Journal of Automated Reasoning manuscript No.
(will be inserted by the editor)

A Prover Dealing with Nominals, Binders, Transitivity and Relation Hierarchies

Marta Cialdea Mayer

Abstract This work describes the Sibyl prover, an implementation of a tableau based proof procedure for multi-modal hybrid logic with the converse, graded and global modalities, and enriched with features largely used in description logics: transitivity and relation hierarchies. The proof procedure is provably terminating when the input problem belongs to an expressive decidable fragment of hybrid logic.

After a description of the implemented proof procedure, the way how the implementation deals with the most delicate aspects of the calculus is explained. Some experimental results, run on sets of randomly generated problems as well as some hand-tailored ones, show only a moderate deterioration in the performances of the prover when the number of transitivity and inclusion axioms increase. Sibyl is compared with other provers (HTab, the hybrid logic prover whose expressive power is closer to Sibyl's one, and the first-order prover SPASS). The obtained results show that Sibyl has reasonable performances.

Keywords Automated Proof Systems · Tableaux · Modal Logic · Hybrid Logic

1 Introduction

Hybrid languages extend ordinary modal logic with the possibility of naming and accessing states of a structure explicitly (see, for instance, [6]). Their main distinguishing feature is represented by special atomic propositions, called *nominals*, which give names to states: a nominal is true in exactly one state of the model. The two operators specific of hybrid languages are the *satisfaction operator* and the *binder*. The former allows one to jump to a state named by a nominal, regardless of the accessibilities in the structure: if a is a nominal, a formula of the form $a{:}\,F$ states that F holds at the state named a. For instance,

M. Cialdea Mayer
Dipartimento di Ingegneria, Università degli Studi Roma Tre, Italy

if b is also a nominal, $a\colon(\Diamond b \wedge \Box b)$ means that the state named b is the only state related to a (hence, in particular, a is not related to itself). The *binder* ($\downarrow$) dynamically binds *state variables* to states so that they can be referred to later on: $\downarrow x.F$ states that F holds at a state w when the variable x occurring in F is interpreted as w. For example, $\downarrow x.\Box\neg x$ holds at any state that is not related to itself.

Other modal operators can be added to the basic hybrid language, such as the converse modalities ($\Diamond^-$ and $\Box^-$) and the global ones (E and A). Moreover, hybrid languages can rely on a multi-modal base, allowing for modelling structures with different accessibility relations. In this case, the basic modalities $\Diamond$ and $\Box$ (and their converses, if present) are indexed by relation symbols.

Some examples of interesting formulae that can be expressed in hybrid languages are given in Figure 1.

(1) $\downarrow x.\Diamond\downarrow y_1.(x : \Diamond(\neg y_1 \wedge \downarrow y_2.(x : \Diamond(\neg y_1 \wedge \neg y_2))))$	(the current state is related to at least 3 different states)
(2) $\downarrow x.\Diamond^-\Diamond\neg x$	(the current state has at least a sibling)
(3) $\downarrow x.\Diamond_r(F \wedge \downarrow y.x\colon \Diamond_r(F \wedge \neg y))$	(the current state is r-related to at least two different states where F holds)
(4) $\mathsf{E}\downarrow x_1 \ldots \mathsf{E}\downarrow x_n.\mathsf{A}(x_1 \vee \cdots \vee x_n)$	(there exist at most n states)
(5) $\mathsf{A}\downarrow x.\Diamond_r x$	(r is reflexive)
(6) $\downarrow x.\Diamond_r\Diamond_s\Box_s^-\neg x$	(the current state is a stepmother/stepfather: r is "married to" and s is "has child" [34])
(7) $\mathsf{E}\downarrow x.(\Box\neg x \wedge \mathsf{A}(x \vee \downarrow y.x\colon \Diamond y))$	(there exists a spy point)
(8) $\mathsf{A}\downarrow x.\Box_r\neg x$	(r is irreflexive)
(9) $\mathsf{A}\downarrow x.\Box_r\Diamond_r x$	(r is symmetric)
(10) $\mathsf{A}\downarrow x.\Box_r\Box_r\neg x$	(r is asymmetric)
(10) $\mathsf{A}\downarrow x.\Box(\Diamond x \rightarrow x)$	(r is antisymmetric)
(11) $\mathsf{A}\downarrow x.\Box_r\Box_r\downarrow y.x\colon \Diamond y$	(r is transitive)
(12) $\mathsf{A}\downarrow x.\Box_r\Box_r\Diamond_r^- x$	(r is transitive, with converse)
(13) $\mathsf{A}(\downarrow x.\downarrow y.(x\colon \Diamond_r y \rightarrow x\colon \Diamond_s y))$	(r is a subrelation of s)
(14) $\mathsf{A}(\downarrow x.\Box_r\Diamond_s^- x)$	(r is a subrelation of s, with converse)
(15) $\Box_r F \vee \downarrow x.\Diamond_r\downarrow y_0.x\colon \Diamond_r\downarrow y_1.$ $x\colon \Box_r(F \vee y_0 \vee y_1)$	(the current world is r-related to at most two different states where F does not hold)

Fig. 1 Sample hybrid logic formulae

In this work, basic hybrid logic (with nominals only, beyond the modal operators $\Diamond$ and $\Box$) will be denoted by $\mathcal{H}$, and basic multi-modal hybrid logic by $\mathcal{H}_m$. Logics extending $\mathcal{H}$ or $\mathcal{H}_m$ with operators $O_1, \ldots, O_n$ (and their duals) are denoted by $\mathcal{H}(O_1, \ldots, O_n)$ and $\mathcal{H}_m(O_1, \ldots, O_n)$, respectively. In these contexts, the satisfaction operator (for which this work uses an infix colon) will be denoted by the symbol @ (an often used alternative notation for $a\colon F$ is in fact $@_a F$).

The satisfiability problem for formulae of any hybrid logic $\mathcal{H}(O_1, \ldots, O_n)$ or $\mathcal{H}_m(O_1, \ldots, O_n)$, where $O_i \in \{@, \Diamond^-, \mathsf{E}\}$, is decidable [6]. Unfortunately, due to the high expressive power of the binder, $\mathcal{H}(\downarrow)$ is undecidable [1,8].

