REPORTS ON MATHEMATICAL LOGIC
56 (2021), 57-74
doi:10.4467/20842589RM.21.006.14375

Tin PERKOV and Luka MIKEC

TABLEAU-BASED TRANSLATION FROM FIRST-ORDER LOGIC TO MODAL LOGIC

A b s t r a c t. We define a procedure for translating a given first-order formula to an equivalent modal formula, if one exists, by using tableau-based bisimulation invariance test. A previously developed tableau procedure tests bisimulation invariance of a given first-order formula, and therefore tests whether that formula is equivalent to the standard translation of some modal formula. Using a closed tableau as the starting point, we show how an equivalent modal formula can be effectively obtained.

1. Introduction

Kripke semantics makes modal logic a fragment of first-order logic, namely the bisimulation invariant fragment. Unfortunately, this is an undecidable

Received 19 January 2021
This work has been supported by Croatian Science Foundation (HRZZ) under the projects UIP-05-2017-9219 and IP-01-2018-7459.
Keywords and phrases: modal logic, bisimulation invariance, tableaux˙ AMS subject classification: 03B45.

fragment of first-order logic (see [1]). Truth clauses of modal formulas have obvious translations to first-order logic. This is the basis of standard translation from modal logic to first-order logic. Of course, the set of these translations is decidable, but the set of all first-order formulas (over the appropriate signature) each of which is equivalent to the standard translation of some modal formula is not decidable. By the Van Benthem Characterization Theorem, this is exactly the set of all formulas invariant under bisimulation, which makes the latter the basic equivalence between Kripke models.

In [6] we developed a tableau-based procedure to test whether a given first-order formula is bisimulation invariant. Using reduction to the standard first-order tableau (see, e.g., [7] for reference), we proved soundness and completeness. The latter implies semi-decidability of the problem, since it means that each bisimulation invariant formula has a closed tableau. In other words, in the case of an affirmative answer, the procedure does terminate and gives the correct answer, i.e., the test is positive.

In this paper we use a given closed tableau of a bisimulation invariant formula not only as a justification that this formula is equivalent to the standard translation of a modal formula, but also to obtain some such modal formula, using the tableau as a starting point.

In Section 2, we briefly overview basic notions and results from [6], for the sake of self-containment of the present paper. In Section 3, we present a procedure of obtaining a modal correspondent of a given bisimulation invariant first-order formula. We conclude with a brief description of an implementation of the procedure.

2. Preliminaries

We assume familiarity with modal logic (see e.g. [2] for further details if needed), so the following several paragraphs are here just in order to fix notation, terminology and a convenient choice of primitive symbols. We will only consider the basic modal language, the alphabet of which extends that of classical propositional logic with dual modalities $\Diamond$ and $\Box$. The syntax of modal formulas is given by

$$\varphi ::= p \mid \bot \mid \top \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \Diamond\varphi \mid \Box\varphi,$$

where p ranges over the set of propositional variables. We often write $\varphi \to \psi$ instead of $\neg\varphi \vee \psi$.

A *Kripke model* is $\mathfrak{M} = (W, R, V)$, where $W \neq \emptyset$, $R \subseteq W \times W$, and V is a *valuation*, a function that maps each propositional variable p to some $V(p) \subseteq W$.

Let σ be the first-order signature consisting of a binary relational symbol R and a unary relational symbol P for each propositional variable p. A Kripke model can be regarded as a σ-structure, with $|\mathfrak{M}| = W$, $R^{\mathfrak{M}} = R$ and $P^{\mathfrak{M}} = V(p)$ for each p. The *standard translation* is defined as follows:

$$\begin{aligned}
ST_x(p) &= Px, \text{ for each propositional letter } p \\
ST_x(\bot) &= \bot \\
ST_x(\top) &= \top \\
ST_x(\neg\varphi) &= \neg ST_x(\varphi) \\
ST_x(\varphi_1 \vee \varphi_2) &= ST_x(\varphi_1) \vee ST_x(\varphi_2) \\
ST_x(\varphi_1 \wedge \varphi_2) &= ST_x(\varphi_1) \wedge ST_x(\varphi_2) \\
ST_x(\Diamond\varphi) &= \exists y(Rxy \wedge ST_y(\varphi)) \\
ST_x(\Box\varphi) &= \forall y(Rxy \to ST_y(\varphi)),
\end{aligned}$$

where y in the last two clauses is a fresh variable.

Having the aforementioned identification of Kripke models and σ-structures in mind, we can speak about an equivalence between modal formulas and first-order formulas with one free variable: we say that a modal formula φ is *equivalent* to a first-order formula $F(x)$ if for each Kripke model $\mathfrak{M}$ and each world w in $\mathfrak{M}$, the formula φ is satisfied at w, which is denoted by $\mathfrak{M}, w \Vdash \varphi$, if and only if $\mathfrak{M} \models F(x)[w]$, i.e., if and only if $F(x)$ is satisfied in $\mathfrak{M}$, now regarded as a σ-structure, under the assignment of w to the variable x.

