set of unit assumptions.

For example, if we are given the set $\Box B\ ;\ \Diamond d_1\ ;\ \cdots\ ;\ \Diamond d_m\ ;\ \Box\mathcal{C}$ where each $B \cup \{d_i\}$ is a set of unit assumptions, and $\mathcal{C}$ is a set of arbitrary cpl-clauses, then we can initially load the SAT-solver with the cpl-clauses in $\mathcal{C}$ to put it in some state σ and then iteratively test the un/satisfiability of each set $B\ ;\ d_i\ ;\ \mathcal{C}$ for $i = 1, \cdots, m$ using just one SAT-solver which reuses the state σ, rather than using m separate SAT-solvers with states $B\ ;\ d_i\ ;\ \mathcal{C}$. We assume that the SAT-solver we use provides the following operations:

`addClause(`s, C`)`: adds the cpl-clause C as a constraint to the SAT-solver s.

`solve(`s, A`)`: accepts a set $A = \{l_1, \cdots, l_m\}$ of unit assumptions, and tries to find a classical valuation ϑ that satisfies the cpl-clauses added so far to s under the unit assumptions in A. The call returns one of two answers:
- (sat, ϑ): if it is possible to find such an assignment ϑ representing the literals that are true, and so we have that $A \subset \vartheta$
- (unsat, A'): if it is impossible to find such an assignment with $A' \subset A$ a, not necessarily unique, *unsatisfiable core* of A, which causes the classical unsatisfiability of $s \cup A$. Note that A' itself may be classically satisfiable. The smaller A' is the more efficient our algorithm becomes.

We also use the following operation as a shorthand to avoid complicating specifications with the intricacies of implementation:

$sat(\mathcal{C}, A)$: Creates a SAT-solver s, adds the set $\mathcal{C}$ of cpl-clauses to s, then returns `solve(`s`,`A`)` where A is a set of unit-assumptions (literals).

We use the SAT-solver MiniSat [14], in our implementation.

3.1 Counter-Example Guided Abstraction Refinement (CEGAR)

The standard way to use a SAT-solver, besides a direct translation, is called Counter-Example Guided Abstraction Refinement (CEGAR) which involves creating an under-abstraction ψ which is less constrained than the original formula φ. We use $\psi := \mathcal{C}^{cpl}(\varphi)$ as our under-abstraction. Using a SAT-solver, we check whether a classical valuation ϑ can be found for ψ. If not, then it is impossible to create a Kripke model for the more constrained φ, and we conclude that φ is modally unsatisfiable. Otherwise, we check whether the classical valuation ϑ can be extended into a Kripke model for φ. If so we conclude that φ is modally satisfiable. Else we refine the under-abstraction ψ to be closer to φ by learning new cpl-clauses from the classical unsatisfiable core, and repeat the whole procedure.

Some versions of CEGAR use an over-approximation or even both, but we elide details for brevity as the method of under-approximation is the one we use.

We now present two tableau-like rules and a rule-application search strategy to mimic modal clausal tableaux using a CEGAR approach.

4 CEGAR Tableaux: Modal Clause-Learning via SAT

We describe tableau-like rules which mimic CEGAR. Each rule has a single parent above the line and multiple children below the line with the traditional modal (rather than description logic) tableaux reading of "if the parent is modally satisfiable then so is at least one child". To handle "satisfiability under unit assumptions", let $\mathcal{A}(w_0) \subseteq w_0$ be a set of designated literals called **assumptions**.

local CPL satisfiability rule:

$$(\text{local})\ \frac{w_0 := \mathcal{A}(w_0)\ ;\ \mathcal{C}^{cpl}(w_0)\ ;\ \mathcal{C}^{\to\Box}(w_0)\ ;\ \mathcal{C}^{\to\Diamond}(w_0)\ ;\ \Box\mathcal{MC}(w_0)}{sat(\mathcal{C}^{cpl}(w_0), \mathcal{A}(w_0))}$$

where $sat(\mathcal{C}^{cpl}(w_0), \mathcal{A}(w_0))$ either returns "closed" because it finds an unsatisfiable core $\mathcal{A}'(w_0) \subseteq \mathcal{A}(w_0)$ of literals or returns "open" because it finds a classical valuation $\vartheta(w_0) \supseteq \mathcal{A}(w_0)$ such that $\vartheta(w_0) \models \mathcal{C}^{cpl}(w_0)$. We can implement $sat(\mathcal{C}^{cpl}(w_0), \mathcal{A}(w_0))$ with a SAT-solver via $\texttt{sat(}\mathcal{C}^{cpl}(w_0)\texttt{, }\mathcal{A}(w_0)\texttt{)}$.

Proposition 2. *If the parent of the (local) rule is modally satisfiable at some world w via $\vartheta(w)$ then its subset, the child, is classically satisfiable under $\vartheta(w)$.*

modal (jump/restart) rule:

$$\frac{w_0 := \mathcal{A}(w_0) \ ; \ \mathcal{C}^{cpl}(w_0) \ ; \ \mathcal{C}^{\to\Box}(w_0) \ ; \ \mathcal{C}^{\to\Diamond}(w_0) \ ; \ \Box\mathcal{MC}(w_0)}{w_1 := d \ ; \ B \ ; \ \mathcal{MC}(w_0) \qquad w_0' := w_0 \ ; \ \varphi(w_0)} \ \vartheta(w_0)$$

where $\vartheta(w_0) \supseteq \mathcal{A}(w_0)$ is a classical valuation such that $\vartheta(w_0) \models \mathcal{C}^{cpl}(w_0)$ and
(1) there is at least one "fired" diamond $(c \to \Diamond d) \in \mathcal{C}^{\to\Diamond}(w_0)$ & $\vartheta(w_0) \models c$
(jump): left child $w_1 := d; B; \mathcal{MC}(w_0)$ for the fired diamond $\Diamond d$ where the "fired" (un)boxes are
(2) $B := \{b \mid (a \to \Box b) \in \mathcal{C}^{\to\Box}(w_0) \text{ and } \vartheta(w_0) \models a\}$
(restart): right child $w_0' := w_0; \varphi(w_0)$ if the left child w_1 with $\mathcal{A}(w_1) = (d \ ; \ B)$ closes with an unsatisfiable core $\mathcal{A}'(w_1) \subseteq (d \ ; \ B)$ and

$$\begin{aligned} A_d(w_1) &:= \{d\} \cup \mathcal{A}'(w_1) \text{ is the unsatisfiable core of } w_1 \text{ extended with } d \\ CS(w_0) &:= \{e \mid \vartheta(w_0) \models e \ \& \ (e \to \Box f) \in \mathcal{C}^{\to\Box}(w_0) \ \& \ f \in A_d(w_1)\} \\ &\cup \ \{e \mid \vartheta(w_0) \models e \ \& \ (e \to \Diamond f) \in \mathcal{C}^{\to\Diamond}(w_0) \ \& \ f \in A_d(w_1)\} \\ &= \ \{l_1, \ldots, l_n\} \supseteq \{c\} \text{ are the "culprits" from } w_0 \end{aligned}$$

and $\varphi(w_0) := (\neg l_1 \vee \ldots \vee \neg l_n)$ is the local learned cpl-clause $\varphi(w_0)$ with which we restart w_0 as w_0' to refine $\vartheta(w_0)$ since $\vartheta(w_0) \models c$ and the single diamond $\Diamond d$ that it fired leads to a counter example w_1 w.r.t. $\Box\mathcal{MC}(w_0)$.