There are both semantic and syntactic restrictions allowing for regaining decidability of hybrid logic with the binder. Restricting the frame class is a way of restoring decidability (see, for instance [35,36,43]). Decidable subclasses of hybrid logic formulae can be also identified by means of syntactic restrictions. In particular, [43] proves that the satisfiability problem for formulae in $\mathcal{H}(@, \downarrow, \mathsf{E}, \Diamond^-)$ is decidable, provided that their negation normal form (NNF) does not contain the pattern $\Box\downarrow\Box$, i.e. it contains no universal operator (either $\Box$ or $\Box^-$ or A) scoping over a binder, that in turn has scope over a universal operator. Such a fragment of hybrid logic is denoted by $\mathcal{H}(@, \downarrow, \mathsf{E}, \Diamond^-) \setminus \Box\downarrow\Box$. The result is proved by showing that there exists a satisfiability preserving translation of $\mathcal{H}(@, \downarrow, \mathsf{E}, \Diamond^-) \setminus \Box\downarrow\Box$ into $\mathcal{H}(@, \downarrow, \mathsf{E}, \Diamond^-) \setminus \downarrow\Box$, i.e. the set of formulae in NNF where no universal operator occurs in the scope of a binder. The standard translation of hybrid logic into first order classical logic [1,43] maps, in turn, formulae in $\mathcal{H}(@, \downarrow, \mathsf{E}, \Diamond^-) \setminus \downarrow\Box$ into universally guarded formulae, that have a decidable satisfiability problem [22]. The satisfiability problem of both fragments $\mathcal{H}(@, \downarrow, \mathsf{E}, \Diamond^-) \setminus \downarrow\Box$ and $\mathcal{H}(@, \downarrow, \mathsf{E}, \Diamond^-) \setminus \Box\downarrow\Box$ is in 2ExpTime [43]. Decidability of $\mathcal{H}_m(@, \downarrow, \mathsf{E}, \Diamond^-) \setminus \Box\downarrow\Box$ can be proved by the same reasoning.

Among the formulae of figure 1, the first 7 ones do not contain the pattern $\Box\downarrow\Box$ (also prefixing the first 3 ones by the global modality A leaves them in the fragment), while the others do not.

Considering that multi-modal languages are notational variants of terminological logics [40] and that restricted uses of the binder are of interest also in the context of description logics (see for instance [26,34]), further important extensions of the expressive power of $\mathcal{H}_m(@, \downarrow, \mathsf{E}, \Diamond^-) \setminus \Box\downarrow\Box$ are the graded modalities (corresponding to description logic qualified number restrictions) and the possibility of dealing with relation hierarchies and transitive relations, still leaving the satisfiability problem decidable. The examples in Figure 1 show that the existential graded modality can be expressed by a formula belonging to the considered fragment (formula 3), but the formulae expressing transitivity (11 and 12), relation inclusion (13 and 14) and the universal graded modality (15) contain the pattern $\Box\downarrow\Box$. As far as relation hierarchies and transitive relations are concerned, they can be treated by means of specific axioms, like in description logics: in this work, we declare a relation r to be transitive by use of the notation $\mathsf{Trans}(r)$, while inclusion assertions have either the form $s \sqsubseteq r$ (s is a sub-relation of r) or $s^- \sqsubseteq r$ (the converse of s is a sub-relation of r).

Even when the pattern $\Box\downarrow\Box$ is not allowed, the graded modalities $\Diamond^n_R$ and $\Box^n_R$ (where R is either a relation of a converse one) cannot be added to the considered logic without endangering decidability: in [16] it is shown that the satisfiability problem for hybrid logic with either the satisfaction operator or the converse modalities, functional restrictions and binders, without the critical pattern $\Box\downarrow\Box$, is undecidable. However, decidability can be preserved by placing additional syntactical restrictions on the occurrences of the graded modalities. The details on such restrictions will be given in Section 2.

The language $\mathcal{H}_m(@, \downarrow, \mathsf{E}, \Diamond^-, \Diamond^n, \mathsf{Trans}, \sqsubseteq)$, obtained by adding the graded modalities, transitivity and relation inclusion assertions to $\mathcal{H}_m(@, \downarrow, \mathsf{E}, \Diamond^-)$,

will be shortly denoted by HL, and HL $\setminus \Box\downarrow\Box$ is used to denote the fragment of HL of formulae whose NNF does not contain the pattern $\Box\downarrow\Box$ and the graded modalities only occur in the allowed forms.

A sound and complete satisfiability decision procedure for HL $\setminus \Box\downarrow\Box$ is defined in [16] (extending [12–14], where proof procedures for sublogics of HL $\setminus \Box\downarrow\Box$ are introduced), thus showing that this rich fragment of hybrid logic, subsuming the description logic $\mathcal{SHOIQ}$, is decidable. This work describes the Sibyl prover, an implementation of the above mentioned proof procedure.

The first implementation of a prover for hybrid logic is the resolution based prover HyLoRes [4], which dates back to 2001. It originally dealt with $\mathcal{H}(@,\downarrow)$, and was then extended to include the converse and global modalities and the binder. However, the implemented algorithm is terminating for formulae in $\mathcal{H}(@,\Diamond^-)$ but not $\mathcal{H}(@,\mathsf{E})$. In 2002, [46] presents HyLoTab, an implementation of the tableau calculus described in [45], dealing with $\mathcal{H}(@,\mathsf{E},\Diamond^-,\downarrow)$. Unfortunately, the rules implemented by HyLoTab do not guarantee termination even for decidable sublogics like $\mathcal{H}(@,\mathsf{E})$. Both HyLoRes and HyLoTab are described by their authors as prototypes. Also the two provers Herod and Pilate [17] are prototypal systems. They deal only with $\mathcal{H}(@)$, and were implemented in 2010 to compare the performances of different approaches to nominal equalities in hybrid logic tableaux [10,11]. The first prover for hybrid logic based on a tableau system guaranteeing termination [9] is HTab [25], developed since 2007, that originally dealt with $\mathcal{H}(@,\mathsf{E})$. It was then extended to treat a much richer language, including the binder, the global modalities, as well as transitive relations and relation hierarchies [24]. In 2010, the prover Spartacus [21] was released, a tableau prover for $\mathcal{H}_m(@,\mathsf{E})$ based on the calculus presented in [32] and supporting also reasoning in the presence of reflexive and transitive relations.