It is easy to see that for any modal formula φ we have $\mathfrak{M}, w \Vdash \varphi$ if and only if $\mathfrak{M} \models ST_x(\varphi)[w]$, i.e., the standard translation of φ is equivalent to φ.

A *bisimulation* between models $\mathfrak{M} = (W, R, V)$ and $\mathfrak{M}' = (W', R', V')$ is a non-empty relation $Z \subseteq W \times W'$ such that:

(at) if wZw', then for every p we have $w \in V(p)$ if and only if $w' \in V'(p)$;

(forth) if wZw' and Rwv, then there is v' such that vZv' and $R'w'v'$;

(back) if wZw' and $R'w'v'$, then there is v such that vZv' and Rwv.

We say that a σ-formula $F(x)$ is *bisimulation invariant* if the following holds: if there is a bisimulation Z between $\mathfrak{M}$ and $\mathfrak{M}'$ such that wZw', then we have $\mathfrak{M} \models F(x)[w]$ if and only if $\mathfrak{M}' \models F(x)[w']$.

By the Van Benthem Characterization Theorem, a σ-formula $F(x)$ is bisimulation invariant if and only if it is equivalent to the standard translation of some modal formula.

Generally, a tableau is a systematic search for a model that satisfies some formula. Thus we can test the validity of a formula with a tableau made for its negation. The idea of bisimulation invariance testing is also to search for a counterexample, in this case to construct two models and a bisimulation between them that does not preserve the truth of a given formula.

Let us briefly introduce the rules of *bisimulation invariance tableau* or *BI-tableau*. Let $F(x)$ be a σ-formula in which only variable x is free. Each node of a BI-tableau is a triple (A, B, C), which we write as $A \cdot B \cdot C$ where A and C can be the empty words or formulas over σ expanded with constant symbols introduced in the tableau, while B can also be the empty word, or have the form cZc'. We will not explicitly denote the empty word in the tableau, so for example a node such that B and C are empty is denoted $A \cdot \cdot$.

Let A be a first-order formula. Denote by $A(c/x)$ a formula obtained from A by substituting every free occurrence of a variable x with a constant symbol c.

The root of a BI-tableau for $F(x)$ is

$$F(w/x) \cdot wZw' \cdot \neg F(w'/x).$$

To reduce the number of rules and to simplify proofs, we assume that both $F(w/x)$ and $\neg F(w'/x)$ are in the negation normal form (NNF), i.e., only atomic subformulas can be in the scope of negation. Note that this makes writing $\neg F(w'/x)$ an abuse of notation, but there should be no danger of confusion.

As suggested by the form of the root, we try to satisfy F at w and $\neg F$ at w' by building $\mathfrak{M}$ and $\mathfrak{M}'$ (with the roots w and w'), together with some bisimulation Z between them such that wZw'. So, formulas on the left-hand side of any node are about $\mathfrak{M}$, formulas on the right-hand side about $\mathfrak{M}'$, and formulas in the middle are about a relation, possibly a bisimulation, between them.

These are the rules:

- $\vee$-*rule*

$$\begin{array}{cc} A_1 \vee A_2 \cdot B \cdot C & A \cdot B \cdot C_1 \vee C_2 \\ \diagup \quad \diagdown & \diagup \quad \diagdown \\ A_1 \cdot\cdot \quad A_2 \cdot\cdot & \cdot\cdot C_1 \quad \cdot\cdot C_2 \end{array}$$

- $\wedge$-*rule*

$$\begin{array}{cc} \underline{A_1 \wedge A_2 \cdot B \cdot C} & \underline{A \cdot B \cdot C_1 \wedge C_2} \\ A_1 \cdot\cdot & \cdot\cdot C_1 \\ A_2 \cdot\cdot & \cdot\cdot C_2 \end{array}$$

- $\exists$-*rule*

$$\begin{array}{cc} \underline{\exists x A \cdot B \cdot C} & \underline{A \cdot B \cdot \exists x C} \\ A(a/x) \cdot\cdot & \cdot\cdot C(a'/x), \end{array}$$

  where a (resp. a') is a new constant symbol, i.e., it does not occur at any ancestor node.

- $\forall$-*rule*

$$\begin{array}{cc} \underline{\forall x A \cdot B \cdot C} & \underline{A \cdot B \cdot \forall x C} \\ A(a/x) \cdot\cdot & \cdot\cdot C(a'/x), \end{array}$$

  where a (resp. a') is any constant symbol that occurs on the left (resp. right) side of any ancestor or descendant node.

Each of the rules above is applied only once to each appropriate node, except for the $\forall$-rule, which is applied once for each constant symbol that occurs on the appropriate side of any node in a tableau.