The (local) and (jump)/(restart) rules are notionally applicable to any set w_0 of modal clauses except that the (jump/restart) rule is additionally parametrised by a classical valuation $\vartheta(w_0)$: that is, they form a "producer-consumer" pair. Thus the (local) rule searches for a classical valuation (using a SAT-solver that returns (sat, ϑ)), effectively finding an open branch of static rule applications. The (jump) rule then uses this valuation to mimic the (K) as follows.

Item (1) non-deterministically chooses a dia-clause $c \to \Diamond d$ from $\mathcal{C}^{\to\Diamond}(w_0)$ which "fires" because $\vartheta(w_0) \models c$ giving us the principal formula $\Diamond d$ of (K).

Item (2) collects each box-clause $a \to \Box b$ from $\mathcal{C}^{\to\Box}(w_0)$ which "fires" because $\vartheta(w_0) \models a$, and unboxes each $\Box b$ producing literals $B \subseteq X$ in the (K)-rule.

The left (jump) child $w_1 := d \ ; \ B \ ; \ \mathcal{MC}(w_0)$ mimics the conclusion of a (K)-rule instance with a premise $\Diamond d \ ; \ \Box B \ ; \ \Box\mathcal{MC}(w_0)$, so w_1 is the set of formulae which must be true at the R-successor of the premise w_0.

Applying these two rules **recursively** will either close w_1 or leave w_1 open.

If w_1 stays open then w_1 is a putative R-successor in the underlying counter-model that we are exploring so we must choose some other dia-clause which is fired by $\vartheta(w_0)$: we must **iterate** over all such fired diamonds as we are looking for a closed tableau but if all choices stay open then we have a Kripke model.

Else, if w_1 closes then it will return an (there may be many) unsatisfiable core $\mathcal{A}'(w_1) \subseteq (d\ ;\ B)$, closing the tableau branch for $\Diamond d$ and pinpointing the unit assumptions from w_1 which cause branch closure, effectively building in a use-check as explained below.

In the "else" case, a traditional tableau would backtrack up to the next highest application of the $(\vee)$-rule. But we can be cleverer by learning a clause as follows. We extend the unsatisfiable core $\mathcal{A}(w_1)'$ to $A_d(w_1)$ to ensure that d is in $A_d(w_1)$ because $\Diamond d$ was the principal formula of the "jump" from w_0 to w_1. We now find the "culprits" $e \in \vartheta(w_0)$ by "unfiring" each $e \to \Box f$ and each $e \to \Diamond f$ that caused their f to be put into the extended unsatisfiable core $A_d(w_1)$ of the R-successor, thereby obtaining the conflict set $CS(w_0)$ (used in the proofs) of w_0. That is, we have moved from w_1 back to w_0.

We know there is at least one culprit in w_0, namely c, but in general $CS(w_0) = \{l_1, \cdots, l_n\} \supseteq \{c\}$. We therefore "switch off" at least one of these culprits by adding the disjunction of their negations $\varphi(w_0) := (\neg l_1 \vee \cdots \vee \neg l_n)$ as a new clause and restart w_0 as $w_0' := w_0\ ;\ \varphi(w_0)$. Intuitively, rather than backtracking to the next highest disjunction, our traditional tableau procedure is effectively re-starting the Static rules on the new incarnation w_0' to find a "saturation" that is guaranteed to be different from $\vartheta(w_0)$. Traditional tableau would only be guaranteed to find a different "saturation" if they included use-check or cut.

Proposition 3. *If the parent w_0 of the (jump/restart) rule is K-satisfiable in a Kripke model with root valuation $\vartheta(w_0)$ then so is its left (jump) child w_1.*

Proposition 4. *If the (jump/restart) rule is applied with $\vartheta(w_0)$ and the right (restart) child w_0' is classically satisfied by $\vartheta(w_0')$ then $\vartheta(w_0')$ is a different classical valuation from $\vartheta(w_0)$, and all previous such restarts, as there is at least one literal l_i which is true in the previous valuation but false in the new one.*

Example 1. Consider the standard K axiom instance $\Box(p \to q) \to (\Box p \to \Box q)$. We negate it and obtain the negation normal form $\neg K$: $\Box(\neg p \vee q) \wedge \Box p \wedge \Diamond \neg q$. A legitimate clausal form of $\neg K$ for illustrative purposes is: $w_0 = \{a_1, a_1 \to \Box b_1, \Box(b_1 \to (\neg p \vee q)), a_2, a_2 \to \Box p, c_1, c_1 \to \Diamond \neg q\}$ with $\mathcal{A}(w_0) = \emptyset$ and $\mathcal{C}^{cpl}(w_0) = \{a_1, a_2, c_1\}$ and $\mathcal{C}^{\to\Box}(w_0) = \{a_1 \to \Box b_1, a_2 \to \Box p\}$ and $\mathcal{C}^{\to\Diamond}(w_0) = \{c_1 \to \Diamond \neg q\}$ and $\Box\mathcal{C}(w_0) = \{\Box(b_1 \to (\neg p \vee q))\}$. Our final optimised normal forming procedure produces something smaller. Figure 1 contains the search-space for the resulting closed tableau. If we try $\Box(p \to q) \to (\Box p \to \Box r)$ then $c_1 \to \Diamond \neg q$ above becomes $c_1 \to \Diamond \neg r$ and the tableau will remain open and will return a Kripke (counter-)model $w_0 R w_1$ with $\vartheta(w_0) = \{a_1, a_2, c_1\}$ and $\vartheta(w_1) = \{b_1, p, q, \neg r\}$ which falsifies $\Box(p \to q) \to (\Box p \to \Box r)$ at w_0.

4.1 Termination, Soundness and Completeness of the Strategy

We dub our search strategy as **CEGARTab** (in bold font).