This work is organized as follows. The syntax and semantics of HL are defined in Section 2, while Sections 3 and 4 are devoted to present the theoretical foundations of the prover. The main guidelines of the implementation are presented in Section 5 (more details can be found in Appendix A) and the experimental results collected so far are described in Section 6. Finally, some directions for future work are outlined in Section 7.

2 Syntax and Semantics of Multi-Modal Hybrid Logic with Transitivity and Inclusion Axioms

The language of HL is based on a set PROP of propositional letters, a set NOM of nominals, an infinite set VAR of state variables, and a set REL of relation symbols (all such sets being mutually disjoint). If $r \in \mathsf{REL}$ is a relation symbol, the corresponding uppercase letter, R, denotes a *relation*, i.e. either r itself (a *forward relation*) or its converse r^- (a *backward relation*). A backward relation r^- is intended to denote the set of pairs of states $\langle w, w'\rangle$ such that $\langle w', w\rangle$ is in the relation denoted by r. In order to allow for more compact notations, $\Box_{r^-}$

and $\Diamond_{r^-}$ will be used as synonyms of $\Box_r^-$ and $\Diamond_r^-$, respectively, and analogously for the graded modalities.

Well-formed expressions of HL are partitioned into two categories: *formulae* and *assertions*. *Formulae* are defined by the following grammar:

$$\begin{aligned} F := p \mid u \mid \neg F \mid F \wedge F \mid F \vee F \mid u{:}\, F \mid \downarrow x.F \\ \mid \Diamond_R F \mid \Box_R F \mid \Diamond_R^n F \mid \Box_R^n F \mid \mathsf{E}F \mid \mathsf{A}F \end{aligned}$$

where $p \in \mathsf{PROP}$, $u \in \mathsf{NOM} \cup \mathsf{VAR}$, $x \in \mathsf{VAR}$, $n \in \mathbb{N}$ and R is either a forward or backward relation. Lowercase letters from the beginning of the alphabet will be used as meta-variables for nominals, while x, y, z are used for state variables.

A *ground* formula is a formula where every occurrence of a state variable x occurs in the scope of a $\downarrow x$. A *ground satisfaction statement* is a formula of the form $a{:}\, F$, where F is a ground formula. The nominal a in a satisfaction statement $a{:}\, F$ is called the *outermost nominal* of the formula, and F is its *body*.

If F is a formula, x a state variable and a a nominal, then $F[a/x]$ denotes the formula obtained from F by substituting a for every free occurrence of x (an occurrence of x is free if it is not in the scope of a $\downarrow x$). If $a_0, \ldots, a_n, b_0, \ldots, b_n$ are nominals, then $F[b_0/a_0, \ldots, b_n/a_n]$ denotes the formula obtained from F by simultaneously replacing b_i for every occurrence of a_i.

Assertions are either *transitivity assertions,* of the form $\mathsf{Trans}(r)$, for $r \in \mathsf{REL}$, or *inclusion assertions,* of either form $r \sqsubseteq s$ or $r^- \sqsubseteq s$, for $r, s \in \mathsf{REL}$. Note that inverse relations are allowed only on the left of the $\sqsubseteq$ symbol. This is only a syntactical restriction, since $r^- \sqsubseteq s^-$ is equivalent to $r \sqsubseteq s$, and $r \sqsubseteq s^-$ is equivalent to $r^- \sqsubseteq s$. If s is a relation and R either a forward or a backward one, the notation $R \sqsubseteq s^-$ will be used as an abbreviation for the equivalent inclusion assertion.

An *interpretation* $\mathcal{M}$ of an HL language is a tuple $\langle W, \rho, N, I \rangle$ where W is a non-empty set (whose elements are the *states* of the interpretation), ρ is a function mapping every $r \in \mathsf{REL}$ to a binary relation on W ($\rho(r) \subseteq W \times W$), N is a function $\mathsf{NOM} \to W$ and I a function $W \to 2^{\mathsf{PROP}}$. The following abbreviation will be used:

$$wRw' = \begin{cases} \langle w, w' \rangle \in \rho(r) \text{ if } R = r \text{ is a forward relation} \\ \langle w', w \rangle \in \rho(r) \text{ if } R = r^- \text{ is a backward relation} \end{cases}$$

If $\mathcal{M} = \langle W, \rho, N, I \rangle$ is an interpretation, $w \in W$, σ is a variable assignment for $\mathcal{M}$ (i.e. a function $\mathsf{VAR} \to W$) and F is a formula, the relation $\mathcal{M}_w, \sigma \models F$ is defined adding the following clauses to the usual definition of the classical operators:

1. $\mathcal{M}_w, \sigma \models p$ if $p \in I(w)$, for $p \in \mathsf{PROP}$.
2. $\mathcal{M}_w, \sigma \models a$ if $N(a) = w$, for $a \in \mathsf{NOM}$.
3. $\mathcal{M}_w, \sigma \models x$ if $\sigma(x) = w$, for $x \in \mathsf{VAR}$.
4. $\mathcal{M}_w, \sigma \models a{:}\, F$ if $\mathcal{M}_{N(a)}, \sigma \models F$, for $a \in \mathsf{NOM}$.

