Terminating modal tableaux with simple completeness proof

Olivier Gasquet, Andreas Herzig, Mohamad Sahade

ABSTRACT. In this paper we study formal tools for the definition of tableaux systems for modal logics. We begin by some preliminaries about graphs. Then we present graph rewriting rules together with control constructs for defining strategies of application of these rules. We establish fair strategies that are sound, complete and terminating. These results cover almost all basic modal logics such as K, T, S4, KB4, K+Confluence etc., and can be generalized for logics such LTL and PDL. Our framework provides the theoretical basis for our generic tableau theorem prover LoTREC.

Keywords: modal logics, tableau methods, automatic proof.

1 Introduction

Basic modal logics are normal modal logics defined by any combination of axioms D, T, B, 4, 5 [4]. For all these logics the complexity of satisfiability is in PSPACE, and research (in particular in the field of description logics) has focussed on tableau decision procedures for them that stay within that boundary. For all these logics one can find sound, complete and terminating tableau algorithms in the standard textbooks [5, 7], and implemented tableau theorem provers such as LogicWorkBench (LWB) [8] are readily available.

This contrasts with the situation for more complex modal logics that are used to reason about mental states of artificial agents, about knowledge and communication, or norms and regulations. In these domains we have to deal with BDI logics (that is, logics capturing the notions of belief, desires and intentions), logics of knowledge, common knowledge and time, logics of action and obligation, etc. Such logics are not as 'stable' as the traditional modal logics we find in textbooks: for example, it often happens that one would like to slightly modify the interplay between the modalities (preservation of beliefs through actions, influence of actions on obligations, interplay between belief and knowledge, etc.), or to add a new modality to an existing logic (add actions or time to a logic of belief, add goals to a logic of belief and action, add obligations to BDI logics, etc.) The variety and complexity of these logics make that in practice we do not always have at our disposal tableau algorithms that enjoy soundness, completeness and termination.

The contributions of our paper are the following:

1. We provide a general formal framework for tableau systems, within which virtually all tableau algorithms for modal logics can be implemented. This framework is based on rooted directed acyclic graphs (RDAGs) in order to stay close to possible worlds models, and its basic mechanism is graph rewriting. It provides a simple language to define tableau rules and a simple language to define strategies. We formally define the semantics of the usually rather implicit or purely operational notions of rule and strategy.

2. Then we give simple and uniform completeness and soundness proof for a large family of modal logics.

3. We state general termination theorems for two classes of strategies. These classes contain strategies for the modal logics K, T, S4, S5, KB4, and even LTL and PDL.

This paper presents the theoretical elements which are the basis of the generic tableau prover LoTREC that we develop in the LILaC group of IRIT. LoTREC is available at: *http://www.irit.fr/LILAC/Lotrec*

In particular there is a version of LoTREC that is executable via the web. A system description can be found in [6].

2 Modal logics and relational properties

The modal logics we consider are all obtained by extending the basic modal logic K by one or several of the well-known axioms T, B, 4, 5, D, and C (axiom of confluence: $\Diamond\Box p \rightarrow \Box\Diamond p$). Thus KDC4 denotes the modal logic obtained by adding the axioms D, C and 4 to the basic system K.

With each of these axioms there is a associated relational property of the accessibility relation of the Kripke models. The axioms: T:$\Box p \rightarrow p$, 4: $\Box p \rightarrow \Box\Box p$, B:$\Diamond\Box p \rightarrow p$, 5:$\Diamond\Box p \rightarrow \Box p$, D:$\Box p \rightarrow \Diamond p$ and C:$\Diamond\Box p \rightarrow \Box\Diamond p$ correspond respectively to: reflexivity (*Ref*), transitivity (*Tr*), symmetry (*Sym*), euclideanity (*Eucl*), seriality (*Ser*), confluence (*Conf*).

As a consequence of Sahlqvist's theorem [11], a modal logic based on K plus any combination of these axioms is characterized by the class of Kripke models whose accessibility relation satisfies the corresponding properties. Thus, KD4 is characterized by Kripke models where the accessibility relation is both serial and transitive.

Let group 1 be {*Ref*, *Tr*, *Sym*, *Eucl*} and group 2 be {*Ser*, *Conf* }.

DEFINITION 1. Given a set ρ of relational properties among group 1 and 2, a ρ-*model* is a Kripke model whose accessibility relation satisfies ρ. A formula is ρ-satisfiable iff it is satisfiable in a ρ-model. It is ρ-valid iff it is valid in the class of all ρ-models, which will be noted $\models_\rho A$. Thus A is a theorem of a system denoted by a set ρ of properties iff it is ρ-valid.

3 Rooted Directed Graph

In the paper [3] we have advocated that tableaux systems should work on rooted directed acyclic graphs (RDAGs). Such a choice not only has the advantage of generality: as RDAGs are closer to Kripke models than sequent and trees, it also makes the way from an open tableau to a countermodel shorter. Here we pursue that approach.

DEFINITION 2. A labelled ρ-directed graph is a triple $\langle \mathcal{N}, \Sigma, \textsc{for} \rangle$ where:

- $\langle \mathcal{N}, \Sigma \rangle$ is a directed graph: $\mathcal{N}$ is some set and $\Sigma \subseteq \mathcal{N} \times \mathcal{N}$
- $\langle \mathcal{N}, \Sigma \rangle$ satisfies all properties of ρ;
- FOR is a function that associates a set of formulae with each of the nodes.

A labelled *rooted directed graph* ρ-(RGRAPH) is a couple $\langle G, root \rangle$ where $G = \langle \mathcal{N}, \Sigma, \textsc{for} \rangle$ is a labelled directed graph, and $root \in \mathcal{N}$ is a root of G, that can access every other node in the transitive closure of Σ.

The elements $\langle x, x' \rangle$ of Σ are also written $x\Sigma x'$. $x\Sigma x'$ means that x' is accessible from x via the relation Σ. For $\langle x, A \rangle \in \textsc{for}$ we sometimes also write $A \in \textsc{for}(x)$. $A \in \textsc{for}(x)$ means that the formula A is associated with node x in the graph. Graphs are denoted by $G, g, g', \ldots$. Small letters will be used for subgraphs.