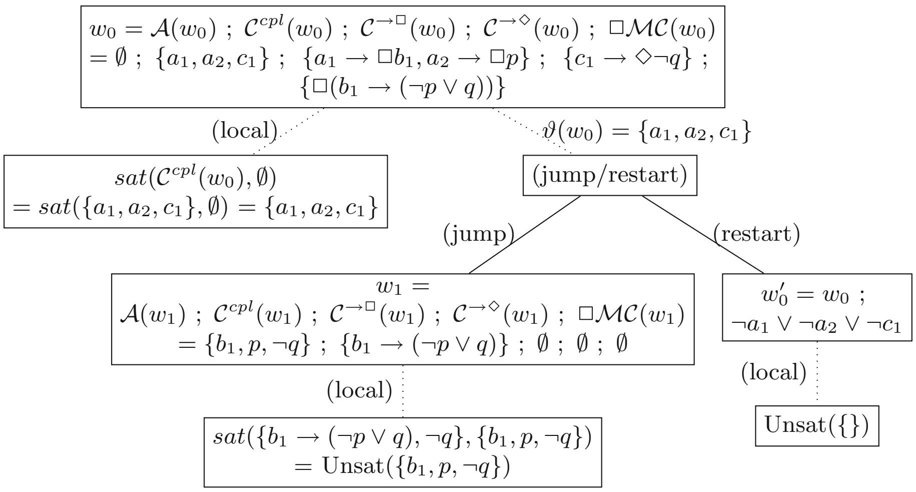
$w_0 = \mathcal{A}(w_0)$; $\mathcal{C}^{cpl}(w_0)$; $\mathcal{C}^{\rightarrow\Box}(w_0)$; $\mathcal{C}^{\rightarrow\Diamond}(w_0)$; $\Box\mathcal{MC}(w_0)$
$= \emptyset$; $\{a_1, a_2, c_1\}$; $\{a_1 \rightarrow \Box b_1, a_2 \rightarrow \Box p\}$; $\{c_1 \rightarrow \Diamond\neg q\}$;
$\{\Box(b_1 \rightarrow (\neg p \vee q))\}$
(local)
$\vartheta(w_0) = \{a_1, a_2, c_1\}$
$sat(\mathcal{C}^{cpl}(w_0), \emptyset)$
$= sat(\{a_1, a_2, c_1\}, \emptyset) = \{a_1, a_2, c_1\}$
(jump/restart)
(jump)
(restart)
$w_1 =$
$\mathcal{A}(w_1)$; $\mathcal{C}^{cpl}(w_1)$; $\mathcal{C}^{\rightarrow\Box}(w_1)$; $\mathcal{C}^{\rightarrow\Diamond}(w_1)$; $\Box\mathcal{MC}(w_1)$
$= \{b_1, p, \neg q\}$; $\{b_1 \rightarrow (\neg p \vee q)\}$; $\emptyset$; $\emptyset$; $\emptyset$
$w_0' = w_0$;
$\neg a_1 \vee \neg a_2 \vee \neg c_1$
(local)
(local)
Unsat($\{\}$)
$sat(\{b_1 \rightarrow (\neg p \vee q), \neg q\}, \{b_1, p, \neg q\})$
$=$ Unsat($\{b_1, p, \neg q\}$)

Fig. 1. The search-space for the negation $\neg$K of the K axiom $\Box(p \rightarrow q) \rightarrow (\Box p \rightarrow \Box q)$ where dotted lines indicate rule choices and solid lines indicate branching rules.

Each **iteration** in **CEGARTab** is finite because each node contains a finite number of dia-clauses. Thus the only way to not terminate is for **CEGARTab** to **recurse** for ever. But each **recursion** via the (jump) rule reduces the maximal modal degree of the formula set in the child node and each **recursion** via the (restart) rule enumerates a different classical valuation from the finite set of classical valuations for $\mathcal{C}^{cpl}(w_0)$. Thus **CEGARTab** must terminate.

Theorem 1. *For all sets of modal clauses $w_0 := nnf(\neg\varphi_0)$, **CEGARTab**($w_0$) returns closed iff w_0 is K-unsatisfiable (and hence φ_0 is K-valid).*

Proof. Both proofs proceed by a simple induction on the number of restarts.

Soundness: If the (local) rule returns closed then w_0 contains a classically unsatisfiable, and hence K-unsatisfiable, subset. Else there is a closed application of the (jump)/(restart) rule with learned clause $\varphi = (\neg l_1 \vee \cdots \vee \neg l_n)$. The induction hypothesis on the closed (jump) child implies that $w_0 \cup \neg\varphi$ is K-unsatisfiable. The induction hypothesis on the closed (restart) child implies that $w_0 \cup \varphi$ is K-unsatisfiable. Hence w_0 is K-unsatisfiable (and cut is admissible!).

Completeness: The open (local) rule returns $\vartheta(w_0)$. If the (jump)/(restart) rule is not applicable then there are no fired diamonds, and so the Kripke model is just a dead-end w_0 with $\vartheta(w_0)$. Else if the (jump) child is open then the induction hypothesis implies that we can extend $\vartheta(w_0)$ into Kripke submodels for every diamond jump. Adding a new root with $\vartheta(w_0)$ that sees all these sub-models gives a Kripke model for w_0 itself. Else if the (jump) child is closed then the (restart) child with learned clause $\varphi = (\neg l_1 \vee \cdots \vee \neg l_n)$

is open. Then the induction hypothesis implies that $w_0; \varphi$ is K-satisfiable, which implies that w_0 is K-satisfiable. $\square$

5 Implementation: Our Modal Satisfiability Tester `CEGARBox`

The only data-structures our base algorithm uses are a trie and lists. Memoisation, outlined in Sect. 7.2, was implemented using a binary tree.

5.1 Initialising a Trie During Normal Forming

Normal forming creates new atomic "names" p_ψ for certain subformulae ψ of the original formula: for example $\Box\Diamond(p_1 \wedge p_2)$ becomes a_1 ; $a_1 \rightarrow \Box b_2$; $\Box(b_2 \rightarrow \Diamond d_3)$; $\Box\Box(d_3 \rightarrow p_1)$; $\Box\Box(d_3 \rightarrow p_2)$ [4] where a_1 names $\Box\Diamond(p_1 \wedge p_2)$ and d_3 names $(p_1 \wedge p_2)$. We make a linear recursive descent of the formula and store modal clauses in a trie where each trie-node represents a modal context. If we stored modal contexts, our normal form would be quadratic in size, and thus our algorithm would have a quadratic time and space complexity, as does the one from Goré and Nguyen [4]. Below is a trie that stores the above clauses:

$$\begin{tikzcd}
\boxed{a_1 \ ; \ a_1 \rightarrow \Box b_2} \arrow[r, "\Box"] & \boxed{b_2 \rightarrow \Diamond d_3} \arrow[r, "\Box"] & \boxed{d_3 \rightarrow p_1 \ ; \ d_3 \rightarrow p_2}
\end{tikzcd}$$

Each node of the trie at a given level (context) has the following components:

`sat`: A SAT-Solver initialised with the purely classical clauses $\mathcal{C}^{cpl}(.)$ in the node
`BoxCl`: the box clauses $\mathcal{C}^{\rightarrow\Box}(.)$ of the form $a \rightarrow \Box b$ in the node
`DiaCl`: the dia-clauses $\mathcal{C}^{\rightarrow\Diamond}(.)$ of the form $c \rightarrow \Diamond d$ in the node
`Child(1)`: the node's (only) child node "containing" $\mathcal{MC}(.)$ as explained next.

