A General Theorem Prover for Quantified Modal Logics

V. Thion[1], S. Cerrito[1] and M. Cialdea Mayer[2]

[1] Université de Paris-Sud, L.R.I.
[2] Università di Roma Tre, Dipartimento di Informatica e Automazione

Abstract. The main contribution of this work is twofold. It presents a modular tableau calculus, in the free-variable style, treating the main domain variants of quantified modal logic and dealing with languages where rigid and non-rigid designation can coexist. The calculus uses, to this end, light and simple semantical annotations. Such a general proof-system results from the fusion into a unified framework of two calculi previously defined by the second and third authors. Moreover, the work presents a theorem prover, called **GQML-Prover**, based on such a calculus, which is accessible in the Internet. The fair deterministic proof-search strategy used by the prover is described and illustrated via a meaningful example.

1 Introduction

This work presents a theorem prover, called **GQML-Prover** for quantified analytic modal logics, having the following features:

1. it deals with formulae possibly containing both rigid and non-rigid terms;
2. it is parametric with respect to three domain variants: constant, increasing and freely varying domains;
3. it is parametric with respect to five propositional bases of the logic: **D**, **K**, **K4**, **T**, **S4**.

The prover is implemented in Objective Caml and is based on a global modular tableau calculus in the free variable style, that results from the fusion into a same framework of two calculi presented, respectively, in [3, 4] and [2], i.e.:[1]

- free-variable tableaux parametric w.r.t. rigid and non-rigid designation of terms, increasing or varying domains and propositional analytical bases, and
- a free variable calculus for constant domain quantified modal logics, with both rigid and non-rigid symbols.

The general proof-system is not simply the justapposition of two different calculi. To start with, it deals with both rigid and non-rigid designation coexisting in the same language, for all domains. Such a feature is absent in [3, 4], where varying and increasing domains are treated, while present in [2], where constant domains

[1] For the sake of simplicity, in this work we do not consider non-local terms, that are taken into account in [3, 4].

are dealt with. Having both kinds of designation in the language is interesting for many applications, for instance in database modeling, where both rigid and non-rigid attributes need to be dealt with (e.g. *birth-date*, and, respectively, *salary*).

As a second point, the present system uses a unique labeling technique for all domain variants, while the node labelling mechanism was slightly different in the cited previous works.

The final, and more important, point concerns the treatment of the expansion rule for existential formulae. In fact, the calculi in [3, 4] use the "traditional" δ-rule to expand existential quantified formulae, where the arguments of Skolem functions are *all* the free variables (of appropriate level) in the set of formulae $\exists x A(x), S$ to be expanded. On the contrary, [2] shows that the "liberalized" δ^+-rule, where the only arguments of Skolem functions are the free variables actually occurring in A, and that is proved sound for classical logic in [10], characterizes constant domains, in the absence of any semantical annotation. So, the fusion of the two approaches requires proving soundness of the "liberalized" δ^+-rule for non-constant domain logics, in the annotated calculi we consider. This is not a trivial task: the proof requires some subtleties (see the end of Section 3).

Section 2 introduces the syntax and semantics of Quantified Modal Logics (QML). The general tableau system is presented in Section 3, together with the soundness proof of the liberalized δ^+-rule used by the **GQML-Prover** for all domain variants. The main feature of the system, in comparison with other approaches such as prefixed tableaux [6], matrix methods [15] and others, such as those cited in [2, 4], is the effort to minimize expression labelling.

A further important issue that must be faced in order to turn a tableau calculus into an automated theorem prover is the definition of a complete proof-search strategy. The issue is not trivial, because of the interplay of applications of the γ-rule, possible sources of an infinite number of instantiations of a universal formula, and the choice of closing substitutions with the "dynamical" modal rules, which cause one to forget the past. Section 4 presents the **GQML-Prover** and the underlying search strategy. In order to ensure termination of proof search, the strategy uses two upper bounds on the size of candidate proofs, whose values are entered by the user. It is complete in the sense that if a formula is valid, then a proof is found, provided that the input bounds are big enough. Note that, although some theorem provers for non-classical logics exist (for instance, ModLeanTap [1] deals with propositional modal logics, while IleanTap [12] deals with quantification, but only in a intuitionistic logic setting), up to our knowledge, not much has been done on QML.

Section 5 concludes this work with a discussion on equality.

2 Quantified Modal Logics

In Kripke semantics for quantified modal logic an interpretation is, roughly, a set of first-order classical interpretations (the "possible worlds") connected by an accessibility relation. The worlds can either all have the same object domain (constant domains) or the domains can be allowed to be different (varying domains), possibly monotonically increasing with the accessibility relation (cumulative domains). Terms can designate the same object in each world (rigid terms) or can be allowed to designate differently in different worlds (non-rigid terms). Rigid and non-rigid functional symbols may coexist in the same language: the set L_F of functional symbols of a language is partitioned into a set L_{F_R} of rigid functional symbols, and a disjoint set $L_{F_{NR}}$ of non-rigid functional symbols.

From now on, we consider modal formulae in negation normal form, i.e. built out of literals (atoms and negated atoms) by use of $\wedge$, $\vee$, $\Box$, $\Diamond$ and the quantifiers $\forall$ and $\exists$. Negation over non-atomic formulae and implication are considered as defined symbols.