DEFINITION 3. (RDAG) A labelled *rooted directed acyclic graph*(ρ-RDAG) is a labelled RGRAPH where Σ contains no cycles.

4 Closure of RGRAPH

We define the following closure operation on RGRAPH:

DEFINITION 4. Let $\langle G, root \rangle$ be an RGRAPH over a set and ρ a set of relational properties from group 1; the ρ-closure of Σ (denoted by Σ^ρ) is the least RGRAPH that contains Σ and which satisfies every property of ρ.

This ρ-closure always exists for properties among $\{Ref, Tr, Sym, Eucl\}$. A very important point is that for properties of group 1, the closure can be expressed in terms of the initial RGRAPH. For example, the transitive closure of an RGRAPH $\langle G, root \rangle$ is defined by: $(x, y) \in \Sigma^{Tr}$ iff $\exists n \geq 1$ such that $(x, y) \in \Sigma^n$. Note that we do not consider here properties of group 2: is not so easy to define closure under a property of group 2. This is the reason why they will be handled in a different way: as we will see later no propagation rule can simulate them.

LEMMA 5 (Relational Closure Lemma).
Let Σ be an RGRAPH over a set $\mathcal{N}$ of nodes:

- $(x, y) \in \Sigma^{Ref}$ *iff* $(x, y) \in \Sigma$ *or* $x = y$.
- $(x, y) \in \Sigma^{Sym}$ *iff* $(x, y) \in \Sigma$ *or* $(y, x) \in \Sigma$.
- $(x, y) \in \Sigma^{Tr}$ *iff* $\exists n \geq 1$ *such that* $(x, y) \in \Sigma^n$.
- $(x, y) \in \Sigma^{Eucl}$ *iff* $(x, y) \in \Sigma$ *or* $\exists u \in \mathcal{N}$ $\exists n \geq 1$ $\exists m \geq 1$ *such that* $(u, x) \in \Sigma^n$ *and* $(u, y) \in \Sigma^m$.

- $(x, y) \in \Sigma^{Ref,Sym}$ *iff* $(x, y) \in \Sigma$ *or* $x = y$ *or* $(y, x) \in \Sigma$.
- $(x, y) \in \Sigma^{Ref,Tr}$ *iff* $\exists n \geq 0$ *such that* $(x, y) \in \Sigma^n$.
- $(x, y) \in \Sigma^{Ref,Eucl}$ *iff* $\exists n \geq 0\ \exists x_0 = x, \ldots x_i, x_{i+1}, \ldots, x_n = y$: $(x_i, x_{i+1}) \in \Sigma$ *or* $(x_{i+1}, x_i) \in \Sigma$.
- $(x, y) \in \Sigma^{Sym,Tr}$ *iff* $\exists n \geq 1\ \exists x_0 = x, \ldots x_i, x_{i+1}, \ldots, x_n$: $(x_i, x_{i+1}) \in \Sigma$ *or* $(x_{i+1}, x_i) \in \Sigma$.
- $(x, y) \in \Sigma^{Tr,Eucl}$ *iff* $\exists u \in \mathcal{N}\ \exists n \geq 0\ \exists m \geq 1$ *such that* $(u, x) \in \Sigma^n$ *and* $(u, y) \in \Sigma^m$.

LEMMA 6. *The remaining cases are reducible to those of the previous lemma:*

- $\Sigma^{Sym,Eucl} = \Sigma^{Sym,Tr,Eucl} = \Sigma^{Sym,Tr}$;
- $\Sigma^{Ref,Sym,Tr} = \Sigma^{Ref,Tr,Eucl} = \Sigma^{Ref,Sym,Eucl} = \Sigma^{Ref,Sym,Tr,Eucl} = \Sigma^{Ref,Eucl}$.

The above lemmas will be a powerful tool for proving completeness: it will allow to define a model for a formula from an open tableau. But this is not the whole story. As we previously said, some properties are handled structurally; roughly speaking seriality and confluence are treated by generating the RDAG underlying of the tableau.

LEMMA 7 (Structural Lemma). *Let ρ_2 be a subset of group 2, ρ_1 a subset of group 1 and let Σ be a ρ_2-RGRAPH over a set $\mathcal{N}$ of nodes. Then Σ^{ρ_1} is also a ρ_2-RGRAPH, and hence is a $(\rho_1 \cup \rho_2)$-RGRAPH.*

For the proof of the above three lemmas see ([3]).

5 A simple graph-rewriting language $\mathcal{RL}$

As we would like to handle complex modal logics, the tableau rules that we need must be able to perform at least the following operations on graphs: add a formula A to a node x, add a link with some label R between two nodes x, x', create a new node x, duplicate the graph (to take into account disjunctions), mark and unmark a node x by an expression E, mark and unmark a formula A in a node x by an expression E.

Note that links have to be labelled in order to be able to handle multiple modalities. Note also that marking is not needed for the simplest modal logics (the basic modal logics defined by any combination of axioms D, T, B, as well as $K45, KD45$ and $S5$).

Such rewriting steps are executed under some conditions. We need at least the following ones: node x contains a formula A (or not), node x is linked to node x' by a link having label R (or not), node x is an ancestor of node x', node x is marked by an expression E (or not).

Note that only the first two conditions are necessary for the simplest modal logics.

We here choose a minimal rule language that is restricted to exactly the above conditions and actions. It is also the rule language of LoTREC, with self-explaining conditions such as: if a node x contains a formula A, or does not contain a formula A, …and actions such as: add an expression E to a node x, link two nodes x, x'…

DEFINITION 8. (pattern) A pattern is a connected graph.

DEFINITION 9. A *tableau rule* r is a couple $(p, \{g_1, \ldots, g_n\})$ where p is a pattern (the pattern on which the rule can be applied) and $g_i, 1 \leq i \leq n$ are the graphs resulting from the application of the rule on patterns which match p.

The rule r is *analytic* if every formula appearing in $g_i, 1 \leq i \leq n$ is a subformula of some formula appearing in p. r is strictly analytic if every formula appearing in $g_i, 1 \leq i \leq n$ is a strict subformula of some formula appearing in p.

Note that in the sequel I use the terms: *conditions part* of a rule to represent the pattern p and *action part* of a rule to represent the modifications done which lead to the new graphs $\{g_1, \ldots, g_n\}$.