Proposition 5. *If the input set of modal clauses is the set* $w_0 := \mathcal{C}^{cpl}(w_0)$ *;* $\mathcal{C}^{\rightarrow\Box}(w_0)$ *;* $\mathcal{C}^{\rightarrow\Diamond}(w_0)$; $\Box^1\mathcal{C}_1$ *; ... ;* $\Box^k\mathcal{C}_k$ *then the trie has depth equal to the maximal modal depth* k *of* w_0 *and* $\forall i \geq 0$, ***`Trie.node`*** *at depth* i *contains* $\mathcal{C}_i := \mathcal{C}_i^{cpl}$ *;* $\mathcal{C}_i^{\rightarrow\Box}$ *;* $\mathcal{C}_i^{\rightarrow\Diamond}$ *with* ***`Trie.node.sat`*** $= \mathcal{C}_i^{cpl}$ *and* ***`Trie.node.BoxCl`*** $= \mathcal{C}_i^{\rightarrow\Box}$ *and* ***`Trie.node.DiaCl`*** $= \mathcal{C}_i^{\rightarrow\Diamond}$ *and* ***`Trie.node.Child(1)`*** $= \mathcal{C}_{i+1}$ *: as below.*

Logic	Trie Depth				Intuition where `TrieNode(i)` is i-th node of Trie
	0	1	$\cdots$	k	k is the maximum modal depth of the given w_0
K	$\mathcal{C}_0$	$\mathcal{C}_1$	$\cdots$	$\mathcal{C}_k$	finite chain with `TrieNode(k).child(1) = nil`

Algorithm 1 CEGARBox(A, TrieNode)

```
1:  {Inputs: $A$ is a set of unit assumptions and TrieNode is at level $i$ in our trie}
2:  Let $t_0$ := solve(TrieNode.sat, $A$) {apply the (local) rule}
3:  if $t_0$ = (unsat, $A'$) then
4:      return Unsatisfiable($A'$)
5:  else if $t_0$ = (sat, $\vartheta$) then
6:      {Check box- and dia-clauses that fire under classical valuation $\vartheta$}
7:      for every $(c \to \Diamond d) \in$ TrieNode.DiaCl with $c \in \vartheta$ do
8:          Let $B = \{b \mid (a \to \Box b) \in$ TrieNode.BoxCl and $a \in \vartheta\}$
9:          {apply the (jump) rule at next modal context}
10:         if CEGARBox(($d$ ; $B$), TrieNode.child(1)) = Unsatisfiable(X') then
11:             Let $C = \{c\} \cup \{a \mid (a \to \Box b) \in$ TrieNode.BoxCl and $a \in \vartheta$ and $b \in X'\}$
12:             { Learn new clause $\varphi := \neg C$}
13:             Let $\varphi := \bigvee_{l \in C} \neg l$
14:             addClause(TrieNode.sat, $\varphi$) {modify the $i$-th context globally}
15:             { apply (restart) }
16:             return CEGARBox($A$, TrieNode)
17:         end if
18:     end for
19:     return Satisfiable {because every fired diamond is fulfilled}
20: end if
```

Fig. 2. The main algorithm of CEGARBox with A a set of unit assumptions

5.2 The Main Algorithm

Our algorithm follows Fiorentini et al.'s reworking [3] of `intuit` for propositional intuitionistic logic of Claessen and Rosén [1], which itself was "inspired" by `bddtab` of Goré et al. [5]. The pseudocode is in Fig. 2. Our implementation does **not** return an actual Kripke model, nor a closed tableau, as such, but it is trivial to extend it with the bookkeeping required to do so.

5.3 Inputs and Outputs

We write `node.child(i)` for the child labelled i of the trie rooted at `node`: as our logic is monomodal, $i = 1$. Similar to SAT-solvers we allow the use of a set of unit assumptions A. Our algorithm either returns Satisfiable, or Unsatisfiable(A'), where $A' \subset A$ is an unsatisfiable core of A.

We call `CEGARBox`(A, `Trie`) as the initial call with $A = \emptyset$.

Note that line 11 computes the correct conflict set as per the (jump)/(restart) rule because we ensure that no two box-clauses have the same RHS and no two dia-clauses have the same RHS, as explained later.

5.4 Use of Satisfiability Under Unit Assumptions

Note that in Line 10 of Fig. 2, we call the main algorithm recursively on `Trie.child(1)` with a set $X = (d\ ;\ B)$ of unit assumptions dependent

Logic	Trie Depth				Intuition where `TrieNode(i)` is i-th node of Trie
	0	1	$\cdots$	k	k is the maximum modal depth of the given w_0
K	$\mathcal{C}_0$	$\mathcal{C}_1$	$\cdots$	$\mathcal{C}_k$	finite chain with `TrieNode(k).child(1) = nil`
KT	$\mathcal{C}_0$	$\boldsymbol{\mathcal{C}}_1^k$	$\cdots$	$\boldsymbol{\mathcal{C}}_k^k$	descending chain with `TrieNode(k).child(1) = nil`
K4	$\mathcal{C}_0$	$\boldsymbol{\mathcal{C}}_1^1$	$\cdots$	$\boldsymbol{\mathcal{C}}_1^k$	ascending chain with `TrieNode(k).child(1) = TrieNode(k)`
S4	$\mathcal{C}_0; \boldsymbol{\mathcal{C}}_1^k$	$\boldsymbol{\mathcal{C}}_1^k$	$\cdots$	$\boldsymbol{\mathcal{C}}_1^k$	fixpoint at depth 1 with `TrieNode(1).child(1) = TrieNode(1)`