5. $\mathcal{M}_w, \sigma \models x{:}\, F$ if $\mathcal{M}_{\sigma(x)}, \sigma \models F$, for $x \in \mathsf{VAR}$.
6. $\mathcal{M}_w, \sigma \models \downarrow x.F$ if $\mathcal{M}_w, \sigma^w_x \models F$, where σ^w_x is the variable assignment such that $\sigma^w_x(x) = w$ and, for $y \neq x$, $\sigma^w_x(y) = \sigma(y)$.
7. $\mathcal{M}_w, \sigma \models \Box_R F$ if for every w' such that wRw', $\mathcal{M}_{w'}, \sigma \models F$.
8. $\mathcal{M}_w, \sigma \models \Diamond_R F$ if there exists w' such that wRw' and $\mathcal{M}_{w'}, \sigma \models F$.
9. $\mathcal{M}_w, \sigma \models \Diamond^n_R F$ iff there are at least $n+1$ distinct states $w_1, ..., w_n$ such that wRw_i and $\mathcal{M}_{w_i}, \sigma \models F$.
10. $\mathcal{M}_w, \sigma \models \Box^n_R F$ iff there are at most n distinct states $w_1, ..., w_n$ such that wRw_i and $\mathcal{M}_{w_i}, \sigma \not\models F$.
11. $\mathcal{M}_w, \sigma \models \mathsf{A}F$ if $\mathcal{M}_{w'}, \sigma \models F$ for all $w' \in W$.
12. $\mathcal{M}_w, \sigma \models \mathsf{E}F$ if $\mathcal{M}_{w'}, \sigma \models F$ for some $w' \in W$.

Two formulae F and G are logically equivalent when, for every interpretation $\mathcal{M}$, assignment σ and state w of $\mathcal{M}$: $\mathcal{M}_w, \sigma \models F$ if and only if $\mathcal{M}_w, \sigma \models G$. Every formula in HL is logically equivalent to a formula in negation normal form (NNF), where negation appears only in front of atoms. Therefore, considering only formulae in NNF does not restrict the expressive power of the language.

If $\mathcal{A}$ is a set of assertions, an interpretation $\langle W, \rho, N, I\rangle$ is a model of $\mathcal{A}$ if:

1. for all $r \in \mathsf{REL}$ such that $\mathsf{Trans}(r) \in \mathcal{A}$, $\rho(r)$ is a transitive relation;
2. for all $r, s \in \mathsf{REL}$, if $r \sqsubseteq s \in \mathcal{A}$, then $\rho(r) \subseteq \rho(s)$;
3. for all $r, s \in \mathsf{REL}$ and all $w, w' \in W$, if $r^- \sqsubseteq s \in \mathcal{A}$ and $\langle w, w'\rangle \in \rho(r)$, then $\langle w', w\rangle \in \rho(s)$.

If F is a formula and $\mathcal{A}$ a set of assertions, $\{F\} \cup \mathcal{A}$ is satisfiable if there exist a model $\mathcal{M}$ of $\mathcal{A}$, a variable assignment σ for $\mathcal{M}$ and a state w of $\mathcal{M}$, such that $\mathcal{M}_w, \sigma \models F$.

This section concludes with the precise definition of the fragment $\mathsf{HL} \setminus \Box\downarrow\Box$. Let universal modalities be the operators $\Box_R$, $\Box^n_R$ and A. A formula F in NNF belongs to $\mathsf{HL} \setminus \Box\downarrow\Box$ if:

1. F contains no universal modality in the scope of a binder, that is in turn in the scope of a universal operator.
2. no subformula $\Box^n_R G$ of F occurs in the scope of any universal modality;
3. no subformula $\Diamond^n_R G$ of F is both in the scope and contains in its scope a universal modality.

3 The Preprocessing Step of the Satisfiability Decision Procedure

The calculus implemented by Sibyl (presented in Section 4) applies to formulae without graded modalities and whose NNF does not contain any universal operator in the scope of the binder, i.e. belongs to the fragment $\mathsf{HL} \setminus \downarrow\Box$. Any formula in $\mathsf{HL} \setminus \Box\downarrow\Box$ can however be translated into an equisatisfiable formula satisfying the required constraints. This section is devoted to describe such a preprocessing step.

In order to eliminate graded modalities, every subformula of the form $\Diamond^n_R$ or $\Box^n_R$ is replaced by $(\Diamond^n_R F)^*$ or $(\Box^n_R F)^*$, respectively, as defined in Figure 2.

Intuitively, $\mu_R(x, F, n, W)$ states that there are at least $n+1$ pairwise distinct states $y_0, \ldots, y_n$ where F holds and such that for all i, xRy_i and $y_i \notin W$. Similarly, $\nu_R(x, F, n, W)$ states that there are (not necessarily pairwise distinct) states $y_1, \ldots, y_n$ such that xRy_i and, for every state z such that xRz, if $z \notin W \cup \{y_1, \ldots, y_n\}$ then F holds at z.

$$
\begin{array}{ll}
(\Diamond_R^0 F)^* & \equiv_{def} \Diamond_R F \\
(\Diamond_R^{n+1} F)^* & \equiv_{def} \downarrow x.\mu_R(x, F, n+1, \emptyset) \\
\mu_R(x, F, 0, W) & \equiv_{def} x{:}\,\Diamond_R(F \wedge \bigwedge_{y \in W} \neg y) \\
\mu_R(x, F, n+1, W) & \equiv_{def} x{:}\,\Diamond_R(F \wedge \bigwedge_{z \in W} \neg z \wedge \downarrow y.\mu_R(x, F, n, W \cup \{y\})) \\
 & \quad \text{(where } y \text{ is a fresh variable)} \\
 & \\
(\Box_R^0 F)^* & \equiv_{def} \Box_R F \\
(\Box_R^{n+1} F)^* & \equiv_{def} \Box_R F \vee \downarrow x.\nu_R(x, F, n+1, \emptyset) \\
\nu_R(x, F, 0, W) & \equiv_{def} x{:}\,\Box_R(F \vee \bigvee_{y \in W} y) \\
\nu_R(x, F, n+1, W) & \equiv_{def} x{:}\,\Diamond_R(\downarrow y.\nu_R(x, F, n, W \cup \{y\})) \\
 & \quad \text{(where } y \text{ is a fresh variable)}
\end{array}
$$