Application of a rule r to a pattern p_0 is defined by:

$$(p, \{g_1, \ldots, g_n\})(p_0) = \begin{cases} \{\nu(g_1), \ldots, \nu(g_n)\} & \text{if } \exists\nu \text{ such that } \nu(p) = p_0 \\ \perp & \text{otherwise} \end{cases}$$

Note that ν is an isomorphism on the structure of the graph, but it is a homomorphism on the set of formulas.

We extend the partial function r to a total function (application of r on a graph) by stipulating one of the following definitions:

$$\begin{array}{lll} 1- & r(G) = \{G \cup (\bigcup_{p \subseteq G} \delta) : \delta \in r(p)\} & \text{where } \forall p, r(p) \text{ is applicable} \\ 2- & r(G) = \{(G \cup (\bigcup_{\substack{p \subseteq G \\ r(p) \neq \perp}} \delta)) : \delta \in r(p)\} & \end{array}$$

Note that a graph is a set of patterns (and may be only one pattern).

Application of a rule r to a set of graphs Γ is defined by:

$$r(\Gamma) = \{\Gamma \cup (\bigcup_{G \subseteq \Gamma} r(G))$$

As every rule $(p, \{g_1, \ldots, g_n\})$ is monotonic (in the sense that that $p \subseteq g_i, 1 \leq i \leq n$), we only give the new expressions of $g_i, 1 \leq i \leq n$.

REMARK 10. p and $g_i, 1 \leq i \leq n$ may contain node variables and formula variables (that are instantiated via ν when the rule is applied). We can think of such rules as the (infinite) set of their instances.

REMARK 11. Of course, $r(p)$ may contain new nodes and/or edges that do not belong to p and that have to be named. We suppose that such new nodes will be denoted by ground first-order terms: a Skolem function denoted by the name of the rule (and possibly with an integer subscript if the rule introduces more that one new node) and with elements of g as arguments. As an example, if some node x contains a formula $\Diamond A$ then applying the rule for $\Diamond$ will introduce the new node $\Diamond(x, A)$.

6 Tableau rules

In order to define a tableau calculus for a modal logic denoted by $KA_1, \ldots, A_n$ corresponding to a set of properties ρ_1 form group 1 and a set of properties ρ_2 of group 2, we must associate a set of rules with it. All the tableaux calculi we are going to define contain: the classical rules, `createSuccessor` and `propagateFormulas` rules which will be written in $\mathcal{RL}$ as
$(\langle\{n_0\}, \emptyset, \{(n_0, \{\neg\Box A\})\}\rangle, \{\langle\emptyset, \{(n_0, n)\}, \{(n, \{\neg A\})\}\rangle\})$,
$(\langle\{n_0, n_1\}, \{(n_0, n_1)\}, \{(n_0, \{\Box A\})\}\rangle, \{\langle\emptyset, \emptyset, \{(n_1, \{A\})\}\rangle\})$ respectively (see also appendix B). As these rules are common to all tableaux calculi, we will henceforth omit them. More over we have some structural and propagation rules.

A tableau calculus for a logic defined by $(\rho_1 \cup \rho_2)$ is obtained by taking (in addition to classical rules, `createSuccessor` and `propagateFormulas` rules) the rules corresponding to properties of $(\rho_1 \cup \rho_2)$; this correspondence is given in the figure below.

	Properties	Rules
Group 1	*Ref*	`propagateNec2ActualWorld`
	Sym	`addFormula2Parent`
	Tr	`propagateNec2Child`
	Eucl	`copyNecIfLinked, copyNec2Parent, copyNec2Brothers`
Group 2	*Ser*	`addPosTrue`
	Conf	`confluence`

In the case of logics which contain the relational properties: *Eucl* or *Tr*, to stop the possibly infinite computation we perform an inclusion test called loop test. It tests if a node x is included in some ancestor. In this case we don't create any successor for x and we stop the computation on the tableau which contain x.

REMARKS 12.

It is worth noticing that our three rules handling euclideanity are reminiscent of the three sequent rules used in [2]. This connection between a tableau approach and a sequent approach probably deserves to be investigated further.

DEFINITION 13. A $(\rho_1 \cup \rho_2)$-tableau for a formula A is the limit of a sequence of RDAGs $\Upsilon_0, \ldots, \Upsilon_i, \Upsilon_{i+1}, \ldots$ such that[1]:

- Υ_0 is an RDAG consisting of only one node whose associated set of formulas is $\{A\}$: $\Upsilon_0 = \langle root, \emptyset, \{(root, \{A\})\}\rangle$;

[1] this limit being defined as $\bigcup_{i \geq 0} \Upsilon_i$

- Υ_{i+1} is obtained from Υ_i by applying either a classical rule, or the `createSuccessor` rule, or the rule `propagateFormulas`, or a rule for $(\rho_1 \cup \rho_2)$;
- every applicable rule has been applied.

Note that what we call a tableau here is usually called a tableau branch in the literature. While this is appropriate for classical logic, it might lead to confusions to have branches that are in fact graphs.

DEFINITION 14. A tableau is closed if some node in it contains `False`; it is open otherwise. A formula is $(\rho_1 \cup \rho_2)$-closed iff all its $(\rho_1 \cup \rho_2)$-tableaux are closed [2].

7 A simple strategy language

Suppose we have defined a set of rewriting rules *RULES* for our logic **L**, and we now want to design a strategy and prove it terminates.

Strategies define how the rules from *RULES* are applied, and in which order. Given a set of graphs, the simplest way of applying rule $r \in RULES$ is to apply it to *every* formula in *every* node in *every* graph. It is also the most radical definition. It follows that in order to avoid loops (in the case of modal logics beyond the simplest ones), rule application has to be blocked by means of appropriate marking conditions. For example in S4, `createSuccessor`-rules application in node x is blocked if x is included in some ancestor node.

7.1 Combining rule applications

One of the simplest ways of combining rule applications is by means of regular expressions. We here provide a language that is made up of such expressions, and that is inspired by the concepts that are used in the Isabelle and HOL provers for higher-order logics [10].