The second group of rules have two premises each. Distinct applications of each of them may share one premise, but not both.

- (forth)-*rule*

$$\begin{array}{l} Rab \cdot\cdot \\ \underline{A \cdot aZa' \cdot C} \\ \quad \cdot bZb' \cdot \boxed{Ra'b'} \end{array}$$

(where b' is new)

- (back)-*rule*

$$\dfrac{\begin{array}{r}\cdot\cdot\, Ra'b'\\ \underline{A\cdot aZa'\cdot C}\end{array}}{\boxed{Rab}\cdot bZb'\cdot}$$

(where b is new)

- (at)-*rule*

$$\dfrac{\begin{array}{l}Pa\cdot\cdot\\ \underline{A\cdot aZa'\cdot C}\end{array}}{\cdot\cdot\,\boxed{Pa'}} \qquad \dfrac{\begin{array}{r}\cdot\cdot\, Pa'\\ \underline{A\cdot aZa'\cdot C}\end{array}}{\boxed{Pa}\cdot\cdot}$$

Atomic formulas appended by all of these rules are depicted boxed. To avoid introducing redundant nodes to a tableau, this group of rules does not make any subsequent use of nodes with boxed formulas.

We use the usual notions for tableaux. Since the present paper focuses on closed tableaux, let us emphasize only that we say that a branch of a BI-tableau is *closed* if some formula and its negation occur on the same side of some of its nodes, while a BI-tableau is *closed* if all of its branches are closed.

Theorem 2.1 ([6])**.** *The bisimulation invariance tableau calculus is sound and complete: a σ-formula $F(x)$ is bisimulation invariant if and only if there is a closed BI-tableau for $F(x)$.*

The procedure terminates in the case of a bisimulation invariant formula. Otherwise it might not terminate, since in some cases the only counterexamples are infinite. With an adjustment known from first-order logic tableaux (see [3]), it will always terminate if there exists a finite example. A counterexample can be read off an open branch. In this paper we will not use this adjustment, since only bisimulation invariant formulas are presently important.

3. Obtaining a formula from a closed tableau

The procedure has three stages: first the *interim* first-order formula is read off the tableau, then it is normalized in a way that enables the final stage – translation to a modal formula.

Suppose we have a closed BI-tableau for $F(x)$. Let $a_0 = w$ and let $a_1, a_2, \ldots$ be the other constant symbols that occur in formulas on the left. We read the tableau starting from the root until one of the following cases occurs and construct an interim first-order formula F' as follows.

- The branch closes. If this is due to an occurrence of a boxed formula of the form $\boxed{Pa_i}$ and $\neg Pa_i$ on the left side, put $F' = \neg Px_i$. If such a contradiction occurs on the right side, put $F' = Px_i$.

  Similarly, if we have $\boxed{Ra_ia_j}$ and $\neg Ra_ia_j$ on the left, put $F' = \neg Rx_ix_j$, and if we have such a contradiction on the right side, put $F' = Rx_ix_j$.

  If the branch closes due to a contradiction of some formulas that are not boxed, put $F' = \bot$ if this contradiction occurs on the left side, but if it occurs on the right side, put $F' = \top$.

- A branching occurs. If this is due to an application of the $\vee$-rule on the left side, then put $F' = F'_1 \vee F'_2$, and otherwise, if it is due to an application of the $\vee$-rule on the right side, put $F' = F'_1 \wedge F'_2$, where F'_1 and F'_2 are obtained from the respective branches. Proceed inductively.

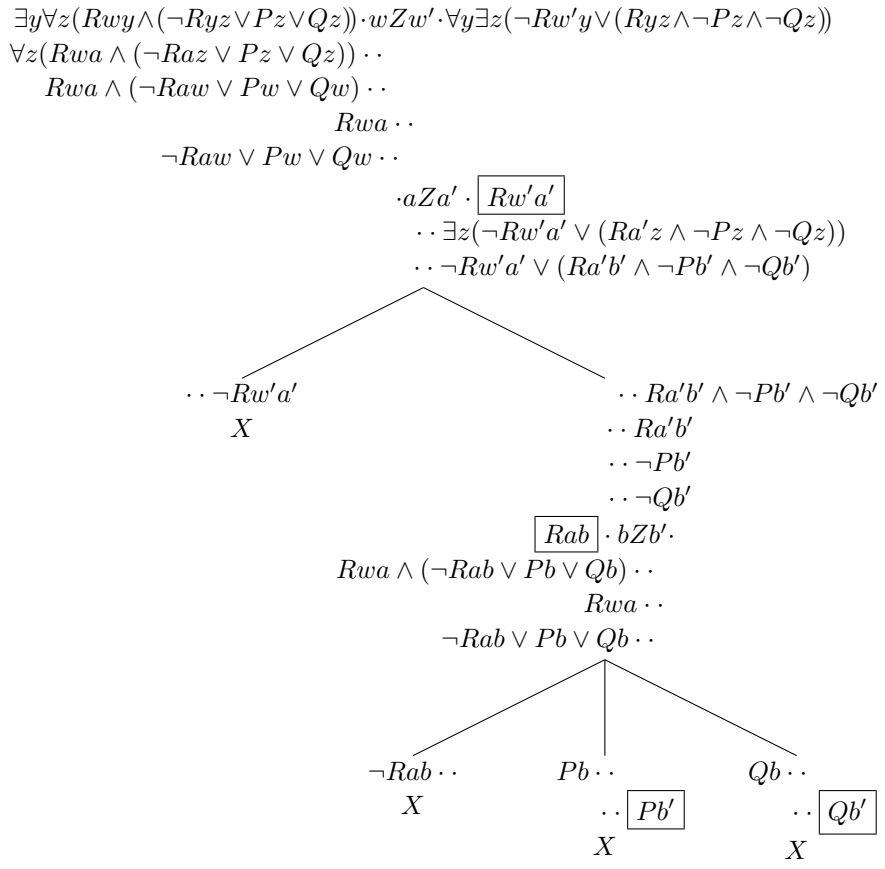
- An application of the (forth)-rule occurs, involving a formula of the form Ra_ja_k on the left-hand side. In this case put $F' = \exists x_k F'_1$ and obtain F'_1 from the rest of the current branch inductively.