Fig. 2 Translation of the graded modalities

For instance,

$$(\Diamond_R^2 F)^* = \downarrow x.\Diamond_r(F \wedge \downarrow y_0.x{:}\,\Diamond_r(F \wedge \neg y_0 \wedge \downarrow y_1.x{:}\,\Diamond_r(F \wedge \neg y_0 \wedge \neg y_1)))$$

$$\text{and } (\Box_R^3 F)^* = \Box_r F \vee \downarrow x.\Diamond_r \downarrow y_0.x{:}\,\Diamond_r \downarrow y_1.x{:}\,\Diamond_r \downarrow y_2.x{:}\,\Box_r(F \vee y_0 \vee y_1 \vee y_2)$$

It can be shown that, if F^* is obtained from F by eliminating the graded modalities as described above, then F is equivalent to F^*. Moreover, if F belongs to $\mathsf{HL} \setminus \Box\downarrow\Box$, then so does F^* [16]. However, when $n > 0$, $(\Diamond_R^n F)^*$ contains the binder, so that formulae in the fragment $\mathsf{HL} \setminus \Box\downarrow\Box$ must not contain subformulae of the form $\Diamond_R^n F$ in the scope of a universal modality, when F contains a universal modality too. Analogously, $(\Box_R^n F)^*$ contains the pattern $\downarrow\Box$, so that, in order for a formula to belong to $\mathsf{HL} \setminus \Box\downarrow\Box$, it must not contain a $\Box_R^n F$ in the scope of a universal modality. This explains the restrictions on the occurrences of the graded modalities given at the end of Section 2.

The second preprocessing step consists in converting the formula F obtained in the previous step into an equisatisfiable formula $\tau(F)$ in $\mathsf{HL} \setminus \downarrow\Box$ (the subfragment of $\mathsf{HL} \setminus \Box\downarrow\Box$ where no universal modality occurs in the scope of a binder), by means of the multi-modal analogous of the (polynomial) satisfiability preserving translation given in [43]. The mapping τ is defined in Figure 3, where fresh nominals are nominals that not only do not occur in the formula to be translated, but also are used nowhere else in the translation. For instance:

$$\begin{array}{l}
\tau((\mathsf{A}\downarrow x.\Diamond_r x) \wedge ((\downarrow y.\Box_r y) \vee (\downarrow z.\mathsf{A}z))) \\
= (\mathsf{A}\downarrow x.\Diamond_r x) \wedge ((a_1 \wedge \Box_r a_1) \vee (a_2 \wedge \mathsf{A}a_2))
\end{array}$$

Assuming that F does not contain the pattern $\Box\downarrow\Box$, and $\downarrow x.G$ is a a subformula of F, if G contains a universal operator, then $\downarrow x.G$ does not occur in

$$
\begin{array}{rcl}
\tau(u\colon F) & = & u\colon \tau(F) \quad \text{where } u \in \mathsf{NOM} \cup \mathsf{VAR} \\
\tau(F \wedge G) & = & \tau(F) \wedge \tau(G) \\
\tau(F \vee G) & = & \tau(F) \vee \tau(G) \\
\tau(\Diamond_R F) & = & \Diamond_R \tau(F) \\
\tau(\mathsf{E}F) & = & \mathsf{E}\,\tau(F) \\
\tau(\downarrow x.F) & = & \begin{cases} \downarrow x.F & \text{if } F \text{ contains no universal operator} \\ b \wedge \tau(F[b/x]) & (\text{where } b \text{ is a fresh nominal}) \text{ otherwise} \end{cases} \\
\tau(F) & = & F \quad \text{in all the other cases}
\end{array}
$$

Fig. 3 Satisfiability preserving translation of $\mathsf{HL} \setminus \Box\downarrow\Box$ into $\mathsf{HL} \setminus \downarrow\Box$

the scope of a binder in F. Therefore $\tau(\downarrow x.G)$ is a kind of skolemization inside F of $\downarrow x.G$. If on the contrary G does not contain universal operators, then the subformula $\downarrow x.G$ cannot be responsible of the critical pattern in F and is left unchanged.

4 The Tableau Calculus Underlying Sibyl

This section is devoted to describe the tableau calculus implemented by Sibyl. The peculiarity of the proof procedure is that termination is ensured whenever the input formula does not contain graded modalities and its NNF belongs to the fragment $\mathsf{HL}\setminus\downarrow\Box$.[1] In particular, tableau construction terminates whenever the input formula is obtained from a formula in $\mathsf{HL} \setminus \Box\downarrow\Box$ by means of the preprocessing step described in Section 3. It is worth pointing out that also the completeness proof (that extends the corresponding one given in [13] and whose details can be found in [15]) relies on the same assumption guaranteeing termination.

The presentation will be as self-contained as possible, therefore it overlaps with the descriptions given in [13,14,16] in many points. However, in order to minimize repetitions and considering that the main goal of this work is the presentation of the Sibyl prover, being the calculus itself introduced elsewhere, explanations on the more subtle notions will be essentially given on *what* are things like and not *why*, since a deep understanding of such reasons falls outside the scope of the present paper and can often be grasped only by inspecting the termination and completeness proofs. Analogously, a comparison with other approaches and several examples can be found in the above cited papers presenting the proof procedure and will not be repeated here.

4.1 Tableaux

A *tableau branch* is a sequence of *nodes* $n_0, n_1, \ldots$, where each node is labelled either by an assertion or a ground satisfaction statement whose body is a formula in NNF. If a node n occurs before m in a branch, we write $n < m$. The label of the node n is denoted by $\mathrm{label}(n)$. The notation $(n)\, a\colon F$ is used

[1] Note that $\mathsf{HL} \setminus \downarrow\Box$ is a proper subset of $\mathsf{HL} \setminus \Box\downarrow\Box$.

to denote the node n, and simultaneously say that its label is $a\colon F$. If a node $(n)\, a : F$ is in a branch, then the nominal a is said to label the formula F in the branch.