Fig. 3. The structure of the Trie for different logics with a modal context $\Box^0\mathcal{C}_0$; $\Box^1\mathcal{C}_1$; $\cdots$; $\Box^n\mathcal{C}_n$ and $\boldsymbol{\mathcal{C}}_i^n := \mathcal{C}_i$; $\cdots$; $\mathcal{C}_n$. For K, the modalised contexts form a descending chain $\boldsymbol{\mathcal{C}}_1^k \supseteq \boldsymbol{\mathcal{C}}_2^k \supseteq \cdots \supseteq \boldsymbol{\mathcal{C}}_k^k$ while for K4 (not implemented) they form an ascending chain $\boldsymbol{\mathcal{C}}_1^1 \subseteq \boldsymbol{\mathcal{C}}_1^2 \subseteq \cdots \subseteq \boldsymbol{\mathcal{C}}_1^k$. For S4, they are the constant $\boldsymbol{\mathcal{C}}_1^k$ after depth 1.

upon the fired dia-clause $c \to \Diamond d$. Moreover, this call is inside a for-loop which iterates over the fired diamonds. That is, if the set of fired diamonds is $\{c_1 \to \Diamond d_1, \cdots, c_n \to \Diamond d_n\}$, and the set of fired boxes gives $B = \{b \mid (a \to \Box b \in$ `Trie.BoxCl` and $a \in \vartheta\}$ then the putative n successor worlds must contain the unit assumption sets $X_1 = (d_1 \ ; \ B)$ and $X_2 = (d_2 \ ; \ B)$ up to $X_n = (d_n \ ; \ B)$. We *iteratively* test the classical satisfiability of each set X_i ; `Trie.child(1)` by putting $X = (b_i \ ; \ B)$ while keeping the parameter `Trie.child(1)` constant. This is sound because the sat-solver `Trie.child(1).sat` in `Trie.child(1)` undoes the assumptions it makes while computing the classical satisfiability of one set d_1 ; B ; `Trie.child(1)` (say) so that the same sat-solver `Trie.child(1).sat` can be reused for the next set d_2 ; B ; `Trie.child(1)` (say) without their interfering with each other. That is, this is only sound because our SAT-solver provides the ability to test for "satisfiability under unit assumptions".

5.5 Modal Clause Learning Modifies the Modal Context at Level i

Note that we learn a new cpl-clause in Line 14 via `addClause(Trie.sat,` φ`)`.

Consider a set of formulae and suppose that we saturate it using the traditional static tableau rules for cpl giving two OR-leaves, $\Diamond\varphi_1$; $\Box X_1$; $\Diamond Y_1$; L_1 and $\Diamond\varphi_2$; $\Box X_2$; $\Diamond Y_2$; L_2 where each L_i is a set of literals. Thus we can treat L_1/L_2 as a classical valuation ϑ_1/ϑ_2 which assigns all members of L_1/L_2 to true.

Suppose we try the successor φ_1 ; X_1 and find that it is modally unsatisfiable. Putting $\hat{X}_1$ for the conjunction of the members of X_1, we know that $\hat{X}_1 \to \neg\varphi_1$ is K-valid, independently of ϑ_1 itself. By necessition, we know that $\Box(\hat{X}_1 \to \neg\varphi_1)$ is K-valid. Goré et al. [5] tried to implement this insight into `bddtab` but it was refined nicely into the current form by Claessen and Rosén [1].

As explained previously, the i-th level of our `Trie` stores the cpl-clauses $\mathcal{C}_i^{cpl}$ from the i-th modal context $\Box^i\mathcal{C}_i$ inside the sat-solver `Trie.sat` at level i. Thus `addClause(Trie.sat,` φ`)` modifies the i-th modal context across level i.

We now describe extensions to handle the modal logics KT and S4.

6 Extensions to KT and S4

Three aspects of our framework handle modalities: the modal contexts $\Box\mathcal{MC} = \Box^1\mathcal{C}_1; \Box^2\mathcal{C}_2; \cdots ; \Box^k\mathcal{C}_k$ with $\mathcal{C}_i$ stored in the i-th level of the Trie; fired box-clauses $a \to \Box b$ when $\vartheta \models a$; and fired dia-clauses $c \to \Diamond d$ when $\vartheta \models c$.

Capturing Reflexivity. The characteristic axioms for reflexivity are $\Box\varphi \to \varphi$ and its dual $\varphi \to \Diamond\varphi$ so we make the following modification in these three aspects:

modal contexts: starting from level k, for all Trie nodes at levels $i \geq 1$, add `TrieNode.child(1)` to `TrieNode` so that the contexts in the `Trie` form a descending chain, building $\Box^i\mathcal{C}_i \to \mathcal{C}_i$ globally into the Trie, see Fig. 3

fired box-clauses: for every context $\mathcal{C}_i$ of modal clauses, if $(a \to \Box b) \in \mathcal{C}_i$, add the cpl-clause $(a \to b)$ to $\mathcal{C}_i$, building in the T-axiom $\Box b \to b$

fired dia-clauses: when calculating "fired" diamonds via "$(c \to \Diamond d) \in$ `Trie.DiaCl with` $c \in \vartheta$", add the extra condition "`and` $d \notin \vartheta$". Thus, $(c \to \Diamond d) \in \mathcal{C}^{cpl}(w_0)$ fires only if $\vartheta(w_0) \not\models d$ since w_0 is its own successor.

Termination is as before for K. Soundness is obvious. For completeness, take the reflexive closure of the tree-model created by our procedure. Why can the deepest world be made reflexive when `TrieNode(k)` is not its own child? It contains no box-clauses that fire so $\Box\varphi \to \varphi$ holds there vacuously.

Capturing Transitivity. Traditional proof-search in K4 can loop: e.g., the node $\{\Box\Diamond p, \Diamond p\}$ usually creates an infinite sequence of (K4)-successors each containing the set $\{p, \Box\Diamond p, \Diamond p\}$, leading to an infinite branch unless we check for ancestor loops. Thus the modal satisfiability of a given world depends not only on its assumptions but also on its ancestors because a world w might be modally satisfied only because some descendent v of w loops back to one of w's ancestors.

The characteristic axiom $\Box\varphi \to \Box\Box\varphi$ for transitivity implies that $\Box\varphi \to \Box^i\varphi$ for all $i \geq 1$. So the modal contexts form an ascending chain: see Fig. 3.

modal contexts: In the K4 `Trie`, the k-th level is its own child and level i contains $\boldsymbol{\mathcal{C}}_1^n := \mathcal{C}_1 \ ; \ \cdots \ ; \ \mathcal{C}_n$ building in $\Box\varphi \to \Box\Box\varphi$;

fired box-clauses: In the i-th node of `Trie`, replace every box-clause $a \to \Box b$ with $a \to \Box P_b$, and add the modal clauses $P_b \to \Box P_b$ and $P_b \to b$ to every node of the Trie from $i + 1$ to k where P_b is a new propositional variable for "persistent b", thereby encoding a finite state automaton that effectively turns $a \to \Box b$ into $a \to \Box^{j \geq i} b$, a technique from description logic tableaux;

ancestor loop-check: We add an additional input to `CEGARBox`, which is a list of the classical valuations found for the ancestors of the current world w_0. If w_0 requires us to fulfil $\Diamond d \ ; \ \Box B$, where $A = (d \ ; \ B)$ and some ancestor w_a, has $\vartheta(w_a) \models A$, then we can return satisfiable because we can just put $w_0 R w_a$.