We choose the following constructs: `allRules`, `firstRule` and `repeat` to combine rules for defining strategies. They work as follows, where each r_i is itself a strategy, possibly consisting of only one rule:

- (`allRules` $r_1 \dots r_n$) sequentially applies each of the applicable rules to the current graph, starting with r_1.
- (`firstRule` $r_1 \dots r_n$) applies only the first applicable rule among the listed ones to the current tableau.
- (`repeat` r_1) repeatedly applies r_1 until a fixpoint is reached.

The semantic of our constructs `allRules`, `firstRule` and `repeat` for strategies is:

DEFINITION 15. (semantics of a strategy)

[2]Due to the rule $\vee$, which is equivalent to the rule `NotAnd` in the language of LoTREC, a formula may have several distinct tableaux.

$$\texttt{allRules}\ (r_1 \dots r_k)(G) \ = \ (r_k(\dots(r_2(r_1(G)))\dots)$$

$$\texttt{firstRule}\ (nil)(G) \ = \ G, \quad \text{where } nil \text{ is the empty sequence of rules}$$

$$\texttt{firstRule}\ (r_1 \dots r_k)(G) \ = \ r_1(G) \qquad\qquad \text{if } k \geq 1 \text{ and } r_1(G) \neq G$$

$$\texttt{firstRule}\ (r_1 \dots r_k)(G) \ = \ \texttt{firstRule}\ (r_2 \dots r_k)(G) \text{ otherwise}$$

$$\texttt{repeat}\ (r)(G) \ = \ \mathrm{lfp}(r, G)$$

Thus `repeat`-loops build a least fix-point.

To apply a strategy to a formula A means to apply the strategy to the initial graph $G = (\texttt{root}, \emptyset, \{(\texttt{root}, \{A\})\}$.

Of course, a `repeat`-loop may make the resulting strategy loop forever if r is not designed carefully.

8 Completeness

In this section we prove the completeness of our tableau calculi[3]. We show how, from a given open **L**-tableau for A we can construct a ρ-model for A, where ρ is the set of accessibility relations which characterize the logic **L**.

DEFINITION 16 (End Node). x is an end node iff neither `createSuccessor` nor `addPosTrue` nor `confluence` rule are applicable.

DEFINITION 17 (Loop Node). A node x is a loop node in a tableau Υ iff:

1. the rule `createsuccessor` or `addPosTrue` or `confluence` is applicable to x and
2. $\exists y\colon y\Sigma^+ x$ and $x \subseteq y$ (such y will be denoted by *loop(x)*).

DEFINITION 18 (Looping Tableau). A tableau $\Upsilon = (\mathcal{N}, \Sigma, \textsc{for})$ is a *looping tableau* iff each of its leafs is either an end node or a looping node.
The set of loop nodes of Υ is $loop(\Upsilon)$. Note that looping tableaux are finite, since the computation is stopped either because no create new node rules are applicable or due to the inclusion test.

DEFINITION 19 (Folded Tableau).

Let Υ be a looping open **L**-tableau for A such that Tr or $Eucl$ are in **L**, where we stop the execution by making the inclusion test.

The Folded Tableau $\mathcal{F}\Upsilon = (\mathcal{FN}, \mathcal{F}\Sigma, \mathcal{F}\textsc{for})$ corresponding to Υ is defined as follows:

- $\mathcal{FN} = \mathcal{N} - loop(\Upsilon)$;
- $\mathcal{F}\Sigma = \Sigma - \{(x, y)/y \in loop(\Upsilon)\} \cup \{(x, loop(y))/y \in loop(\Upsilon)\}$ where $x \in \mathcal{N}$;
- $\mathcal{F}\textsc{for}$ is the restriction of $\textsc{for}$ to $\mathcal{FN}$.

[3]We make the usual assumption of *fairness*.

Let Υ be an open Folded **L**-tableau for A. Now let $\mathcal{M} = (W, R, \tau)$ be the Kripke model defined as follows:

DEFINITION 20 (Folded Model).

- $W = \mathcal{FN}$
- R is the ρ_1-closure of $\mathcal{F}\Sigma$, i.e. $R = \mathcal{F}\Sigma^{\rho_1}$
- for all $w \in W$, $w \in \tau(p)$ iff $p \in \mathcal{F}\textsc{for}(w)$.

To prove the completeness of a **L**-tableaux we will consider for a formula A a **L**-folded open tableau $\mathcal{F}\Upsilon = (\mathcal{FN}, \mathcal{F}\Sigma, \mathcal{F}\textsc{for})$ and then give a **L**-model for this formula from Υ (as described in Definition 20). Of course, folded models are finite. We first establish the following important lemma:

LEMMA 21 (Box Lemma). *Let $\mathcal{F}\Upsilon = (\mathcal{FN}, \mathcal{F}\Sigma, \mathcal{F}\textsc{for})$ be a folded $(\rho_1 \cup \rho_2)$-tableau. Let x, y be such that $(x, y) \in \mathcal{F}\Sigma^{\rho_1}$ and $\Box A \in x$; then $A \in y$.*

Proof. There are several cases, depending on $(\rho_1 \cup \rho_2)$; we only prove the lemma for some complex cases involving euclideanity:

- $\rho_1 = \{Eucl\}$: if $(x, y) \in \mathcal{F}\Sigma^{\rho_1}$ then by the Relational Closure Lemma (Lemma 4) in the case of a folded tableau, we have:
  - either $(x, y) \in \Sigma$ and then $A \in y$ by rule `propagateFormulas`
  - or $\exists u\ \exists n \geq 1\ \exists m \geq 1$ such that $(u, x) \in \mathcal{F}\Sigma^n$ and $(u, y) \in \mathcal{F}\Sigma^m$. Hence
    - $\exists x_0 = x, \ldots, x_i, x_{i+1}, \ldots, x_n = u$:
      for $0 \leq i \leq n$ we have either:
      - $(x_{i+1}, x_i) \in \Sigma$;then $A, \Box A \in x_i$ (by rules `propagateFormulas` and `CopyNec2Parent`)
      - $(x_{i+1}, x_i) \in \mathcal{F}\Sigma$ and $(x_{i+1}, x_i) \notin \Sigma$; then $\exists z \in \mathcal{N}$: $(x_{i+1}, z) \in \Sigma$ and $z \subseteq x_i$ then $A, \Box A \in z$ (by rules `propagateFormulas` and `CopyNecIfLinked`) and $\Box A \in x_i$ since $z \subseteq x_i$.