- An application of the (back)-rule occurs, which appends a formula of the form $\boxed{Ra_ja_k}$ on the left-hand side. Put $F' = \forall x_k F'_1$ and obtain F'_1 from the rest of the current branch inductively.

In examples with only a few constant symbols, to simplify notation, we use constant symbols w, a, b and variables x, y, z.

Consider some examples.

Example 1. Let $F(x) = \exists y \forall z (Rxy \wedge (\neg Ryz \vee Pz \vee Qz))$.



Reading form the root, the first point relevant for the construction of the interim formula is the application of the (forth)-rule at the 6th node. We put $F' = \exists y F'_1$ and then obtain F'_1 from this point onwards.

The next relevant point is the first branching, which occurs on the right side, so at this point we put $F'_1 = F'_{11} \wedge F'_{12}$, and then obtain F'_{11} and F'_{12} from the respective branches. So, at this point we have $F' = \exists y(F'_{11} \wedge F'_{12})$.

The left branch immediately closes in the way for which the procedure tells us to put $F'_{11} = Rxy$. The next relevant event on the right branch is the application of the (back)-rule, resulting in $F'_{12} = \forall z F'_2$. Thus, $F' = \exists y(Rxy \wedge \forall z F'_2)$, with F'_2 due to be obtained from the rest of the right branch.

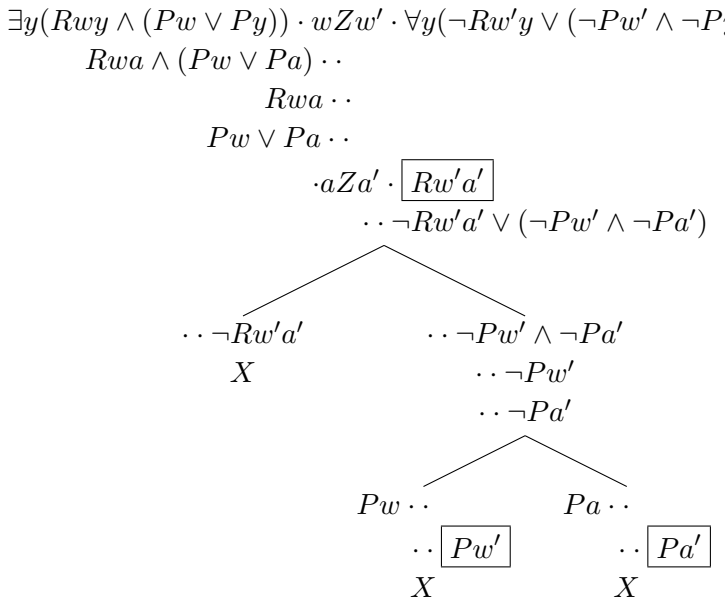
The remaining step is the branching below, which occurs on the left side, so we have $F'_2 = F'_{21} \vee F'_{22} \vee F'_{23}$.

All new branches immediately close, resulting in $F'_2 = \neg Ryz \vee Pz \vee Qz$.

Thus we have $F' = \exists y(Rxy \wedge \forall z(\neg Ryz \vee Pz \vee Qz))$, which is the standard translation of a modal formula $\Diamond\Box(p \vee q)$. Clearly, F' is equivalent to F.

The interim formula enables an easier translation to a modal formula. In the previous example the interim formula is in fact a standard translation. This is not always the case.

Example 2. Let $F(x) = \exists y(Rxy \wedge (Px \vee Py))$.