Termination follows via ancestor loop-check. Soundness follows by noting that the above transformations are all sound. For completeness, we just take the transitive closure of our Kripke model.

Capturing Reflexivity and Transitivity Together. We add the changes for both KT and K4 as outlined above. The axiom $\Box\varphi \rightarrow \varphi$ made our Trie into a descending chain while the axiom $\Box\varphi \rightarrow \Box\Box\varphi$ made our Trie into an ascending chain, so adding both means that our Trie contains only two levels 0 and 1 with the second level being a fixed point. That is, level 1 is its own child with level 0 containing $\mathcal{C}_0 \ ; \ \boldsymbol{\mathcal{C}}_1^n$ and level 1 containing $\boldsymbol{\mathcal{C}}_1^n$: see Fig. 3.

Termination is by ancestor loop-check. Soundness follows by seeing that we are effectively encoding both of the (KT) and (K4) rules. For completeness, we just take the reflexive and transitive closure of our underlying Kripke model.

7 Optimisations Which Made `CEGARBox` faster

We utilise standard simplification techniques, truth propagation, formula sorting for the renaming process, and box lifting as described by Sebastiani and Vescovi [15]. We also used techniques from Nalon et al. [12] when normal forming to avoid new literals when old ones suffice. For modal logic K, this was implemented by keeping a map in every modal context that associates the literal l with the formula φ it names. Then when renaming any occurrence of φ, we check the map, and find l. For reflexivity, we can instead use a global map.

For brevity, we skip many small optimisations which allow us to avoid new literals, as each such literal potentially doubles the number of classical valuations the SAT-solvers need to search.

The running time of our algorithm depends on the number of box- and dia-clauses, so our final processing stage involves replacing these with cpl-clauses. Intuitively, the SAT-solver is better at handling cpl-clauses than `CEGARBox` is at handling modal clauses. We therefore do the following:

1. If two box-clauses $\{a_1 \rightarrow \Box b, a_2 \rightarrow \Box b\}$ or dia-clauses $\{a_1 \rightarrow \Diamond b, a_2 \rightarrow \Diamond b\}$, share a RHS b, create a new atomic formula p_a and replace them with $\{p_a \rightarrow \Box b/\Diamond b,\ a_1 \rightarrow p_a,\ a_2 \rightarrow p_a\}$ so that no two RHSs are the same.
2. If two box clauses $\{a \rightarrow \Box b, a \rightarrow \Box c\}$ share a LHS a, create a new literal p_a and replace these modal clauses with $a \rightarrow \Box p_a$, $\Box(p_a \rightarrow b)$, and $\Box(p_a \rightarrow c)$, thereby moving information from the box-clauses into the modal context;

We use negative polarity for all literals in MiniSAT so it sets unknown variables to false, thus decreasing the number of box- and dia-clauses that fire.

7.1 Reducing the Number of Dia-Clauses

We make minor changes to the algorithm of `CEGARBox` from Fig. 2. First, the previous optimisations mean that no two triggered dia-clauses have the same RHS, however, a triggered box $a \rightarrow \Box r$ and a triggered diamond clause $c \rightarrow \Diamond r$ may share the subformula r. Then, we can typically reduce the number of dia-clauses we have to consider. Let B and D be the set of RHSs of triggered box and dia-clauses, respectively. Instead of checking $B \ ; \ d$ for every $d \in D$, we can skip those with $d \in B$, as any world created by another triggered diamond clause will contain d. For the case $D \setminus B = \emptyset$ with $D \neq \emptyset$, we just check one diamond clause, as this one world (if created) will contain D.

If a conflict set X' for some fired diamond contains literals only from box-clauses, we know the box-clauses have no consistent successor, so we can learn the appropriate clauses for all (fired and unfired) dia-clauses in one hit.

Finally, note that we can learn a new clause only when we find an unfulfillable dia-clause. We experimented with a heuristic to remember unfulfillable dia-clauses by tracking how many times a given dia-clause lead to Unsatisfiable, and sorted them highest to lowest. In general this lead to improvements, but for some benchmarks the overhead of sorting lead to a slower time. So we experimented with a quicker approximation which instead just moves a clause to the front of the list when it leads to Unsatisfiable, which also lead to performance improvements. However, placing the failed dia-clauses at the end also lead to performance improvements, and requires further investigation.

7.2 Memoisation of Satisfiable Assumptions

Memoisation of Satisfiable Assumptions in K. We can store which assumptions have lead to Satisfiable wrt each particular modal context. During proof search, if we find that the current unit assumptions have been found to be satisfiable in the given modal context, we can immediately return Satisfiable, saving time.

We call this exact-cache, as we only return Satisfiable if we find that the exact same assumptions have lead to Satisfiable before. Assumptions were stored in a Binary Tree based implementation of a set. We experimented with a "subset cache" approach which returns Satisfiable if the current assumptions is a subset of any cached assumptions. To get more matches, once we find that some unit assumptions leads to satisfiable we would store not the unit assumptions but rather the whole classical valuation instead. While this gave more matches, it was implemented with the slow process of checking each set in the cache individually, which made it slower than exact-cache. A faster way of implementing a subset check over a collection of sets may make "subset cache" more feasible.

Memoisation of Satisfiable Assumptions in KT. Descending chains mean that an assumptions set A that leads to Satisfiable in the modal context $\Box^i$ will also lead to Satisfiable in the modal context $\Box^{i+1}$. Thus we can use a global cache, instead of a cache for each modal context, and store the assumptions A as well as the smallest i for which it returns Satisfiable at modal context $\Box^i$. Then when searching the assumptions in the modal context $\Box^{j \geq i}$ we immediately return Satisfiable if the assumptions occurs in the cache.

Conversely, any clause learnt in the modal context $\Box^i$ applies to the modal context $\Box^{i-1}$ since context $\Box^i$ is a subset of context $\Box^{i-1}$: not implemented yet.

Loop-Check and Memoisation of Satisfiability in S4. Recall that traditional proof-search in S4 require a loop-check for termination.

Caching not just the assumptions of a world but also its ancestors would work, however it is unlikely matches would ever occur, leading to a limited speed up. Instead we take a different approach, that allows us to avoid storing any information related to ancestors in the cache. The idea is to store worlds only if every world reachable from it is modally satisfiable. That is, if a world w_1 has been deemed to be modally satisfiable, but uses a back edge to w_0, we will only add w_1 to the cache once w_0 has been shown to be modally satisfiable.