Below, we recall standard definitions on quantified modal logics [6]. A first-order modal interpretation $\mathcal{M}$ of a language L is a tuple $\langle W, w_0, R, D, \delta, \phi, \pi \rangle$ such that:

- W is a non empty set (the set of "possible worlds");
- w_0 is a distinguished element of W (the "initial world");
- R is a binary relation on W (the *accessibility relation*); wRw' abbreviates $\langle w, w' \rangle \in R$;
- D is a non empty set (the "global" object domain);
- δ is a function assigning to each $w \in W$ a non empty subset of D, the domain of w: $\delta(w) \subseteq D$;
- ϕ represents the interpretation of constants and functional symbols in the language: for every world $w \in W$ and k-ary functional symbol $f \in L_F$ (with $k \geq 0$),
$$\phi(w, f) \in D^k \to D$$
Moreover, if $f \in L_{F_R}$, then for all $w, w' \in W$, $\phi(w, f) = \phi(w', f)$. We consider here only *local* terms, i.e. for all $w \in W$ and $f \in L_F$, if $d_1, ..., d_n \in \delta(w)$, then $\phi(w, f)(d_1, ..., d_n) \in \delta(w)$.
- π is the interpretation of predicate symbols: if p is a k-ary predicate symbol and $w \in W$, then $\pi(w, p) \subseteq D^k$ is a set of k-ples of elements in D.

The accessibility relation R of a modal structure can be required to satisfy additional properties, characterizing different logics: seriality (**D**), reflexivity (**T**), transitivity (**K4**), both reflexivity and transitivity (**S4**). When no additional assumption on R is made, the logic is **K**.

On the first-order side, the main "domain variants" of QML are characterized as follows:

Constant domains: for all $w \in W$, $\delta(w) = D$.
Cumulative domains: for all $w, w' \in W$, if wRw' then $\delta(w) \subseteq \delta(w')$.
Varying domains: no restrictions on $\delta(w)$.

If $\mathcal{M}$ is an interpretation with object domain D, a variable assignment on $\mathcal{M}$ is a function $s : X \to D$, where X is the set of variables of the language. If s is a variable assignment, x is a variable and $d \in D$, then s_x^d is the variable assignment such that $s_x^d(x) = d$ and $s_x^d(y) = s(y)$ for all $y \neq x$.

Given a world $w \in W$ and a variable assignment s, the interpretation of a term t in w according to s, $s(w, t)$, is inductively defined as follows:

1. If x is a variable, then $s(w,x) = s(x)$.
2. If c is a constant, then $s(w,c) = \phi(w,c)$.
3. If f is a k-ary functional symbol and $t_1,...,t_k$ are terms, then $s(w, f(t_1,...,t_k)) = \phi(w,f)(s(w,t_1),...,s(w,t_k))$.

If t is a ground term, its designation is independent from the variable assignment and is denoted by $\mathcal{M}(w,t)$.

The relation $\models$ between an interpretation $\mathcal{M} = \langle W, w_0, R, D, \delta, \phi, \pi\rangle$, a variable assignment s on $\mathcal{M}$, a world $w \in W$ and a formula is defined inductively as follows:[2]

1. $\mathcal{M}, s, w \models p(t_1,...,t_n)$ iff $\langle s(w,t_1),...,s(w,t_n)\rangle \in \pi(w,p)$.
2. $\mathcal{M}, s, w \models \neg A$ iff $\mathcal{M}, s, w \not\models A$.
3. $\mathcal{M}, s, w \models A \wedge B$ iff $\mathcal{M}, s, w \models A$ and $\mathcal{M}, s, w \models B$.
4. $\mathcal{M}, s, w \models A \vee B$ iff $\mathcal{M}, s, w \models A$ or $\mathcal{M}, s, w \models B$.
5. $\mathcal{M}, s, w \models \forall x A$ iff for all $d \in \delta(w)$, $\mathcal{M}, s^d_x, w \models A$
6. $\mathcal{M}, s, w \models \exists x A$ iff there exists $d \in \delta(w)$ such that $\mathcal{M}, s^d_x, w \models A$
7. $\mathcal{M}, s, w \models \Box A$ iff for all $w' \in W$ such that wRw', $\mathcal{M}, s, w' \models A$
8. $\mathcal{M}, s, w \models \Diamond A$ iff there is a $w' \in W$ such that wRw' and $\mathcal{M}, s, w' \models A$

A formula A is true in $\mathcal{M}$ iff for all variable assignments s on $\mathcal{M}$: $\mathcal{M}, s, w_0 \models A$, and it is valid iff it is true in all interpretations. Note that, if t is a ground term and $w \in W$, then for all variable assignment s, $s(w,t) \in \delta(w)$. However, if domains are allowed to vary without restrictions, if $\mathcal{M}, s, w \models \exists x \Diamond p(f(x))$, and d is the element of $\delta(w)$ such that $\mathcal{M}, s^d_x, w \models \Diamond p(f(x))$, then (by definition) there exists a world w' accessible from w and such that $\mathcal{M}, s^d_x, w' \models p(f(x))$; but $s^d_x(w', f(x))$ is not necessarily in the domain of the world w', unless $d \in \delta(w')$.

3 The Tableaux Systems

In this section we introduce the calculi which are the basis of the **GQML-Prover**. They result from the fusion into a same framework of the free-variable tableaux systems described in [3, 4] and [2], but for the fact that a liberalized δ-rule is used in all calculi, and not only for the constant domain case. The soundness proof for the varying and cumulative domain logics can be found at the end of the section.

In the calculi we are going to describe, tableau nodes are labelled by $n : S$ where n is a positive integer (the "name" of a possible world) and S a set of annotated formulae. The integer n is called the *depth* of the node. In an annotated formula, functional symbols can be annotated by a superscript natural number: the "name" of the world where they are meant to designate. If f is a rigid symbol of the original language, it always occurs as f^0 in the tableau. If f is non-rigid, it occurs either with no annotation or as $f^1, f^2, ...$ The language of a tableau includes also annotated *free variables*, $v^k_0, v^k_1, v^k_2, ...$, and annotated

[2] The reader will recognize that this approach to the interpretation of non-rigid functional symbols corresponds to the "narrow-scope" approach discussed in [8].