The steps in building F' are the following:

- the (forth)-rule is applied: $F' = \exists x F'_1$
- branching on the right: $F' = \exists x(F'_{11} \wedge F'_{12})$
- left branch closes, while the right branch has a further left-hand branching: $F' = \exists x(Rxy \wedge (F'_{21} \vee F'_{22}))$
- remaining branches close: $F'_{21} = Px$, $F'_{22} = Py$

Thus we have $F' = \exists y(Rxy \wedge (Px \vee Py))$, which is actually equal to F. This is not exactly the standard translation of some modal formula. However, we have the following sequence of formulas clearly equivalent to F':

$\exists y((Rxy \wedge Px) \vee (Rxy \wedge Py))$
$\exists y(Rxy \wedge Px) \vee \exists y(Rxy \wedge Py)$

$(\exists y Rxy \wedge Px) \vee \exists y(Rxy \wedge Py)$

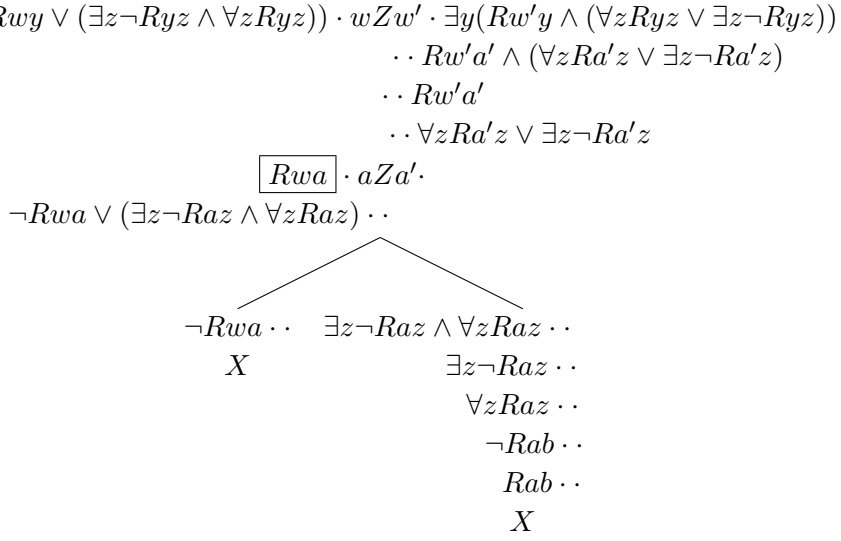
Now, the last formula is the standard translation of $(\Diamond\top \wedge p) \vee \Diamond p$. The key to obtain this was to remove Px from the scope of $\exists y$. Indeed, the above sequence of formulas is an example of the well-known procedure of anti-prenexing or miniscoping (see, e.g., [5]).

As in the previous example, we can always use basic equivalences of first-order formulas (cf. [7], p. 117) to obtain the standard translation of some modal formula from an interim formula.

Our procedure, of course, needs an algorithm which determines whether a first-order formula is the standard translation of a modal formula. To check this, we can traverse the formula's syntax tree and check if (1) the only free variable contained in a subformula is the variable introduced by the closest quantified formula ancestor (or w if no such variable exists) and (2) if the formula's operator is a logical connective or has the form $\forall\beta(R\alpha\beta \rightarrow \dots)$ or $\exists\beta(R\alpha\beta \wedge \dots)$, with α not having been bound already and β being the variable introduced by the closest quantified formula ancestor (or w if no such variable exists).

Consider another example, which illustrates how interim formula can be significantly simpler then the initial formula, which was not the case in the previous examples.

Example 3. Let $F(x) = \forall y(\neg Rxy \vee (\exists z \neg Ryz \wedge \forall z Ryz))$.



The steps in building F' are the following:

- the (back)-rule is applied: $F' = \forall y F'_1$
- branching on the left: $F' \forall y(F_{11} \vee F_{12}$
- branches close: $F_{11} = \neg Rxy$, $F_{12} = \bot$

Thus we have $F' = \forall y(\neg Rxy \vee \bot) = ST_x(\Box\bot)$.

Theorem 3.1. *Assume we have a closed BI-tableau for $F(x)$. Then the interim formula F' is equivalent to F. Furthermore, F' can be effectively rewritten to an equivalent formula that is the standard translation of some modal formula.*

Proof. Consider any node of the given BI-tableau. Let G' be the subformula of F' obtained by the procedure starting from that node. In what follows, we will treat two conclusions of the $\wedge$-rule as belonging to one node (this is the only case in which we will have more than one formula on the same side). We claim that G' is equivalent to one of the following:

(1) $Q_1 x_{i+1} Q_2 x_{i+2} \ldots G(x/w, x_1/a_1, x_2/a_2, \ldots)$, where G is a formula or the conjunction of all formulas which occur on the left side or boxed on the right side of some preceding nodes (the rest of the proof below makes it clear which ones), with at least one of these nodes being between the previous branching (if there is one) and the current node, including the current node, or