      In particular: $A, \Box A \in x_n = u$.
    - $\exists y_0 = u, \ldots, y_i, y_{i+1}, \ldots, y_m = y$:
      - if $m = 1$ then $y = y_1$ and $A \in y$ by rule `propagateFormulas`.

      We have $\Box A \in y_1$ (by rule `copyNec2Brothers` since $\Box A \in x_{n-1}$). For $1 \leq i \leq m$ we have :
      - either $(y_i, y_{i+1}) \in \Sigma$; then $A, \Box A \in y_i$ (by rules `propagateFormulas` and `CopyNecIfLinked`)
      - or $(y_i, y_{i+1}) \in \mathcal{F}\Sigma$; and $(y_i, y_{i+1}) \notin \Sigma$: then $\exists z \in \mathcal{N}$: $(y_i, z) \in \Sigma$ and $z \subseteq y_{i+1}$ then $A, \Box A \in z$ (by rules `propagateFormulas` and `CopyNecIfLinked`) and $A, \Box A \in y_{i+1}$ since $z \subseteq y_{i+1}$.

Since $\Box A \in x_n = u = y_0$ we have: $\Box A \in y_i$ for $0 \leq i \leq m$. Hence $A \in y_i$ for $1 \leq i \leq m$ (by rule `propagateFormulas`), in particular $A \in y$.

- $\rho_1 = \{Tr, Eucl\}$: if $(x, y) \in \mathcal{F}\Sigma^{\rho_1}$ then by the Relational Closure Lemma, we have $\exists u \in \mathcal{N}\ \exists n \geq 0\ \exists m \geq 1$ such that $(u, x) \in \mathcal{F}\Sigma^n$ and $(u, y) \in \mathcal{F}\Sigma^m$. This implies that:
  - $\exists n \geq 0\ \exists x_0 = x, \ldots, x_i, x_{i+1}, \ldots, x_n = u$:
    * $(x_{i+1}, x_i) \in \Sigma$; then $\Box A \in x_i$ implies $\Box A \in x_{i+1}$ (by rule `CopyNec2Parent`)
    * $(x_{i+1}, x_i) \in \mathcal{F}\Sigma$ and $(x_{i+1}, x_i) \notin \Sigma$; then $\exists z \in \mathcal{N}$: $(x_{i+1}, z) \in \Sigma$ and $z \subseteq x_i$ then $\Box A \in x_i$ implies $\Box A \in z$ (by rule `CopyNecIfLinked`) and $\Box A \in x_{i+1}$ since $z \subseteq x_{i+1}$.

    In particular $\Box A \in x_0 = u$.
  - $\exists m \geq 1\ \exists y_0 = u, \ldots, y_i, y_{i+1}, \ldots, y_{m+1} = y$: :
    * if $m = 1$ then $y = y_1$ and $A \in y$ by rule `propagateFormulas`.
    * $(y_i, y_{i+1}) \in \Sigma$; hence $\Box A \in y_i$ implies $\Box A \in y_{i+1}$(by rule `propagateNec2Child`).
    * $(y_i, y_{i+1}) \in \mathcal{F}\Sigma$ and $(y_i, y_{i+1}) \notin \Sigma$: then $\Box A \in y_{i+1}$(same proof as above)

    In particular, if $\Box A \in y_{m-1}$, then $A \in y$ (by rule `propagateFormulas`).
- In the case of logics with the axiom D, we build a serial model by adding to every leaf of the folded tableau a reflexive edge. We call this operation reflexive leaf closure (RLC). The folded tableau will be the tableau obtained from the original tableau by RLC.

  Now if $(x, y) \in \mathcal{F}\Sigma$, we have:
  - either $(x, y) \in \Sigma$, so $A \in y$ by rule `propagateFormulas`;
  - or (x, y) is a RLC edge; in this case $x = y$ is a leaf and it cannot contain a formula of the form $\Box A$. Because if it is the case, the rule `addPosTrue` adds a $\Diamond\top$ to x and `createSuccessor` creates a successor to it.

■

To achieve the completeness proof, it remains to prove that:

for all $B \in sf(A)$ if $B \in x$ then $\mathrm{M}, x \models B$ for all **L**-models obtained from the tableaux of A.

Where $sf(A)$ is the set of subformulas of A. The following fundamental lemma brings us to the desired conclusion:

LEMMA 22 (Fundamental Lemma). *Let $\mathcal{F}\Upsilon$ be an open ρ_1-tableau for A, let $\mathcal{M}$ be the ρ_1-model defined as in (Definition 20) from $\mathcal{F}\Upsilon$, let $B \in sf(A)$ and let $x \in \mathcal{N} - loop(\mathcal{F}\Upsilon)$. Then (i) $B \in x$ iff $\mathcal{M}, x \models B$.*

Proof. By induction on the structure of B. Without loss of generality suppose that B is written only with $\neg, \bot, \wedge, \Box$.

- Induction initialization: let B be an atom, (i) holds by the definition of τ;
- Induction step:
  1. B cannot be `False`, otherwise Υ would be closed;
  2. If B is a propositional classical formula the proof is straightforward.
  3. Let $B = \neg\Box C$.
     $\neg\Box C \in x$
     $\Rightarrow \exists y$ such that $(x, y) \in \mathcal{F}\Sigma$ and $\neg C \in y$ (rule `createSuccessor` or because of the inclusion if x is a loop node)
     $\Rightarrow \exists y$ such that $(x, y) \in R$ and $C \notin y$ ($\mathcal{F}\Sigma \subseteq R$)
     $\Rightarrow \exists y$ such that $(x, y) \in R$ and $\mathrm{M}, y \not\models C$
     $\Rightarrow \mathrm{M}, x \models \neg\Box C$
  4. Let B be $\Box C$ and suppose $(x, y) \in R$; then by the Box Lemma 21, $C \in y$. Then by induction hypothesis, $\mathcal{M}, y \models C$ for every y such that $(x, y) \in R$. Hence, $\mathcal{M}, x \models \Box C$.