Skolem functional symbols. Skolem functions and annotated functional symbols are considered as rigid symbols.

The initial tableau for a set S of formulae is $1 : S^*$, where S^* is obtained from S by annotating rigid symbols with 0 and *non-modal occurrences* of non-rigid symbols with 1 – where a symbol occurrence is non-modal if it is in the scope of no modal operators.

The formulation of the expansion rules require the definition of *depth* of a term, that – intuitively – establishes a relation between annotated terms and positive integers. So, "t is at depth n" means its intended designation belongs to the domain of the world named by n.

> A term t is at **depth** n iff every functional symbol in t is annotated and such symbols are only annotated with:
> **varying domains:** n and 0;
> **cumulative domains:** $k \leq n$;
> **constant domains:** any k.

Similarly, we define the world domains over which a variable may range:

> A free variable v^n **ranges** over depth k iff:
> **varying and cumulative domains:** $k = n$;
> **constant domains:** $k \in \mathbb{N}$ (i.e. always)

Non-rigid symbols occurring in the scope of some modal operator are initially non-annotated. They get their annotation when, by application of a modal expansion rule, they come to the surface. The annotation they are given is the label of the tableau node in which they occur. The definition of the expansion rules requires then the following last definition: if A is a modal formula and $n \in \mathbb{N}$, then A^n is obtained from A by annotating each non-modal occurrence of a non-rigid functional symbol with n. If S is a set of modal formulae, then $S^n = \{A^n \mid A \in S\}$.

As a leading example we consider the case of QMLs based on **S4**. The other analytic propositional bases treated by the **GQML-Prover**, i.e. **K**, **D**, **T** and **K4**, are obtained simply by modification of the modal rules. The tableau expansion rules for **S4** are shown in Figure 1, where S, S' are sets of annotated modal formulae, $\Box S$ stands for $\{\Box A \mid A \in S\}$, S' is a set of non-boxed modal formulae, comma is set union. Rules generating a new node label are called *dynamic* (only π_4 in the case of **S4**), the others are *static*.

Note that, in the case of varying or cumulative domains, in the δ^+-rule, $k_1 = k_2 = ... = k_m = n$. In the case of constant domains, instead, the annotation of free-variables and Skolem functional symbols can actually be ignored. In this variant, in fact, the only relevant annotations are those on non-rigid symbols, which are taken into account by unification (symbols annotated differently are considered as different symbols).

Besides the expansion rules, the calculi provide a substitution rule. The notion of substitution however takes into account the depth of terms:

> A modal substitution σ is a set of pairs $\{t_1/v_1^{k_1}, ..., t_m/v_m^{k_m}\}$, where each t_i is at depth k_i.

Note again that, in the case of constant domains, the annotation of symbols is not relevant, since any term is at any depth.

The substitution rule of the calculus is the *MGU atomic closure rule*: if $\mathcal{T}$ is a tableau for a set S of sentences and some leaf of $\mathcal{T}$ contains P and $\neg Q$, where P and Q are atomic, then $\mathcal{T}\sigma$ is also a tableau for S, where σ is a most general unifier of P and Q.

A tableau is closed iff each of its leaves contains a pair of complementary literals, i.e. literals P and $\neg P$. A closed tableau for a formula $\neg A$ is a tableau proof of A, and a closed tableau for a set of formulae S is a refutation of S.

Classical propositional rules	
$(\alpha)\quad \dfrac{n : A \wedge B, S}{n : A, B, S}$	$(\beta)\quad \dfrac{n : A \vee B, S}{n : A, S \qquad\quad n : B, S}$
Modal propositional rules	
$(\nu_T)\quad \dfrac{n : \Box A, S}{n : A^n, \Box A, S}$	$(\pi_4)\quad \dfrac{n : \Diamond A, \Box S, S'}{m : A^m, S^m, \Box S}$ where $m \in \mathbb{N}$ is new in the whole tableau and $m > n$
Quantifier rules	
$(\gamma)\quad \dfrac{n : \forall x A, S}{n : A[v^n/x], \forall x A, S}$ where v is a new free variable	$(\delta^+)\quad \dfrac{n : \exists x A, S}{n : A[f^n(v_1^{k_1}, ..., v_m^{k_m})/x], S}$ where f is a new Skolem function, i.e. a symbol that does not occur elsewhere in the tableau, and $v_1^{k_1}, ..., v_m^{k_m}$ are all the free variables occurring in $\exists x A$ and ranging over n

Fig. 1. Expansion rules

Note that the δ^+-rule only requires free-variables actually occurring in $\exists x A$ to be taken as parameters of the Skolem function. In free-variable calculi with no annotation, the δ^+-rule characterizes constant domains [2]. An intuitive justification of the soundness of the δ^+-rule in the cumulative or varying domain cases can be given as follows: in a non-annotated calculus we need the full δ-rule because the substitution of a free variable v, introduced at a given depth n, with a Skolem term t introduced at a greater depth must be forbidden, because v ranges over a domain that may not contain the designation of t. But such a constraint is automatically satisfied when depths of terms are taken into consideration by the notion of substitution itself.

The main difficulty in adapting the soundness proof of the δ^+-rule for classical logic to the modal non-constant domain cases is due to the interplay between the

δ^+-rule and substitution, in presence of the fact that tableau nodes can contain variables ranging over different world domains and terms denoting in different worlds. The proof sketched below makes use of an unrestricted grounding substitution rule, that must always be applied before dynamic rules, so that one doesn't have to keep track of the domains of previously considered worlds.