(2) $Q_1 x_{i+1} Q_2 x_{i+2} \ldots \neg G(x/w', x_1/a'_1, x_2/a'_2, \ldots)$, with an analogous condition to that in (1), but with the left and the right side swapped,

where each of $Q_1, Q_2, \ldots$ is $\forall$ or $\exists$ and $x_{i+1}, x_{i+2}, \ldots$ are variables corresponding to constant symbols $a_{i+1}, a_{i+2}, \ldots$ or $a'_{i+1}, a'_{i+2}, \ldots$ occurring in G such that $a_{i+1} Z a'_{i+1}$, $a_{i+2} Z a'_{i+2}, \ldots$ do not occur in the tableau prior to the current node.

Note that, applied to the root, this claim becomes the desired claim that F' is equivalent to F. Indeed, at the root G as described above can only be F itself, with no additional quantifiers in the prefix, since at the root there are no constant symbols such as those described above.

We prove the claim by induction on the distance of the current node to the farthest leaf.

In the base case (where nodes are leafs), G' may be of the form Rx_ix_j, $\neg Rx_ix_j$, Px_i, $\neg Px_i$, $\top$ or $\bot$.

- $G' = Rx_ix_j$. This case occurs only if we have a contradiction between $\boxed{Ra'_ia'_j}$ and $\neg Ra'_ia'_j$ on the right side. So, $G = Ra'_ia'_j$ occurs boxed on the right side, while $G(x_i/a'_i, x_j/a'_j) = Rx_ix_j$ is equivalent (in fact, equal) to G'.

- $G' = \neg Rx_ix_j$ results from the occurrence of $\boxed{Ra_ia_j}$ and $G = \neg Ra_ia_j$ on the left side, so $G(x_i/a_i, x_j/a_j)$ is equivalent to (in fact, equals) $\neg Rx_ix_j = G'$.

- $G' = Px_i$ implies that $\boxed{Pa'_i}$ and $\neg Pa'_i$ occur on the right side, so for $G = Pa'_i$ we have that G occurs boxed on the right side and $G(x_i/a'_i) = Px_i = G'$.

- $G' = \neg Px_i$ implies that $\boxed{Pa_i}$ and $G = \neg Pa_i$ occur on the left side, so we have $G(x_i/a_i) = \neg Px_i = G'$.

- $G' = \top$ implies that some G and $\neg G$ occur on the right side on the path. Clearly, $\neg(G \wedge \neg G) \equiv \neg G \vee G$ is equivalent to G'. This remains to be the case if we prefix $\neg G \vee G$ with quantifiers as stated in (1) and (2).

- $G' = \bot$ implies that some G and $\neg G$ occur on the left side. Clearly, $G \wedge \neg G$ is equivalent to $G' = \bot$. Again, this holds regardless of any quantifiers prefixed to $G \wedge \neg G$ (using, of course, the non-empty domain assumption, which we do).

Note that in all the base cases, one of the nodes containing the relevant formulas must occur under the previous branching, since branches are immediately closed after a contradiction occurs.

Inductive step has several cases, depending on the next rule applied in the tableau, starting from the current node.

- If the next rule applied in the tableau is the $\forall$- or $\exists$-rule, which both have no immediate effect on F', the subformula of F' obtained starting after the application of this rule is again G'. By the induction hypothesis, (1) or (2) applies to G', with G (or one of its conjuncts)

occurring either prior to the current node, in which case the claim is proved, or exactly at the node succeeding the application of the $\forall$-rule. In the latter case, the claim also holds, since (this conjunct of) G should be prefixed by $Q_j x_j$, where x_j is the variable involved in this application of the $\forall$- or $\exists$-rule, and this prefixed formula occurs at the current node.

- As in the previous case, an application of the $\wedge$-rule also has no immediate effect on F', i.e., the subformula of F' obtained starting after the application of the $\wedge$-rule is still G'. By the induction hypothesis, G' is implied by a formula of the form described in (1) or (2), where G or one of its conjuncts occur either above the current node, in which case the claim holds, or immediately after the application of the $\wedge$-rule. But, since we treat conclusions of the $\wedge$-rule as one node, taking the conjunction of formulas from that node is the same as taking the formula to which the $\wedge$-rule was applied.

- In the case of the left $\vee$-rule, the current node is of the form $G_1 \vee G_2 \cdot\cdot$, while $G' = G'_1 \vee G'_2$, where G'_1 and G'_2 are subformulas obtained starting from the roots of two branches. By the induction hypothesis, (1) applies to G'_1 and to G'_2, where G or one of its conjuncts is G_1 and G_2, respectively, since these are the only nodes after the previous branching. So clearly, $G_1 \vee G_2$ will work as G (or instead of a conjunct of G) for G'.