■

As a direct consequence of the previous lemma, we have:

COROLLARY 23. *If A has an open looping $(\rho_1 \cup \rho_2)$-tableau then it has a $(\rho_1 \cup \rho_2)$-model.*

We will show later that for all our logics, and for any formula, either all its tableaux are closed or it has at least one open looping tableau. In other words, a formula cannot have an infinite tableau. In the first case we will show that the formula is unsatisfiable (section 9). If is not the case, we will show that it has an open fair looping ρ-tableau (section 10).

9 Soundness

In this section, we prove the soundness of our tableaux calculi: if a formula A is $(\rho_1 \cup \rho_2)$-satisfiable then there exists a open $(\rho_1 \cup \rho_2)$-tableau for A.

We use pattern(r) to note the pattern described by the left hand side of a rule r. To prove the soundness we start by the following proposition:

PROPOSITION 24. *Let Υ be a tableau. Suppose that RULES is subset of the rules given in appendix B. $\forall \rho \in RULES\ \forall g \in pattern(r)$ we have: $g \subseteq \mathcal{F}\Upsilon$ (folded tableau) iff there is $g' \subseteq \Upsilon$ such that g' is a pattern isomorphic to g.*

Proof. The only new patterns are caused by the redirected edges, but these edges existed before the folding operation. ■

PROPOSITION 25. *Let Φ be a modal formula, we have the following implications:*

1. *If Φ is $(\rho_1 \cup \rho_2)$-satisfiable then there is an infinite open tableau for Φ (without loop test);*
2. *If there is an infinite open tableau for Φ then there is a finite open tableau for Φ (by using the looping test);*
3. *If there is an finite open tableau for Φ then there is an open folded tableau for Φ.*

Proof. For the first implication see [3]. The second implication is trivial since we stop the computation by the loop test, the resulting tableau is included in the infinite one and it can not be closed. The last implication results from Proposition 24. ■

10 Termination

The setting we are concerned with is a well-delimited one (labelled graphs and monotonic rules, regular expression-like strategies), and it would be desirable to have general termination results. Unfortunately there seems to be no results in the literature that apply to such a case. For example techniques from rewriting theory such as Knuth-Bendix completion cannot be applied here (in particular because rewriting focusses on rule application in any order, Church-Rosser property etc, while we insist on the contrary).

Here we give a formal termination proof of our strategies: In the case where our strategies are formed only by the constructors: `allRules` and `firstRule` the termination is trivial since in the first one there are a finite number of rules and every applicable one will be applied once, and in the second case only the first applicable rule will be applied.

It remains to prove the termination in the case of `repeat`. We here give two termination criteria, which are enough to handle many logics.

10.1 Adding strict sub-formulas only

Our first kind of strategies only adds strict subformulas in the action part. It might also be called strictly analytic.

THEOREM 26. *Let $d(x)$ be the length of the shortest path from the root to the node x, and let RULES be a set of rules such that for every $r \in RULES$,*

- *r is strictly analytic (only* strict subformulas *of the original formula are added to nodes in the action part of r);*
- *if the action part of r contains the* `createNewNode`*-action creating a node x then*
  - *the condition part of r checks for existence of* at least one formula *in any node among $x_0, \ldots, x_k$, and*

– $d(x) > \max(d(x_0), \ldots, d(x_k))$, *i.e. new nodes are strictly farther from the root (ensuring strict decrease of their contents),*

where $x_0, \ldots, x_k$ *are the nodes referred to in the condition part of* r.

Let S *be any strategy on* $RULES$. *Let* G *be a finite graph. Then the application of* S *to input* G *terminates, and* $S(G)$ *is a finite graph.*

Proof. We prove that

- application of S can never lead to RDAGs with infinite branching factor;
- application of S can never lead to RDAGs of infinite depth.

The proof is part of the proof of Theorem 27.

■

Theorem 26 covers the case of logics like K, KD, KT, K+confluence. More generally, such strategies can account for many modal logics without transitive accessibility relations, in particular those with symmetric accessibility relations such as KB, KDB, and KTB.

10.2 Testing loops

Our second criterion applies in particular to logics with transitive accessibility relations. In these logics we use a dynamic loop test (DLT): the `createSuccessor`-rule does not apply in a node if this node is included in some ancestor [4]. We say that `createSuccessor`-rule is enriched by DLT. We discuss how DLT is implemented in LoTREC in the conclusion.

THEOREM 27. *Let* $RULES$ *be a set of rules such that for every* $r \in RULES$,

- r *is analytic (only* subformulas *of the original formula are added to nodes in the action part of* r*);*
- *The* `createSuccessor`*-rule does not apply in a node which is included in some ancestor.*

Let S *be a strategy that is built from* $RULES$. *Then the tableaux construction terminates, in other words apply* S *to any formula leads to a looping tableau.*

Proof. First, observe that due to the subformula condition:

- every node only contains a finite number of formulas, and
- there can only be a finite number of nodes with differing associated set of formulas.

[4] ...i.e. we have $x_0(\bigcup\Sigma)^+x$, and every formula A appearing in x also appears in x_0.

We prove that (1) application of S can never lead to RDAGs of infinite depth, and (2) application of S can never lead to RDAGs with infinite branching factor.

For Theorem 26, the argument is that creation of new nodes is subject to non-emptiness, but with our constraints, a branch cannot be of length more than the modal degree of the input formula: because of the strict sub-formula condition, nodes situated farther would be empty, and then the criterion for applying the `createNewNode`-action would no more be fulfilled.

For Theorem 27, the argument is that creation of new nodes is subject to non-inclusion in an ancestor node, which due to the condition on S is always tested before node creation. Therefore a branch of infinite length would contain an infinite number of nodes having different associated formula sets. This cannot be the case because our rules are monotonic: no formulas are erased. This even gives us the classical upper bound: length of branches is bounded by an exponential in the length of the input formula[5].

Infinite branching in node x could only be produced by introducing infinitely many new nodes of the form $\Diamond(x, A)$. But given a node n, the number of $\Diamond$ formulas being bounded by the (linear) number of sub-formulae of the input formula, each node may only have a linear number of successors. Thus the branching factor is bounded. This ends the proof. ■

The important point of our works is that contrarily to what can be found in the literature, any fair strategy using the set of rules corresponding to a logic $(\rho_1 \cup \rho_2)$ is sound, complete and terminating. This contrasts with the rather procedural strategies of the literature with either natural language argument for termination [7] or complex and procedural proof [9].
This theorem covers the case of logics build over the axioms K, T, B, 4, 5, k4+confluence, KDB, in addition to any logic results from the combination of the axioms of group 1 and group 2.