Let us consider the tableau calculi obtained by replacing the MGU atomic closure rule with unrestricted substitution: if $\mathcal{T}$ is a tableau for S and σ is a modal substitution then $\mathcal{T}\sigma$ is a tableau for S. Clearly, if there is a closed tableau for S using the MGU atomic closure rule, then there is a closed tableau for S using the unrestricted substitution rule.

Now, let $\mathcal{T}\sigma$ be a closed tableau for a set S of sentences, where σ is the composition of all the modal substitutions used to close $\mathcal{T}$. We may assume, without loss of generality, that σ is a grounding substitution, i.e. that $\mathcal{T}\sigma$ is a ground tableau. In fact, if $\mathcal{T}\theta$ is closed and it contains the free variables $v_1^{k_1}, ..., v_n^{k_n}$, and $a_1^1, a_2^2, ...$ are a new constants (the "dummy" constants, one for each annotation), then clearly

$$\mathcal{T}(\theta \circ \{a_{k_1}^{k_1}/v_1^{k_1}, ..., a_{k_n}^{k_n}/v_n^{k_n}\})$$

is a ground and closed tableau for S.

If $max(\mathcal{T})$ is the maximal label of a node in $\mathcal{T}$, we may assume, without loss of generality, that whenever a tableau $\mathcal{T}$ is expanded by means of a dynamic rule, the new integer m used to label the expansion is equal to $1 + max(\mathcal{T})$. For all $i = 1, ..., n = max(\mathcal{T})$, let $\mathcal{T}_i$ be the maximal subtree of $\mathcal{T}\sigma$ where the label of every node is less than or equal to i. We can assume that $\mathcal{T}\sigma$ is built as the sequence $\mathcal{T}_1, ..., \mathcal{T}_n = \mathcal{T}\sigma$. In other words, at first all the static rules are applied to the initial node, then σ is applied (so that the ground tableau $\mathcal{T}_1$ is obtained), then a dynamic rule is applied, followed by a sequence of static rules; then σ is applied again (obtaining $\mathcal{T}_2$) before the next dynamic rule application, and so on.

Note that, for all k, while building $\mathcal{T}_k$ starting from $\mathcal{T}_{k-1}$, tableau nodes only contain free-variables annotated with k. In fact, $\mathcal{T}_{k-1}$ is a ground tableau, and the new expansion rules applied to obtain $\mathcal{T}_k$ from $\mathcal{T}_{k-1}$ only affect nodes labelled with k, so if any new free variable is introduced by means of the γ-rule, it is annotated with k. Moreover, in the construction of the sequence $\mathcal{T}_1, ..., \mathcal{T}_n$, only "grounding substitutions" are applied, i.e. when σ is applied, the result is a ground tableau.

In order to prove soundness, it is sufficient to show that if S is satisfiable then there is no sequence $\mathcal{T}_1, ..., \mathcal{T}_n$ of tableaux, built as illustrated above, and such that $\mathcal{T}_n$ is closed. This, in turn, amounts to proving soundness of the calculus Tab^*, allowing the general substitution rule but with the additional constraints that:

- when a substitution is applied, the result is a ground tableau, and
- when a dynamic rule is applied, the tableau contains no free variables.

This can be done along standard lines, by showing that if S is satisfiable then any Tab^*-tableau for S is satisfiable. However, the notion of satisfiability of a tableau requires some machinery. We first define when a "static" tableau is satisfiable, i.e. a tableau that has never been expanded by application of a dynamic rule.

In general, the interpretation of a language with annotations is a modal interpretation where:

- symbols annotated differently are considered as different symbols;
- annotated functional symbols are rigid.

Definition 1. *If $\mathcal{T}$ is a static Tab^*-tableau, whose nodes are all labelled by n, $\mathcal{M}$ is an interpretation of the language of $\mathcal{T}$ and w a world in $\mathcal{M}$, then $\mathcal{M}, w \models \mathcal{T}$ ($\mathcal{M}$ satisfies $\mathcal{T}$ at w) iff for all variable assignment s, if $s(v^n) \in \delta(w)$ for all free-variable v^n annotated by n, then there exists a leaf $n : S$ of $\mathcal{T}$ such that $\mathcal{M}, s, w \models S$.*

A static tableau is satisfiable iff there exists an interpretation $\mathcal{M}$ and a world w in $\mathcal{M}$ such that $\mathcal{M}, w \models \mathcal{T}$.

Satisfiability of a generic tableau amounts to satisfiability of one of its maximal static terminal sub-tableaux:

Definition 2. *If $\mathcal{T}$ is a tableau, then its terminal sub-tableaux are all the static sub-trees $\mathcal{T}'$ of $\mathcal{T}$ such that:*

- *the root of $\mathcal{T}'$ is either the root of $\mathcal{T}$ itself, or a node obtained by application of a dynamic rule;*
- *all the leaves of $\mathcal{T}'$ are leaves of $\mathcal{T}$.*

A tableau $\mathcal{T}$ is satisfiable iff it has a satisfiable terminal sub-tableau.

Soundness of Tab^* follows from the following

Lemma 1. *If S is satisfiable, then any Tab^*-tableau $\mathcal{T}$ for S is satisfiable.*

The proof is by induction on $\mathcal{T}$, showing how to build an interpretation $\mathcal{M}$ of the annotated language of $\mathcal{T}$ and a world w in $\mathcal{M}$ such that for some terminal sub-tableau $\mathcal{T}'$ of $\mathcal{T}$, $\mathcal{M}, w \models \mathcal{T}'$. The same induction shows that, if the nodes in $\mathcal{T}'$ are labelled by n, then for every ground term t at depth n in the language of the tableau (i.e. symbols occurring in the tableau and dummy constants), $\mathcal{M}(w, t) \in \delta(w)$. This fact is needed when the substitution rule is applied.