- In the case of the right $\vee$-rule, the current node is of the form $\cdot\cdot G_1 \vee G_2$ and $G' = G'_1 \wedge G'_2$, where G'_1 and G'_2 are obtained starting from the roots of branches. By the induction hypothesis, (2) applies to G'_1 and to G'_2, where G or one of its conjuncts is G_1 and G_2, respectively, as in the previous case. It is easy to see that $G_1 \vee G_2$ will work as G (or instead of a conjunct of G) for G'. For example, if G'_1 is equivalent to $\neg G_1$ and G'_2 is equivalent to $\neg G_2$, then G' is equivalent to $\neg(G_1 \vee G_2)$. It is easy to see that this still works with additional conjuncts or quantifiers which may occur as described in (1) and (2).

- The (at)-rule does not change the subformula obtained thereafter, so it is G'. Also, the new node contains only one formula which is a boxed version of a formula which already occurred above on the other side, so the induction hypothesis implies the claim.

- The (forth)-rule has the same property as the (at)-rule concerning the content of the new node, but the subformula obtained after its application is G'_1, where $G' = \exists x_k G'_1$. We apply the induction hypothesis to G_1. Then the same G from (1) and (2) which corresponds to G'_1 can be used for G', prefixed by the additional quantifier $\exists x_k$. The case of the (back)-rule is proved analogously.

It remains to show that a modal correspondent is effectively obtainable from F'. Clearly, if F' is already the standard translation of some modal formula (or if F is, in which case we do not need to use BI-tableau in the first place), it is easy to reverse the translation to obtain this modal formula. If not, we use anti-prenexing or miniscoping (see, e.g., [5]) to obtain an equivalent formula F''. We prove by induction on the complexity of F'' that F'' is the standard translation of some modal formula.

The base case is trivial, since R cannot occur in a quantifier-free F'' and other cases are obviously standard translations of modal formulas. The Boolean cases in the inductive step are also trivial, since standard translation commutes with Boolean connectives. So, it remains to consider the cases where F'' is of the form $\exists y G''$ or $\forall y G''$.

In the first case, the occurrence of the existential quantifier implies that the (forth)-rule was applied in the tableau, involving Rwa on the left side. This implies that Rxy is a subformula of F, but then $\neg Rxy$ is a subformula of NNF of $\neg F$. Consequently, one of the branches closes due to a contradiction between $\boxed{Rw'a'}$ and $\neg Rw'a'$ on the right side, making Rxy also a subformula of F''. We can assume without loss of generality that F'' is of the form $\exists y(Rxy \wedge H'')$ (or just $\exists y Rxy$, in which case F'' is the standard translation of $\Diamond\top$), since anti-prenexing moves any other quantifier further inside F'' and also distributes existential quantifiers through disjunctions. Furthermore, it is easy to see that in H'' only y occurs freely, thus we can use the induction hypothesis to show that H'' equals $ST_y(\psi)$ for some ψ. Hence, $F'' = ST_x(\Diamond\psi)$. The universal quantifier case is proved analogously. □

Remark 1. Note that the enrichment of modal logic with hybrid operators makes all first-order formulas translatable to the hybrid language, using the *hybrid translation*, an analogue of the standard translation. See [4] for details on translations between hybrid and first-order languages. This means that the present paper also points out to a relation between

modal and hybrid languages by a two-step translation via the first-order logic.

4. Implementation

Let us briefly comment on the implementation details.[1] The algorithm is implemented in JavaScript for easier portability. We use the open-source parser *nearley.js*, a high-quality and robust implementation of the *Earley* parsing algorithm.[2] Our grammar supports most of the usual ASCII replacements for logical symbols.[3]

Implementation closely follows the algorithm as described earlier, with minor modifications. In particular, it supports other truth-functional connectives beside $\wedge$ and $\vee$. Unlike in the paper version, the implementation is based on a variant of the calculus in which the inference rules operate on signed formulas. In other words, the implementation keeps track of whether a formula is "true" ($\top$) or "false" ($\bot$). For example, a $p \to q\ \top$ branch will produce $p\ \bot$ and $q\ \top$ as the offspring. In particular, in the implementation, the connective $\neg$ is never appended to a formula in a child branch. This way, successors of a node may contain only subformulas of formulas that are already present.

To limit time and space, both of which rise sharply (this is a consequence of the combinatorial explosion created by the presence of the $\exists$-rule and $\forall$-rule), executions are limited to a configurable number of steps. For most formulas, it is best to experiment with raising this number in small increments. The time and space consumed depends very much on a formula, but for the majority of formulas, reasonable values are below 100. A *step* consists of the following actions.

1. Resolving propositional connectives. In this step, all tableau rules concerning logical connectives are applied iteratively as long as they can be applied. Any signed formula of the form $\forall \ldots\ \bot$ (resp. $\exists \ldots\ \bot$) is solved by adding a child branch containing a formula $\exists \ldots\ \top$ (resp. $\forall \ldots\ \top$).