Statements of the form $a\colon \Diamond_r b$, where a and b are nominals and r is a forward relation are called *relational formulae*, and nodes labelled by relational formulae are called *relational nodes*. Expressions of the form $a \Rightarrow_R b$ will be used as abbreviations for relational formulae:

$$a \Rightarrow_R b \equiv_{def} \begin{cases} a\colon \Diamond_r b \text{ if } R = r \\ b\colon \Diamond_r a \text{ if } R = r^- \end{cases}$$

By convention, an expression of the form $\mathsf{Trans}(R)$, where R is a meta-symbol standing for either a forward or backward relation, will stand for $\mathsf{Trans}(r)$, where $r \in \mathsf{REL}$ is the relation symbol in R.

A tableau is a set of branches. A tableau for $\{F\} \cup \mathcal{A}$ is initialized with a single branch, constituted by the node $(n_0)\, a_0\colon F$, where a_0 is a fresh nominal, followed by nodes labelled by the assertions in $\mathcal{A}$ and then expanded according to the *Assertion rules* of Table 1. These rules complete the inclusion assertions in $\mathcal{A}$ by the reflexive and transitive closure of $\sqsubseteq$. The sequence of nodes $n_0, \ldots, n_k$ obtained so far is the *initial segment* of any tableau branch and the formula $a_0\colon F$ is the *initial formula* of the tableau.

$$\frac{}{r \sqsubseteq r}\ \mathsf{Rel}_0 \qquad \frac{R \sqsubseteq S \quad S \sqsubseteq P}{R \sqsubseteq P}\ \mathsf{Rel}$$

Table 1 Assertion rules

A tableau branch is expanded by either adding nodes or changing node labels, according to the rules in Table 2. Most rules are standard [9,10,28, 29,31,33], and their reading is standard too. Note that when the formulation of a rule contains (uppercase) relations, it actually stands for different rules, according to the relations' signs.

In applications of either the $\Diamond$ or the E rule, the nominal b occurring in the conclusion(s) is fresh in the branch. Moreover, the $\Diamond$ rule is not applicable to relational nodes, i.e. when R is a forward relation and F is a nominal. In applications of the A rule, the nominal b is any nominal occurring in the branch.

Nominal equalities, i.e. formulae of the form $a\colon b$, for $a, b \in \mathsf{NOM}$, are treated by means of substitution, like in [10,11,17,45]. The *equality rule* (=) does not add any node to the branch, but modifies the labels of its nodes. The schematic formulation of this rule in Table 2 indicates that it can be fired whenever a branch $\mathcal{B}$ contains a *nominal equality* of the form $a\colon b$ (with $a \neq b$); as a result of the application of the rule, every node label F in $\mathcal{B}$ is replaced by $F[b/a]$. It is worth pointing out that, by the effect of substitution,

$$\frac{(n)\, a\colon (F \wedge G)}{\begin{array}{l}(m_0)\, a\colon F\\ (m_1)\, a\colon G\end{array}}\ (\wedge) \qquad \frac{(n)\, a\colon (F \vee G)}{(m_0)\, a\colon F \quad | \quad (m_1)\, a\colon G}\ (\vee)$$

$$\frac{(n)\, a\colon b\colon F}{(m)\, b\colon F}\ (@) \qquad \frac{(n)\, a\colon \downarrow x.F}{(m)\, a\colon F[a/x]}\ (\downarrow)$$

$$\frac{(n)\, a\colon \Box_R F \quad (m)\, a \Rightarrow_R b}{(k)\, b\colon F}\ (\Box) \qquad \frac{(n)\, a\colon \Diamond_R F}{\begin{array}{l}(m_0)\, a \Rightarrow_R b\\ (m_1)\, b\colon F\end{array}}\ (\Diamond)$$

$$\frac{(n)\, a\colon \mathsf{A}F}{(m)\, b\colon F}\ (\mathsf{A}) \qquad \frac{(n)\, a\colon \mathsf{E}F}{(m)\, b\colon F}\ (\mathsf{E})$$

$$\frac{\begin{array}{c}[\mathcal{B}]\\ (n)\, a\colon b\end{array}}{\mathcal{B}[b/a]}\ (=) \qquad \frac{(m)\, a \Rightarrow_R b \quad (i)\, R \sqsubseteq S}{(k)\, a \Rightarrow_S b}\ (\mathsf{Link})$$

$$\frac{(n)\, a\colon \Box_S F \quad (m)\, a \Rightarrow_R b \quad (t)\, \mathsf{Trans}(R) \quad (i) R \sqsubseteq S}{(k)\, b\colon \Box_R F}\ (\mathsf{Trans})$$

Table 2 Expansion rules

distinct node labels may become equal, though the corresponding nodes are still distinct elements of the branch.

Formulae of the form $\Box_R F$ and $\mathsf{A}F$ are called *universal formulae*; nodes whose labels have the form $a\colon G$, where G is a universal formula, are *universal nodes* and the rules $\Box$ and A are *universal rules*. For convenience, also the Trans and Link rules are considered as universal rules. When the A rule is applied producing a node labelled by a formula of the form $b : F$, it is said to *focus* on b (and b is the focused nominal of the inference). The $\Diamond$ and E rules are called *blockable rules*, non relational formulae of the form $a\colon \Diamond_R F$ and $a\colon \mathsf{E}F$ are *blockable formulae* and a node labelled by a blockable formula is a *blockable node*. The premiss n of either the $\Box$ or Trans rule is called the *major premiss* of the rule, while the premiss m of such rules as well as the Link rule is the *minor premiss* of the respective rule.

The first node of a branch $\mathcal{B}$ is called the *top node* and its label the *top formula* of $\mathcal{B}$. Nominals occurring in the top formula are called *top nominals*. The notion of top nominal is relative to a tableau branch (not the whole tableau), because applications of the equality rule may change the top formula, hence the set of top nominals.

A branch is *closed* whenever it contains, for some nominal a, either a pair of nodes $(n)\, a\colon p$, $(m)\, a\colon \neg p$ for some $p \in \mathsf{PROP}$, or a node $(n)\, a\colon \neg a$. As usual, a closed branch is never expanded. A branch which is not closed is *open*. A branch is *complete* when it cannot be further expanded.