The base case is immediate: the interpretation $\mathcal{M}$ is obtained from the model of the initial set of formulae given by the hypothesis, extending it in the obvious way to symbols annotated with 1 (the constant a_1^1 is mapped to any element of the domain of the initial world).

For the induction step, let $\mathcal{T}'$ be a satisfiable terminal sub-tableau of $\mathcal{T}$ (with nodes labelled by n), $\mathcal{M}$ a modal interpretation and w a world in $\mathcal{M}$ such that $\mathcal{M}, w \models \mathcal{T}'$. If $\mathcal{T}$ is expanded by application of an expansion rule to a leaf that is not in $\mathcal{T}'$, then:

- if the applied rule is not the δ^+-rule, then obviously $\mathcal{M}, w \models \mathcal{T}$ is still true.

– If the δ^+-rule is applied, introducing a new Skolem function f, then the interpretation $\mathcal{M}$ is extended to $\mathcal{M}'$ by establishing that $\mathcal{M}'(f) = \lambda(d_1, ..., d_k).d$, where d is any element in the domain of w. Since f does not occur in $\mathcal{T}'$, $\mathcal{M}', w \models \mathcal{T}'$. And it is still true that the interpretation of every ground term at depth n is in the domain of w.

The interesting cases are when $\mathcal{T}'$ is expanded to $\mathcal{T}''$. We consider the following two different cases, according to the applied rule.

1. The applied rule is not the substitution rule and the expanded node is $n : S$. If $\mathcal{T}'$ is expanded by means of a classical rule, the proof is the same as in the classical case, since $\mathcal{T}''$ is still a classical tableau. The only point to be taken care of is to ensure that, when the δ^+-rule is applied, introducing the Skolem function f^n, the interpretation of every ground term $f^n(t_1, ..., t_k)$ at depth n is in the domain of w. But this follows immediately from the induction hypothesis and the fact that $t_1, ..., t_k$ are at depth n too.
   Also the case of static modal rules are straightforward. If $\mathcal{T}'$ is expanded by application of a dynamic rule to $n : S$, $n : S$ contains no free-variables. So the proof of this case is the same as in the soundness proof of the ground tableau calculi presented in [3, 4].
2. The applied rule is the substitution rule. There, a Modal Substitution Theorem is applied, that, in QML, holds in the following form:
   > Let $\mathcal{M}$ be a modal interpretation, w a world in $\mathcal{M}$, s a variable assignment, and t a rigid term (i.e. a term containing no non-rigid symbol). If $s(w, t) = d$, then $\mathcal{M}, s^d_x, w \models A$ if and only if $\mathcal{M}, s, w \models A[t/x]$.

   Now, let us assume that $\mathcal{T}''$ is obtained from $\mathcal{T}'$ by application of the grounding substitution rule, with substitution σ. Then, if $v^n_1, ..., v^n_k$ is a sequence of all the free variables (all annotated with n) occurring in $\mathcal{T}'$, σ has the form $\{t_1/v_1, ..., t_k/v_k\}$, where $t_1, ..., t_k$ are ground terms at depth n.
   Let $\mathcal{M} = \langle W, w, R, D, \delta, \phi, \pi\rangle$ be a model of $\mathcal{T}'$. By definition, for all variable assignment s', if $s'(v^n) \in \delta(w)$ for all free-variable v^n annotated by n, then there exists a leaf $n : S$ of $\mathcal{T}'$ such that $\mathcal{M}, s', w \models S$. Moreover, by the induction hypothesis, for every ground term t at depth n, $\mathcal{M}(w, t) \in \delta(w)$.
   Let s be any variable assignment such that $s(v^n) \in \delta(w)$ for all free-variable v^n annotated by n. If t_j/v^n_j is any element of σ, since t_j is a ground term at depth n, $\mathcal{M}(w, t_j) \in \delta(w)$. Now, let $d_j = \mathcal{M}(w, t_j)$, for any $j = 1, ..., k$, and consider the assignment $s' = s^{d_1,...,d_k}_{v_1,...,v_k}$. Clearly, for all free variable v^n, $s'(v^n) \in \delta(w)$. As a consequence, by hypothesis there is a leaf $n : S$ in $\mathcal{T}'$ such that $\mathcal{M}, s', w \models S$. Since $s(w, t_i) = d_i = s'(v_i)$ and t_i is a rigid term, by the Modal Substitution Theorem $\mathcal{M}, s, w \models S\sigma$.

4 The GQML-Prover

The aim of the **GQML-Prover** is to detect whether a given sentence is a theorem of a given QML. Obviously, termination must be forced with the loss of completeness. The **GQML-Prover** exploits, to this end, two numerical bounds limiting the proof depth and entered by the user. The role of such bounds is explained below. Besides them, the user enters the sentence to prove, the propositional base of the considered logic, the domain variant, and the non-rigid functional symbols.

The **GQML-Prover** is implemented in Objective Caml. It is accessible on line at `http://www.lri.fr/~thion/Proto`. One just needs a standard web browser to use it. The Objective Caml program is compiled on the LRI[3] server. The user formulates a query by filling the form in the web page. A *php* function calls the execution of the program on the LRI server and the answer is given in a frame under the form. In order to help the user, a pre-defined set of sentences to test and help topics supplement the form. The tool will be part of a more important application allowing to test preservation of integrity constraints in data bases.

Like any tableau calculus, the system presented in Section 3 is intrinsically non-deterministic, since a node might be expanded by means of several rules. Hence, the crucial passage from a proof system to an algorithm consists in defining a fair and complete strategy guiding rule application.