[1] Available on `https://github.com/luka-mikec/inverse-standard-translation`.

[2] See `https://nearley.js.org`.

[3] See `https://github.com/luka-mikec/inverse-standard-translation/blob/master/g.ne` for details.

2. Quantified formulas. In this step, a single (if any) $\forall$-rule or $\exists$-rule is applied. Existential formulas are always preferred to universal formulas. If there are multiple available universal formulas, the preferred formula is the one that has been instantiated earliest. Formulas inherit the instantiation time from their parents, thus ensuring that every universal formula will get instantiated at some point. The instantiation constant in the case of universal formulas is lexicographically the smallest constant introduced in the tableau so far.

3. Resolving propositional connectives. The repetition of this action is not needed in order for the algorithm to be correct, but it enables the algorithm to quickly find a contradiction if one exists on the propositional level.

4. Bisimulation rules. The tableau is traversed in order to find a pair of premises where the (forth)-rule, the (back)-rule or the (at)-rule can be applied. The pair must be unused previously. The first such pair is selected and the rule is applied.

The steps are always applied on the whole tableau and not recursively, i.e., an infinite branch will not use up all the available steps.

After obtaining the interim formula with the procedure above, we "flatten" disjunctions and conjunctions into n-ary expressions without nesting. This makes it easier to detect and remove repeated subformulas. The next step is to reposition the quantifiers in order to get the shape of the *standard translations*. At this point we expect that, for example, a $\forall qF$ subformula is actually $\forall q \bigvee F_i$ where at least one F_i is of the form $\neg Rq'q$ for some q'. We continue this process recursively, i.e., the next step is to find a translation of $\bigvee_{j\neq i} F_j$. Once this process is complete, we have a formula which is a standard translation of some modal formula. This formula is converted to a modal formula in the straightforward manner.

Below we provide some input examples which we used to test the implementation.

Note that the second to last formula could have been $\Diamond\neg P\vee\Diamond P$, or even just $\Diamond\top$. The last formula could have been simpler, too: $\Box\neg P\vee\Box P$. In general, the algorithm does not produce the shortest corresponding modal formula, but rather *some* corresponding modal formula.

Input	Output
$\forall x \neg Rcx$	$\Box\bot$
$\exists x Px$	$\Diamond P$
$\exists x(Rcx \wedge (\exists y(Rxy \wedge \exists y Py)))$	$\Diamond\Diamond P$
$\forall x(Rcx \rightarrow (\forall y(Rxy \rightarrow Py)))$	$\Box\Box P$
$\forall x(Rcx \rightarrow Px) \rightarrow Pc$	$\Diamond\neg P \vee P$
$\forall x(Rcx \rightarrow Px) \rightarrow \exists y(Rcy \wedge Py)$	$\Diamond\neg P \vee \Diamond\top$
$\exists y(Rcy \wedge Py) \rightarrow \forall x(Rcx \rightarrow Px)$	$\Box\neg P \vee (\Box P \vee \Box\bot)$

More complex formulas, such as the standard translation of $\Box\Box P \rightarrow \Box P$,

$$\forall x(Rcx \rightarrow \forall y(Rxy \rightarrow Py)) \rightarrow \forall x(Rcx \rightarrow Px),$$

are too complex to be processed by this implementation in reasonable time.

Acknowledgment

We are very grateful to the referee for valuable suggestions, in particular for the suggestion to enrich examples with descriptions of steps in building interim formulas, for pointing out a link between modal, first-order and hybrid logic and last but not least, for patiently annotating numerous small corrections.

References

[1] J. van Benthem, Exploring Logical Dynamics, Studies in Logic, Language and Information, CSLI Publications & FoLLI, Stanford, 1996.

[2] P. Blackburn, M. de Rijke, Y. Venema, Modal Logic, Cambridge University Press, 2001.

[3] G. Boolos, Trees and Finite Satisfiability: Proof of a Conjecture of Burgess, *Notre Dame Journal of Formal Logic* **25** (1984), 193–197.

[4] T. Braüner, Hybrid Logic and its Proof-Theory, Springer, 2011.

[5] J. Harrison, Handbook of Practical Logic and Automated Reasoning, Cambridge University Press, 2009.

[6] T. Perkov, Tableau-based bisimulation invariance testing, *Reports on Mathematical Logic* **48** (2013), 101–115.

[7] R. M. Smullyan, First-Order Logic, Springer-Verlag, 1968.

Tin Perkov

Chair of Mathematics and Statistics, Faculty of Teacher Education, University of Zagreb,
Savska c. 77, HR-10000 Zagreb, Croatia,

tin.perkov@ufzg.hr

Luka Mikec

Department of Mathematics, Faculty of Science, University of Zagreb,
Bijenička c. 30, HR-10000 Zagreb, Croatia

luka.mikec@math.hr