To sum it up, the above theorems give us some general termination criteria for our strategies. These strategies cover all standard modal logics, including Linear Temporal Logic LTL and Propositional Dynamic Logic PDL (see the LoTREC web page for the rules).

For logics that do not fit into the above framework we have to prove termination on a case by case basis.

11 Conclusion

We have defined a graph-based formal language in terms of rules and strategies that enables programming tableau systems for modal logics. We have given a semantics to it, and proved termination results for two important classes of strategies, as well as terminating strategies for the modal logics of confluence. We have proved also the completeness and soundness of all modal logics build from the basic axioms.

[5] In fact the standard argument applies that limits the length of paths to a polynomial, since non strict analytic rules are $\Box$-propagation ones. Hence along a path the number of $\Box$-ed formulae is always increasing

All these strategies can be tested in our generic tableaux theorem prover LoTREC that is accessible via Internet. LoTREC is similar in its aims to the Tableaux Workbench (TWB) [1], which also aims at a generic tableaux theorem prover. They differ in perspective: while LoTREC adopts the more semantic model building view, TWB opts for the more syntactic sequent calculus view.

We implement dynamic loop test in LoTREC by using the marking mechanism. We suppose that the set of rules contains among others the following.

- The rule `markIncludedNodes`, which in LoTREC takes the form

```
rule markIncludedNodes
     if isAncestor    node0  node
     if contains      node0  node
     do mark          node CONTAINED
end
```

- One or more node-creating rules `createSuccessor(pos,R)` (with a modal operator `pos` and an accessibility relation `R` as parameters), which in LoTREC take the form

```
rule createSuccessor
     if hasElement    node    pos (variable A)
     if isNotMarked   node    CONTAINED
     do createNewNode node1
     do ...
     ...
end
```

Where no other `CreateNewNode` action appears in the action part.

The first rule blocks every node x whose associated set of formulas is included in another node x_0 that is an ancestor of x by marking x appropriately. The second kind of rules creates a Σ-successor x_1 for every node x and modal formula[6] $\Diamond A$ in x, and adds some formulas and/or marks to x and x_1, *under the condition that x is not marked as being included in some ancestor node.*

But then one should only mark a node as CONTAINED if this node is not subject to further modifications. This restricts the set of terminating strategies.

If one is only interested in the decision problem of satisfiability then instead of testing if a node is included, it suffices to test its depth in the tableau. But if one wishes to build models (as we do), testing node inclusion is necessary.

[6]In practice the formula will be of the 'possible'-type (we therefore used a generic $\Diamond$, alias "`pos`" in LoTREC's notation), and R is the accessibility relation associated to $\Diamond$.

Appendix

A Some rules in the graph language

In this appendix, we give some rules written in $\mathcal{RL}$ language. Recall that a rule in $\mathcal{RL}$ has the following form: $\langle g, \{g_1, \ldots, g_n\}\rangle$.

For sake of clarity, in the right side of the rule, we give only what is new to the graph. For the same reason, the node's names in LoTREC language (`node0`,`node1`,...) will be written $n_0, n_1, \ldots$. S is supposed to be a set of formulae, and N a set of nodes, n, n_0, n_1, n_2 are universally quantified.

Rules In LoTREC	Rules in $\mathcal{RL}$
`Stop`	$(\langle\{n\}, \emptyset, \{(n, \{A, \neg A\}\}\rangle, \{\langle\emptyset, \emptyset, \{(n, \{\bot\}\}\rangle\})$
`And`	$(\langle\{n\}, \emptyset, \{(n, \{A \wedge B\}\}\rangle, \{\langle\emptyset, \emptyset, \{(n, \{A, B\}\}\rangle\})$
`NotNot`	$(\langle\{n\}, \emptyset, \{(n, \{\neg\neg A\}\}\rangle, \{\langle\emptyset, \emptyset, \{(n, \{A\}\}\rangle\})$
`NotAnd`	$(\langle\{n\}, \emptyset, \{(n, \{\neg(A \wedge B)\}\}\rangle, \{\langle\emptyset, \emptyset, \{(n, \{\neg A\}\}\}\rangle,$ $\langle\emptyset, \emptyset, \{(n, \{\neg B\}\}\}\rangle\})$

For logics built from the combinations of the axioms $K, T, B, D, 5$ here are their transformations:

Rules In LoTREC	Rules in $\mathcal{RL}$
`createSuccessor`	$(\langle\{n_0\}, \emptyset, \{(n_0, \{\neg\Box A\})\}\rangle,$ $\{\langle\emptyset, \{(n_0, n)\}, \{(n, \{\neg A\})\}\rangle\})$
`propagateFormulas`	$(\langle\{n_0, n_1\}, \{(n_0, n_1)\}, \{(n_0, \{\Box A\})\}\rangle,$ $\{\langle\emptyset, \emptyset, \{(n_1, \{A\})\}\rangle\})$
`addFormula2Parent`	$(\langle\{n_0, n_1\}, \{(n_0, n_1)\}, \{(n_1, \{\Box A\})\}\rangle,$ $\{\langle\emptyset, \emptyset, \{(n_0, \{A\})\}\rangle\})$
`addPosTrue`	$(\langle\{n_0\}, \emptyset, \{(n_0, \{\Box A\})\}\rangle, \{\langle\emptyset, \emptyset,$ $\{(n_0, \{\neg\Box\neg\top\})\rangle\})$
`PropagatNec2ActualWorld`	$(\langle\{n_0\}, \emptyset, \{(n_0, \{\Box A\})\}\rangle, \{\langle\emptyset, \emptyset,$ $\{(n_0, \{A\})\rangle\})$
`copyNecIfLinked`	$(\langle\{n_0, n_1, n_2\}, \{(n_0, n_1), (n_1, n_2)\},$ $\{(n_1, \{\Box A\})\}\rangle, \{\langle\emptyset, \emptyset, \{(n_2, \{\Box A\})\rangle\})$
`copyNec2Parent`	$(\langle\{n_0, n_1\}, \{(n_0, n_1)\}, \{(n_1, \{\Box A\})\}\rangle,$ $\{\langle\emptyset, \emptyset, \{(n_0, \{\Box A\})\rangle\})$
`copyNec2Brothers`	$(\langle\{n_0, n_1, n_2\}, \{(n_0, n_1), (n_0, n_2)\},$ $\{(n_1, \{\Box A\})\}\rangle, \{\langle\emptyset, \emptyset, \{(n_2, \{\Box A\})\rangle\})$