So we modify the proof search to look for maximally isolated subgraphs: that is, submodels with a world that reaches only its descendants. Formally, a world w such that there is no back edge connecting a descendent of w to an ancestor of w. When w becomes satisfiable, we add all its descendants to the cache.

7.3 Two Phase Caching

One problem with the previous approach is that in the worst case the highest world reachable by a world might be so high (e.g. the root), that the procedure never caches any worlds leading to no speed improvements.

Suppose world y is satisfiable if an ancestor world x is satisfiable. Previously, we only cache y if and when x becomes satisfiable but we can actually treat y as cached satisfiable for all descendants of x. Such two-phase caching means y is in a temporary cache until x becomes satisfiable, when it moves to a global cache.

Caution: we have to check for self contained models inside bigger ones.

8 Benchmarks and Issues with MOSAIC

We now outline various issues we found during our experiments. Our benchmarks are from Nalon et al. [12] and (corrected) Lagniez et al. [9]:

LWB: extended LWB benchmarks created by Nalon et al. [12] but which need to be generated *in situ* from their instructions as they can take up 14 GB;
3CNF: 1000 randomly generated 3CNF$_K$ formulae over 3 to 10 propositional variables with modal depth 1 or 2 with 457 satisfiable and 464 unsatisfiable;
MQBF: the complete set of TANCS-2000 modalised random QBF formulae and the MQBF formulae provided by Kaminski and Tebbi.
KT and S4: the corrected extended benchmarks from MOSAIC.

The "new kid on the block" is MOSAIC, by Lagniez et al. [9]. But some of their extended LWB benchmark files were blank, unreadable, or lead to incorrect answers. We have confirmed these with Daniel Le Berre.

Daniel Le Berre sent us an executable binary for the latest version MOSAIC 2.4. Unfortunately, MOSAIC 2.4 returned wrong answers for many (corrected) benchmarks. Daniel Le Berre has retrospectively confirmed that MOSAIC 2.0 was also unsound. These issues undermine all of their experimental results [9].

We re-implemented the extended benchmark generator in python and confirmed that there were no differences with the original, smaller benchmarks.

9 Experimental Results

We used the following options: InKreSAT 1.0 - default; FaCT++ 1.6.3 - default; Spartacus 1.1.3 - default; BDDTab 1.0 - default; K$_S$P 0.1.3 - ordered; and Vampire 4.5.1 (OFT) (optimised functional translation) provided by Ullrich Hustadt. Our virtual machine had an Intel Xeon E5-2640@2.40 GHz CPU and 8GB of RAM. We also checked that all provers gave the same answers.

On the MQBF benchmarks, `ksp` is best, with `CEGARBox` second (Fig. 4). On the extended K-LWB benchmarks, `CEGARBox` is best (Fig. 4) with all 56 problems solved in the classes d4_n, d4_p, dum_n, dum_p, lin_n, path_n, path_p, poly_n, and poly_p within 15 s even though *"only the best current provers, if any at all, will be able to solve all the formulae within a time limit of 1000 CPU seconds"* [12]. No other prover managed to solve all 56 problems in any class. In 3CNF, `CEGARBox` triumphs after 1 s (Fig. 4). Over all K-benchmarks, `CEGARBox` at rougly 0.7 s beats every other prover at 15 s (Fig. 4). Indeed over all K-benchmarks, `CEGARBox` solves almost 2500 problems in just 15 s while Nalon et al. [13] report that no other prover solved more than about 2400 problems with 1000 s (16GB). For KT, `CEGARBox` dominates after 1 s (Fig. 5). For S4, `CEGARBox` is by far the best prover, beating every other prover within 0.25 s and solving over 600 problems in 15 s while the best other prover, `bddtab`, solves only 350 (Fig. 5).

10 Related Work

All SAT-based provers except `bddtab`, `intuit` and `CEGARBox` use explicit names for the reachability relation R: for example, a clause $r_{ij} \rightarrow \cdots$ encodes that "if world j is a successor of world i then ...". Instead, we put all formulae into their context while initialising the trie. Via the propositions about modal contexts, we can also move all formulae from modal contexts i to j directly. We believe that this is the reason for the massive improvement seen in our experiments.

As already stated, our approach is based on one from Claessen and Rosén [1], which itself was "inspired" by that of Goré et al. [5], so we articulate the differences. First, `intuit` handles propositional intuitionistic logic (Int), which is characterised by finite, rooted, reflexive and transitive Kripke models without any proper clusters, but as shown by Fiorentini et al. [3], `intuit` implements the loop-free sequent calculus g4ip so termination is not an issue. Second, the persistence property of Kripke models for Int allows them to propagate all formulae "along" the reachability relation using one incremental SAT-solver, while we must discard non-boxed formulae. Third, Claessen and Rosén outline "further work" for classical modal logics using only one SAT-solver, using a similar normal form, only one outermost $\Box$-context and a similar algorithm to ours, but we cannot find anything published about this work. They also do not mention reflexivity, transitivity, caching, loop-checking or optimisations. Thus our work is not "just an implementation of Claessen and Rosén".

InKreSAT [8] interleaves encoding phases with calls to an incremental SAT solver, but uses a labelled tableau calculus, and keeps an explicit encoding of R.