4.2 Blocking and Other Restrictions on Branch Expansion

Termination is achieved by means of a form of anywhere blocking with indirect blocking. The definition of direct and indirect blocking is almost standard, but for the fact that direct blocking may be affected by nominal renaming (and is therefore dynamic), and indirect blocking relies on a particular partial order on tableau nodes. Direct blocking must take into account the fact that, due to the presence of the binder, a potentially infinite number of distinct nominals may occur in the bodies of node labels. The $\Diamond$ and E expansion rules, in fact, add fresh nominals to the branch and the expansion of a node $(n)\, a{:}\downarrow x.F$ produces a node containing a in the body of its label. As a consequence, a branch may contain an infinite number of blockable formulae pairwise differing not only for the respective outermost nominals.

Differently from other tableau calculi for hybrid logics, blocking is a relation between nodes, not nominals: mainly, a node n directly blocks a node m in a tableau branch $\mathcal{B}$ whenever $n < m$ in $\mathcal{B}$ and their respective labels (formulae) are equal up to (a proper form of) nominal renaming.

In order to define direct blocking, some preliminary notions must be introduced. Two nominals a and b are said to be *compatible* in a tableau branch $\mathcal{B}$ if they label the same propositions in PROP and the same formulae of the form $\Box_R F$. A substitution $\theta = \{a_1 \mapsto b_1, \ldots, a_n \mapsto b_n\}$, where $a_i, b_i \in \mathsf{NOM}$, is called a *matching* from a formula F to a formula G if θ is injective and $\theta(F) = G$. A matching $\theta = \{a_1 \mapsto b_1, \ldots, a_n \mapsto b_n\}$ is a *mapping* for $\mathcal{B}$ if:

- the nominals $a_1, \ldots, a_n, b_1, \ldots, b_n$ are all non-top nominals in $\mathcal{B}$;
- for all $i = 1, \ldots, n$, a_i and b_i are compatible in $\mathcal{B}$.

A formula F can be *mapped* to a formula G in $\mathcal{B}$ if there exists a mapping θ for $\mathcal{B}$ mapping F to G (i.e. such that $\theta(F) = G$). In other terms, F can be mapped to G when G can be obtained from F by renaming non-top nominals by other compatible ones.

The (direct) blocking restriction forbids the application of a blockable rule to a node n, whenever the label of a node $m < n$ can be mapped to label(n). Since the set of top nominals and nominal compatibilities may change during branch construction, blocks may also change consequently. In other terms, the blocking relation is dynamic, i.e. blockings are not established forever, since they are relative to a tableau branch, and can be undone when expanding the branch. What may happen is that a node may be blocked in a branch $\mathcal{B}$ and then unblocked after expanding $\mathcal{B}$, because the addition of new nodes or changes in node labels may destroy nominal compatibility. Similarly, when the equality rule affects either the label of a blocked node n or that of its blocking node, n may be unblocked. Possibly, a new blocking can be introduced (but compatibilities must be checked again), by means of a different mapping.

When a node is blocked, all its *descendants* w.r.t. a particular relation called the *offspring relation* are indirectly blocked and are called *phantom nodes*. The offspring relation, denoted by $\prec_{\mathcal{B}}$, is a partial order arranging the nodes of a branch into a family of finitary trees, where each node has at most

one parent and only blockable nodes (i.e. nodes expandable by means of either the $\Diamond$ or the E-rule) may have children. Every tree is rooted at a node called a *root node* (a node with no *parents* w.r.t. the offspring relation).

The nodes of the initial tableau are all root nodes. Blockable rules generate *children* (w.r.t. the offspring relation) of the expanded node: if the expansion of a blockable node n generates m_0 (and m_1), then $n \prec_{\mathcal{B}} m_0$ (and $n \prec_{\mathcal{B}} m_1$). All the other rules, with the exception of the A rule, generate *siblings* of one of the premisses of the inference (two nodes are siblings if either they are both root nodes or they have the same parent). In particular, the application of a (non-blockable) single premiss rule to a node n generates a sibling of n, and the application of a universal rule generates a sibling of the minor premiss of the rule (see Table 3, where the trees on the right represent the offspring relation induced by the corresponding rules, when the considered premiss is not a root node). The conclusion of an application of a $\Box$, Trans or Link rule is a sibling of its minor premiss. Without entering into details, the reason for this is that, in order to ensure termination, the trees induced by the offspring relation must have a bounded width. Since universal nodes may be expanded a potentially unbounded number of times, the conclusions of such inferences cannot be siblings of the universal node generating them.

The A rule is more delicate than the other universal rules, since it can also be applied several times to the same node generating a potentially unbounded number of different conclusions, but it lacks a minor premiss to be taken as a sibling of the conclusion. It is only required that the focused nominal b occurs in the branch $\mathcal{B}$ to which the rule is applied. Among the nodes containing b, the first *non-phantom* one in $\mathcal{B}$ is chosen to play the role of "minor premiss" of the application of the A rule, and it is established that the conclusion of an application of the A rule in a branch $\mathcal{B}$ is a sibling of the first non-phantom node in $\mathcal{B}$ where the focused nominal occurs.[2] It is worth pointing out that, though a non-phantom node may become a phantom when the branch is expanded, the offspring relation is static: if $n \prec_{\mathcal{B}} m$ and $\mathcal{B}'$ is obtained as an expansion of $\mathcal{B}$, then also $n \prec_{\mathcal{B}'} m$. Note that in principle, an application of the A rule could have no minor premiss (when the focused nominal only occurs in phantom nodes). This possibility, however, will be ruled out by the restrictions on rule applications that are introduced later on.

It is worth pointing out that the trees induced by the offspring relation do not correspond to the relational structure of the possible model found. In particular, although a set of siblings may in some respect look like the set of formulae labeling a node in description logic tableaux like those defined in [29], they only loosely characterize a state in the possible model found. First of all, the @ rule (that is in fact missing in description logic tableaux) generates a sibling of the premiss "describing" a different state. Secondly, the A rule focused on a nominal b generates a sibling of a node where b occurs but is not necessarily the outermost nominal. Finally, an application of the equality rule

[2] Properly, the offspring relation and blockings are defined by mutual recursion on branch construction. The reader is referred to [13] for the details.