Tableau building strategies have been proposed for different logics in [11, 5, 13, 9]. The strategy used in our implementation is partially inspired by these papers but is specifically designed to deal with our calculus.

A first classical choice in the strategy underlying the **GQML-Prover** consists in applying deterministic rules (in our case static rules) before a non-deterministic (a dynamic) one.

QML tableaux and tableaux systems for classical logic share the difficulty that, in principle, a γ-rule might need to be applied an unlimited number of times to the same formula (in each given world), in order to achieve completeness. A classical way to deal with this problem is to set an upper bound to the number of γ-rule applications to the same formula in a given world [7]. This is the role of the first bound entered by the user of the **GQML-Prover**: the γ-rule will never be applied more than K_γ times to the same formula at the same depth.

However, when K_γ is large, applying always K_γ times a γ-rule to the same formula in each world leads immediately to an explosion of the search space. In order to overcome this problem, at the first attempt to close a branch, the γ-rule is applied only once to the same formula in each world. If the branch cannot be closed, then the γ-rule is applied again to each formula, i.e. the number of γ-rule applications is increased as an effect of backtracking. The advantage of such a choice is that if a proof exists where the γ-rule is applied a number of times strictly inferior to K_γ, such a proof is found.

The second bound, which we here call K_π, limits the number of dynamic rule applications allowed in a single branch, which will never exceed $K_\pi - 1$.

In order to describe the proof search strategy, we introduce the following classification of the expansion rules of the system:

[3] Laboratoire de Recherche Informatique, *Université Paris-Sud*, France

- *Dynamic rules*, which introduce a new node label and are non-reversible rules; the dynamic rules are the π-rules of all the considered logics, and the ν-rule of system **D**.
- *Looping rules*, that may need to be triggered more than once on the same formula in the same world (i.e. at the same depth). The only looping rule is the γ-rule.
- *Branching rules*, that generates two branches. The only branching rule is the β-rule.
- *Simple rules:* all the others, i.e. static, non-looping and non-branching rules. These are all the rules α, δ^+ and ν_T.

Proof search in the **GQML-Prover** proceeds depth-first, and the construction of a single branch proceeds by stages, as follows:

Step 1: Initialize p with 0.
Step 2: Simple rules are applied as far as possible.
Step 3: For each γ-formula A resulting from the previous step, the γ-rule is applied once, if A has been expanded less than K_γ times at the same depth.
Step 4: Branching rules are applied as far as possible, and a branch is chosen to be expanded next (this is not a choice point: every branch must be expanded, sooner or later).
Step 5 (Choice point): either choose, if any, a substitution that closes the branch and apply it to the whole tableau, or go on to step 6.
Step 6 (Choice point): either go on to step 7, or go back to step 2 (if some formula can still be expanded either by simple rules or by the γ-rule).
Step 7: Increase p by one (one "world" has been explored).
Step 8 (Choice point): If $p \leq K_\pi$, choose a formula (if any) that can be expanded by means of a dynamic rule, expand it and go back to step 2.

Each iteration between steps 2 and 4 constitutes the construction of a *block*. The construction of a branch ends in any case when it contains (at most) K_γ applications for every universal formula at each depth, and (at most) $K_\pi - 1$ applications of dynamic rules (i.e. K_π different node labels). The attempt to close a branch fails when the branch cannot be expanded any more and no substitution can close it. A more detailed description of the algorithm is given in Figure 2.

The strategy described above is sound and complete, provided that, by iterative deepening, the bounds K_γ and K_π are incremented as far as needed (this corresponds to using an "unbounded" version of the strategy). The soundness and completeness proofs can be found in [14].

In order to illustrate the proof-search strategy, consider the formula $F \equiv \neg(\forall x \exists y(\neg p(x) \wedge p(y)) \ \wedge \ \Diamond r(a)) \wedge \Diamond(q(a) \rightarrow \Box q(a))$, where the constant a is rigid. F is a theorem of **S4**-QML with varying domains. Below, we show the construction, according to the strategy illustrated above, of a closed tableau rooted at $\neg F \equiv (\forall x \exists y(\neg p(x) \wedge p(y)) \wedge \Diamond r(a)) \vee \Box(q(a) \wedge \Diamond \neg q(a))$, when the bounds $K_\gamma = 2$ and $K_\pi = 2$.

PROVABLE (A) /* returns a boolean */
return IS-CLOSED ({INITIAL-TABLEAU ($\neg A$)}, 0)
/* INITIAL-TABLEAU returns the node representing the initial tableau for a formula */

IS-CLOSED (*nodes*, p)
/* *nodes* is the set of open leaves to expand, all having the same depth;
p is the number of "worlds" already explored in the branch ($p \leq K_\pi$) */
if $nodes = \emptyset$ **then return** true
else let N = an element of *nodes*
and $rest = nodes - \{N\}$
and *expansions* = EXPAND-A-BLOCK (N)
and $N1$ = an element of *expansions*
and $siblings = expansions - \{N1\}$
return
there exists a substitution σ that closes $N1$
such that IS-CLOSED (($siblings \cup rest$) σ, p)
or $p < K_\pi$
and there exists an expansion $N2$ of $N1$ by a dynamic rule
such that IS-CLOSED ($\{N2\}$, $p+1$)
and IS-CLOSED (($siblings \cup rest$) σ, p)
where σ is the substitution used to close $N2$
or $expansions \neq \{N\}$ /* if $expansions = \{N\}$ nothing happened when expanding a block in N */
and IS-CLOSED ($expansions \cup rest$, p)

EXPAND-A-BLOCK (N)
/* expands node N and returns all the nodes that can be obtained by application of simple rules, a single application of the γ-rule to each formula that has not yet reached the limit, and the β-rule */
let $N1$ = the node resulting from N by applying simple rules as far as possible
and $N2$ = the node resulting from $N1$ by expanding once more each γ-formula that has already been expanded less than K_γ times at this depth
return the leaves of the tableau obtained from $N2$ by applying the β-rule as far as possible

Fig. 2. The algorithm of the GQML-Prover, where K_π and K_γ are global variables, assumed to have a positive value.