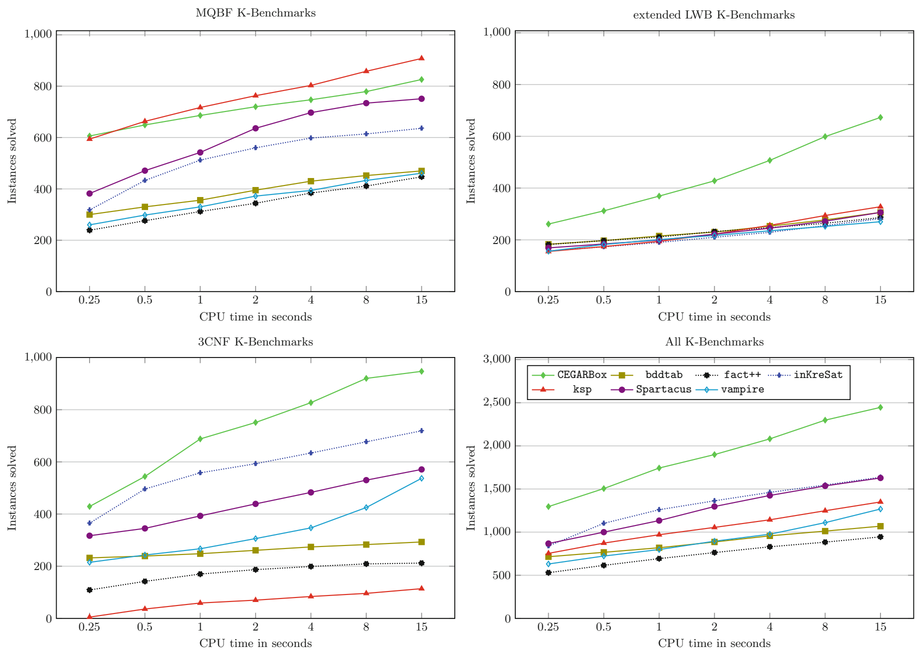

Fig. 4. Experimental results for the (extended) K benchmarks

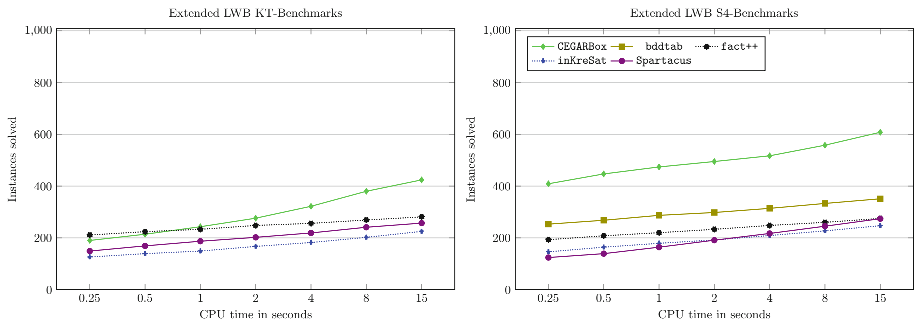

Fig. 5. Experimental results for the extended and corrected KT and S4 benchmarks

11 Further Work and Conclusions

Better heuristics for clause ordering will allow for both Sat and Unsat shortcuts. For example, we found it is possible to solve all instances of branch_p with one clause learnt per modal context but our final prover does not use this ordering.

Our K prover can be extended trivially to multi-modal logics, however for reflexive relations, the number of modal contexts a clause can belong to increases drastically, which most likely would slow down our prover as the number of different modalities increases. By ensuring that each subformula ψ is named uniquely with p_ψ, we can avoid keeping contexts and put $p_\psi \rightarrow \psi$ "globally". It is also easy to extend our prover to handle local and global assumptions. Symmetric relations require the notion of "too small" from Goré and Widmann [6].

Overall, `CEGARBox` is arguably the best prover for K, KT, and S4 on the standard benchmarks, sometimes by orders of magnitude.

Our repository is here: https://github.com/cormackikkert/CEGARBox.

Clearly, efficient SAT-based CEGAR-tableaux are possible for many different non-classical logics, including intuitionistic and modal (description) logics!

Finally, there is a very close connection between CEGAR-tableaux and the KE-tableaux of D'Agostino and Mondadori [2] which we are currently investigating. In particular, note that our proofs utilise a meta-level *semantic* cut-rule rather than a syntactic cut-rule: that is we have identified and absorbed all syntactic cuts required by KE-tableaux into the (jump)/(restart) rule!

Acknowledgment. Work supported by the FWF projects I 2982 and P 33548.

References

1. Claessen, K., Rosén, D.: SAT modulo intuitionistic implications. In: Davis, M., Fehnker, A., McIver, A., Voronkov, A. (eds.) LPAR 2015. LNCS, vol. 9450, pp. 622–637. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48899-7_43
2. D'Agostino, M., Mondadori, M.: The taming of the cut. Classical refutations with analytic cut. J. Log. Comput. **4**(3), 285–319 (1994)
3. Fiorentini, C., Goré, R., Graham-Lengrand, S.: A proof-theoretic perspective on SMT-solving for intuitionistic propositional logic. In: Cerrito, S., Popescu, A. (eds.) TABLEAUX 2019. LNCS (LNAI), vol. 11714, pp. 111–129. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29026-9_7
4. Goré, R., Nguyen, L.A.: Clausal tableaux for multimodal logics of belief. Fundam. Informaticae **94**(1), 21–40 (2009)
5. Goré, R., Olesen, K., Thomson, J.: Implementing tableau calculi using BDDs: BDDTab system description. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) IJCAR 2014. LNCS (LNAI), vol. 8562, pp. 337–343. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08587-6_25
6. Goré, R., Widmann, F.: Optimal and cut-free tableaux for propositional dynamic logic with converse. In: Giesl, J., Hähnle, R. (eds.) IJCAR 2010. LNCS (LNAI), vol. 6173, pp. 225–239. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14203-1_20
7. Bayardo, Jr., R.J., Schrag, R.: Using CSP look-back techniques to solve real-world SAT instances. In: Kuipers, B., Webber, B.L. (eds.) Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97, IAAI 97, Providence, Rhode Island, USA, 27–31 July 1997, pp. 203–208. AAAI Press/The MIT Press (1997)
8. Kaminski, M., Tebbi, T.: InKreSAT: modal reasoning via incremental reduction to SAT. In: Bonacina, M.P. (ed.) CADE 2013. LNCS (LNAI), vol. 7898, pp. 436–442. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38574-2_31

9. Lagniez, J.-M., Le Berre, D., de Lima, T., Montmirail, V.: A recursive shortcut for CEGAR: application to the modal logic K satisfiability problem. In: Sierra, C. (ed.) Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, 19–25 August 2017, pp. 674–680. ijcai.org (2017)
10. Marques-Silva, J.P., Sakallah, K.A.: GRASP - a new search algorithm for satisfiability. In: Rutenbar, R.A., Otten, R.H.J.M. (eds.) Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1996, San Jose, CA, USA, 10–14 November 1996, pp. 220–227. IEEE Computer Society/ACM (1996)
11. Mints, G.: Gentzen-type systems and resolution rules. In: Mints, G., Martin-Löf, P. (ed.) COLOG-88. LNCS, vol. 417, pp. 516–537. Springer (1988)
12. Nalon, C., Hustadt, U., Dixon, C.: KSP: a resolution-based prover for multimodal K, abridged report. In: Sierra, C. (ed.) Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, 19–25 August 2017, pp. 4919–4923. ijcai.org (2017)
13. Nalon, C., Hustadt, U., Dixon, C.: KSP: a resolution-based prover for multimodal K_n: architecture, refinements, strategies and experiments. J. Autom. Reason. **64**(3), 461–484 (2020)
14. Sörensson, N., Een, N.: http://minisat.se/Papers.html. Accessed 10 Feb 2020
15. Sebastiani, R., Tacchella, A.: SAT techniques for modal and description logics. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability. Frontiers in Artificial Intelligence and Applications, vol. 185, pp. 781–824. IOS Press (2009)
16. Various. http://www.satcompetition.org/. Accessed 10 Feb 2020