B Some rules in the LoTREC Language

Here are the rules we need for dealing with the logics of group 1 and 2 written in LoTREC Language. In the second appendix, we give the equivalence of some of these rules in the formal form given in definition 9(texts after // are comments):

Rules for classical logics

```
rule Stop
  if hasElement node0   (variable A)
  if hasElement node0   not (variable A)
  do add node0 FALSE
```

```
  do stop node0
end

rule NotNot
  if hasElement node0   not not (variable A)
  do add        node0   (variable A)
end

rule And
  if hasElement node0   and (variable A) (variable B)
  do add        node0   (variable A)
  do add        node0   (variable B)
end

rule NotAnd
  if hasElement node0   not and (variable A) (variable B)
  do duplicate  node0   begin node0 node1 end
  do add        node0   not (variable A)
  do add        node1   not (variable B)
end
```

Rules for modal logics

```
rule createSuccessor
  if hasElement    node0        not nec (variable A)
  do createNewNode node0  node1
  do link          node0  node1 R
  do add           node1        not (variable A)
end

rule propagateFormulas
 if hasElement node0 nec variable a
 if isLinked node0 node1  R
 do add node1 variable a
end

rule addFormula2Parent // For axiom B
 if hasElement node0 nec variable A
 if isLinked node1 node0 R
 do add node1 variable A
end

rule addPosTrue // For axiom D
   if hasElement node0   nec variable A
   do add node0   not nec not True
end
```

```
rule propagateNec2ActualWorld // For axiom T
  if hasElement node0   nec (variable A)
  do add        node0   (variable A)
end

// The following three rules are for the axiom 5
rule copyNecIfLinked
  if hasElement  node1  (nec (variable A))
  if isLinked    node0  node1  (R)
  if isLinked    node1  node2  (R)
  do add         node2  (nec (variable A))
end

rule copyNec2Parent
  if isLinked     node0  node1 (R)
  if hasElement   node1  nec (variable A)
  do add          node0  nec (variable A)
end

rule copyNec2Brothers
  if isLinked      node0  node1 (R)
  if hasElement    node1  nec (variable A)
  if isLinked      node0  node2 (R)
  do add           node2  nec (variable A)
end
```

Rules for modal logics with Transitivity

```
rule createSuccessor
    if  hasElement    node0 pos variable A
    if  isNotMarked   node0 CONTAINED
    do  createNewNode node0 node1
    do  link          node0 node1 R
    do  add           node1 variable A
end

//This rule is for the inclusion test
rule loopTest
    if isNewNode node1
    if isAncestor node0 node1
    if contains node0 node1
    do mark node1 CONTAINED
end

//This rule is for the 2 axioms K and 4
rule propagateNec2Child
```

```
  if hasElement node0   (nec (variable A))
  if isLinked node0 node1  (R)
  do add node1     (nec (variable A))
end

//In addition to the rule propagateFormulas given above
```

Rules for modal logics with confluence

```
rule confluence true // true is to say that the rule is
commutative
    if isLinked N0 N1   (R)
    if isLinked N0 N2  (R)
    if areNotIdentical N1 N2
    if hasElement N1 variable A // tests if  N1 is not empty.
    if hasElement N2 variable B // tests if  N2 is not empty.
    do createNewNode  N0 N3
    do link N1 N3(R)
    do link N2 N3 (R)
end
```

BIBLIOGRAPHY

[1] Pietro Abate and Rajeev Goré. The tableaux work bench. In *TABLEAUX*, pages 230–236, 2003.

[2] Kai Brünnler. Deep sequent systems for modal logic. This volume.

[3] Marcos A. Castilho, Luis Farinas del Cerro, Olivier Gasquet, and Andreas Herzig. Modal tableaux with propagation rules and structural rules. *Fundamenta Informaticae*, 32(3-4):281–297, 1997.

[4] Brian Chellas. *Modal logic: An introduction*. Cambridge University Press, 1980.

[5] Melvin Fitting. *Proof methods for modal and intuitionistic logics*. D. Reidel, Dordrecht, 1983.

[6] Olivier Gasquet, Andreas Herzig, Dominique Longin, and Mohamad Sahade. Lotrec: Logical tableaux research engineering companion. In *TABLEAUX*, pages 318–322, 2005.

[7] R.P. Goré. Tableau methods for modal and temporal logics. In M. D'Agostino, D. Gabbay, R. Hähnle, and J. Posegga, editors, *Handbook of Tableau Methods*. Kluwer Academic Publishers, 1999.

[8] A. Heuerding, M. Seyfried, and H. Zimmermann. Efficient loop-check for backward proof search in some non-classical propositional logics. In P. Miglioli, U. Moscato, D. Mundici, and M. Ornaghi, editors, *Proceedings of the 5th International Workshop TABLEAUX'96: Theorem Proving with Analytic Tableaux and Related Methods*, number 1071 in LNAI, pages 210–225. Springer-Verlag, 1996.

[9] Fabio Massacci. Strongly analytic tableaux for normal modal logics. In Alan Bundy, editor, *Proc. 5th Int. Conf. on Automated Deduction (CADE)*, volume 814 of *LNAI*, pages 723–737. Springer-Verlag, 1994.

[10] L. C. Paulson. Isabelle: A generic theorem prover. *LNCS*, pages 748–752, 1994.

[11] H. Sahlqvist. Completeness and correspondence in the first and second order semantics for modal logics. In Stig Kanger, editor, *Proc. 3rd Scandinavian Logic Symposium 1973*, number 82 in Studies in Logic. North Holland, 1975.

Olivier Gasquet, Andreas Herzig and Mohamad Sahade
IRIT- Université Paul Sabatier,
118 route de Narbonne,
31062 Toulouse cedex 04, France.
{gasquet, herzig, sahade} @irit.fr