The initial node is:

$$1 : (\forall x \exists y(\neg p(x) \wedge p(y)) \wedge \Diamond r(a^0)) \vee \Box(q(a^0) \wedge \Diamond \neg q(a^0))$$

Such a node can be expanded neither by simple rules nor by the γ-rule, while the β-rule can be applied:

$$\dfrac{1:\forall x\exists y(\neg p(x)\wedge p(y))\wedge\Diamond r(a^0)\vee\Box(q(a^0)\wedge\Diamond\neg q(a^0))}{1:\forall x\exists y(\neg p(x)\wedge p(y))\wedge\Diamond r(a^0)\qquad\qquad 1:\Box(q(a^0)\wedge\Diamond\neg q(a^0))}\;(\beta)$$

At first, one tries to close the first branch. When the first branch is closed, the second one will be treated (by depth-first search).

Dealing with the first branch. The node $1:\forall x\exists y(\neg p(x)\wedge p(y))\wedge\Diamond r(a^0)$ is expanded, and the tableau shown below is built. Note that the γ-rule is applied only once, and is not followed by an application of the α-rule, even if it would be possible, because, in a single block, the α-rule is always applied only before the γ-rule.

$$\dfrac{\dfrac{1:\forall x\exists y(\neg p(x)\wedge p(y))\wedge\Diamond r(a^0)}{1:\forall x\exists y(\neg p(x)\wedge p(y)),\ \Diamond r(a^0)}\;(\alpha)}{1:\forall x\exists y(\neg p(x)\wedge p(y)),\ \Diamond r(a^0),\ \exists y(\neg p(v_1^1)\wedge p(y))}\;(\gamma)$$

Since there are no closing substitutions for the leaf of this branch, the construction goes on with an application of the π_4-rule:

$$\begin{array}{c}\vdots\\ \dfrac{1:\forall x\exists y(\neg p(x)\wedge p(y)),\ \Diamond r(a^0),\ \exists y(\neg p(v_1^1)\wedge p(y))}{1:r(a^0)}\;(\pi_4)\end{array}$$

This branch cannot be expanded any further, and it cannot be closed. Thus, the algorithm backtracks before the application of the π_4-rule, the simple rules are applied again as far as possible and the γ-rule once again to each formula (no application of the β-rule is possible):

$$\begin{array}{c}\vdots\\ \dfrac{\dfrac{\dfrac{1:\forall x\exists y(\neg p(x)\wedge p(y)),\ \Diamond r(a^0),\ \exists y(\neg p(v_1^1)\wedge p(y))}{1:\forall x\exists y(\neg p(x)\wedge p(y)),\ \Diamond r(a^0),\ \neg p(v_1^1)\wedge p(f^1(v_1^1))}\;(\delta^+)}{1:\forall x\exists y(\neg p(x)\wedge p(y)),\ \Diamond r(a^0),\ \neg p(v_1^1),\ p(f^1(v_1^1))}\;(\alpha)}{1:\forall x\exists y(\neg p(x)\wedge p(y)),\ \Diamond r(a^0),\neg p(v_1^1),\ p(f^1(v_1^1)),\ \exists y(\neg p(v_2^1)\wedge p(y))}\;(\gamma)\end{array}$$

At the end of this second block, since again there are no closing substitutions, the π_4-rule is applied again, leading to a tableau that can be neither closed nor expanded further:

$$\begin{array}{c}\vdots\\ \dfrac{1:\forall x\exists y(\neg p(x)\wedge p(y)),\ \Diamond r(a^0),\neg p(v_1^1),\ p(f^1(v_1^1)),\ \exists y(\neg p(v_2^1)\wedge p(y))}{1:r(a^0)}\;(\pi_4)\end{array}$$

So the algorithm backtracks once more before the application of the π_4-rule and builds the third block. At this point, the γ-rule has already been applied twice to the formula $\forall x \exists y(\neg p(x) \wedge p(y))$ at the current depth (during the two previous blocks), so the K_γ bound has been reached and the γ-rule cannot be applied anymore to that formula (at this depth).

$$\dfrac{\dfrac{\begin{matrix}\vdots \\ 1 : \forall x \exists y(\neg p(x) \wedge p(y)),\ \Diamond r(a^0),\ \neg p(v_1^1),\ p(f^1(v_1^1)),\ \exists y(\neg p(v_2^1) \wedge p(y))\end{matrix}}{1 : \forall x \exists y(\neg p(x) \wedge p(y)),\ \Diamond r(a^0), \neg p(v_1^1),\ p(f^1(v_1^1)),\ \neg p(v_2^1) \wedge p(g^1(v_2^1))}\ (\delta^+)}{1 : \forall x \exists y(\neg p(x) \wedge p(y)),\ \Diamond r(a^0),\ \neg p(v_1^1),\ p(f^1(v_1^1)),\ \neg p(v_2^1),\ p(g^1(v_2^1))}\ (\alpha)$$

Finally, this branch is closed by the substitution $\sigma = \{g^1(v_2^1)/v_1^1\}$.

Dealing with the second branch. The application of σ to the rightmost branch leaves it unchanged, since there are no occurrences of v_1^1. The first block of the rightmost branch is:

$$\dfrac{\dfrac{1 : \Box(q(a^0) \wedge \Diamond \neg q(a^0))}{1 : q(a^0) \wedge \Diamond \neg q(a^0),\ \Box(q(a^0) \wedge \Diamond \neg q(a^0))}\ (\nu_T)}{1 : q(a^0),\ \Diamond \neg q(a^0),\ \Box(q(a^0) \wedge \Diamond \neg q(a^0))}\ (\alpha)$$

Since the leaf cannot be closed, the π_4-rule is applied and the branch is expanded by application of simple rules (now $p = 1$, so this is the last application of the π_4-rule in the branch):

$$\dfrac{\dfrac{\dfrac{\begin{matrix}\vdots \\ 1 : q(a^0),\ \Diamond \neg q(a^0),\ \Box(q(a^0) \wedge \Diamond \neg q(a^0))\end{matrix}}{2 : \neg q(a^0),\ \Box(q(a^0) \wedge \Diamond \neg q(a^0))}\ (\pi_4)}{2 : \neg q(a^0),\ q(a^0) \wedge \Diamond \neg q(a^0),\ \Box(q(a^0) \wedge \Diamond \neg q(a^0))}\ (\nu_T)}{2 : \neg q(a^0),\ q(a^0),\ \Diamond \neg q(a^0),\ \Box(q(a^0) \wedge \Diamond \neg q(a^0))}\ (\alpha)$$

Finally, also the second branch closes.

Note that if either $K_\pi < 2$ or $K_\gamma < 2$, no proof would be found.

5 Concluding Remarks

The addition of suitable rules handling equality to the general calculus presented in Section 3 presents a major difficulty, due to the nature of dynamic rules, which lead to a loss of information, in contrast with the fact that equality is to be interpreted as a rigid predicate symbol. To make this point clear, consider the following simple example. Let A be the formula $\exists x\ \exists y\ ((\Diamond x = y)\ \wedge\ (\Diamond x \neq y))$. Since equality is identity everywhere ($\mathcal{M}, s, w \models t_1 = t_2$ iff $s(t_1) = s(t_2)$), the formula A is unsatisfiable. In fact, if $s(x) = s(y)$ in a given world w, then it cannot be $s(x) \neq s(y)$ in w'. However, no matter which set of "reasonable" rules for $=$ (including, for instance, reflexivity and replacement) we add to our calculus, no tableau rooted at $1 : A$ closes, since, after two applications of the δ^+-rule and an application of the α-rule, one gets

$$1 : \Diamond\, a^1 = b^1,\ \Diamond\, a^1 \neq b^1$$

When this latter node is expanded by means of a π-rule, the information about one $\Diamond$-successor of the world named 1 is necessarily lost: two different expansions can be produced, but in two distinct tableaux, namely $2 : a^1 = b^1$ and $2 : a^1 \neq b^1$. Clearly, neither the first nor the second tableau closes.

This failure to capture equality (at least in a simple way) is intrinsic in the nature of "standard" (i.e. unprefixed) tableaux, because of the presence of non-reversible rules (the dynamic ones).

A natural way of overcoming such a difficulty is resorting to prefixed tableaux, in the style of [6], that represent a way of keeping track of several different tableaux in the same prefixed one. In other terms, node labels are replaced by prefixes on formulae. Prefixes, in turn, are structured objects, encoding the accessibility relation between possible worlds, and symbol annotations have the same structure. Of course, this would constitute an important depart from the framework of the proof systems proposed in this work.

References

1. B. Beckert and R. Goré. Free variable tableaux for propositional modal logics. In *Proc. of TABLEAUX'97*, pages 91–106. Springer, 1997.
2. S. Cerrito and M. Cialdea Mayer. Free-variable tableaux for constant-domain quantified modal logic with rigid and non-rigid designation. In *First Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, pages 137–151. Springer, 2001.
3. M. Cialdea Mayer and S. Cerrito. Variants of first-order modal logics. In *Proc. of TABLEAUX 2000*, pages 175–189. Springer, 2000.
4. M. Cialdea Mayer and S. Cerrito. Ground and free-variable tableaux for variants of quantified modal logics. *Studia Logica*, 69:97–131, 2001.
5. F. M. Donini and F. Massacci. EXPTIME tableaux for ALC. *Artificial Intelligence*, 124:87–138, 2000.
6. M. Fitting. *Proof Methods for Modal and Intuitionistic Logics.* Reidel, 1983.
7. M. Fitting. *First-Order Logic and Automated Theorem Proving.* Springer, 1996.
8. M. Fitting and R Mendelsohn. *First-Order Modal Logic.* Kluwer, 1998.
9. R. Goré. Automated reasoning project. Technical report, TR-ARP-15-95, 1997.
10. R. Hähnle and P. H. Schmitt. The liberalized δ-rule in free variable semantic tableaux. *Journal of Automated Reasoning*, 13:211–222, 1994.
11. U. Hustadt and R. A. Schmidt. Simplification and backjumping in modal tableau. In *Proc. of TABLEAUX'98*, pages 187–201. Springer, 1998.
12. Jens Otten. ileanTAP: An intuitionistic theorem prover. In *Proc. of TABLEAUX'97*, pages 307–312. Springer, 1997.
13. J. Posegga and P. Schmitt. Implementing semantic tableaux. In M. D'Agostino, G. Gabbay, R. Hähnle, and J. Posegga, editors, *Handbook of tableau method*, pages 581–629. Kluwer, 1999.
14. V. Thion. A strategy for free variable tableaux for variant of quantified modal logics. Technical report, L.R.I., 2002. `http://www.lri.fr/~thion`.
15. L. A. Wallen. *Automated Deduction in Nonclassical Logics: Efficient Matrix Proof Methods for Modal and Intuitionistic Logics.* MIT Press, 1990.