$\dfrac{(n)\,a\colon (F \wedge G)}{\begin{array}{l}(m_0)\,a\colon F\\(m_1)\,a\colon G\end{array}}\ (\wedge)$	parent n m_0 m_1
$\dfrac{(n)\,a\colon (F_0 \vee F_1)}{(m)\,a\colon F_i}\ (\vee)$ $\dfrac{(n)\,a\colon b\colon F}{(m)\,b\colon F}\ (@)$ $\dfrac{(n)\,a\colon {\downarrow}x.F}{(m)\,a\colon F[a/x]}\ (\downarrow)$	parent n m
$\dfrac{(n)\,a\colon \Box_R F \quad (m)\,a \Rightarrow_R b}{(k)\,b\colon F}\ (\Box)$ $\dfrac{(n)\,a\colon \Box_S F \quad (m)\,a \Rightarrow_R b \quad (t)\ldots \quad (i)\ldots}{(k)\,b\colon \Box_R F}\ (\mathsf{Trans})$ $\dfrac{(m)\,a \Rightarrow_R b \quad (i)\,R \sqsubseteq S}{(k)\,a \Rightarrow_S b}\ (\mathsf{Link})$	parent m k
$\dfrac{(n)\,a\colon \Diamond_R F}{\begin{array}{l}(m_0)\,a \Rightarrow_R b\\(m_1)\,b\colon F\end{array}}\ (\Diamond)$	parent | n m_0 m_1
$\dfrac{(n)\,a\colon \mathsf{E}F}{(m)\,b\colon F}\ (\mathsf{E})$	parent | n | m

Table 3 A graphical representation of the offspring relation

does not "merge" the sets of siblings (partially) describing the replacing and replaced nominal.

The restrictions on the applicability of expansion rules can now be defined. While expanding a tableau branch $\mathcal{B}$:

R1. no node labelled by a formula already occurring in $\mathcal{B}$ as the label of a *non-phantom* node is ever added to $\mathcal{B}$.
R2. Blockable nodes can be expanded at most once in a branch.

R3. A phantom node cannot be expanded by means of a single-premiss rule (including the equality rule), nor can it be used as the minor premiss of a universal rule.

R4. A blockable node n cannot be expanded if it is directly blocked in $\mathcal{B}$.

Note that, as a particular case of restriction **R3**, the A rule cannot focus on a nominal which only occurs in phantom nodes in the branch. Consequently, thanks to this restriction, every application of the A rule has a minor premiss.

This section ends up with an example showing the dynamic nature of blockings. Figure 4 represents a complete and open tableau branch $\mathcal{B}$ for the assertion $\mathsf{Trans}(r)$ and the formula $F = \Diamond_r\top \wedge \mathsf{A}\Box_{r^-}p \wedge \Box_r G$, where $G = \downarrow x.\Diamond_r\downarrow y.x\colon\Diamond_r\neg y$. The notations $n \rightsquigarrow^{\mathcal{R}} m$ or $(n_1,\ldots,n_k) \rightsquigarrow^{\mathcal{R}} m$, used in the rightmost column, mean that the addition of node m is due to the application of rule $\mathcal{R}$ to node n (or nodes $n_1,\ldots,n_k$).

0)	$a_0\colon F$	
1)	$\mathsf{Trans}(r)$	
2)	$r \sqsubseteq r$	
3)	$a_0\colon(\Diamond_r\top \wedge \mathsf{A}\Box_{r^-}p)$	$0 \rightsquigarrow^{\wedge} 3$
4)	$a_0\colon\Box_r G$	$0 \rightsquigarrow^{\wedge} 4$
5)	$a_0\colon\Diamond_r\top$	$3 \rightsquigarrow^{\wedge} 5$
6)	$a_0\colon\mathsf{A}\Box_{r^-}p$	$3 \rightsquigarrow^{\wedge} 6$
7)	$a_0\colon\Diamond_r a_1$	$5 \rightsquigarrow^{\Diamond} 7$
8)	$a_1\colon\top$	$5 \rightsquigarrow^{\Diamond} 8$
9)	$a_1\colon\Box_{r^-}p$	$(6,7) \rightsquigarrow^{\mathsf{A}} 9$
10)	$a_0\colon\Box_{r^-}p$	$(6,0) \rightsquigarrow^{\mathsf{A}} 10$
11)	$a_1\colon\Box_r G$	$(4,7,1,2) \rightsquigarrow^{\mathsf{Trans}} 11$
12)	$a_1\colon G$	$(4,7) \rightsquigarrow^{\Box} 12$
13)	$a_0\colon p$	$(9,7) \rightsquigarrow^{\Box} 13$
14)	$a_1\colon\Diamond_r\downarrow y.a_1\colon\Diamond_r\neg y$	$12 \rightsquigarrow^{\downarrow} 14$
15)	$a_1\colon\Diamond_r a_2$	$14 \rightsquigarrow^{\Diamond} 15$
16)	$a_2\colon\downarrow y.a_1\colon\Diamond_r\neg y$	$14 \rightsquigarrow^{\Diamond} 16$
17)	$a_2\colon\Box_r G$	$(11,15,1,2) \rightsquigarrow^{\mathsf{Trans}} 17$
18)	$a_2\colon G$	$(11,15) \rightsquigarrow^{\Box} 18$
19)	$a_2\colon a_1\colon\Diamond_r\neg a_2$	$16 \rightsquigarrow^{\downarrow} 19$
20)	$a_2\colon\Diamond_r\downarrow y.a_2\colon\Diamond_r\neg y$	$18 \rightsquigarrow^{\downarrow} 20$
21)	$a_1\colon\Diamond_r\neg a_2$	$19 \rightsquigarrow^{@} 21$
22)	$a_2\colon\Diamond_r a_3$	$20 \rightsquigarrow^{\Diamond} 22$
23)	$a_3\colon\downarrow y.a_2\colon\Diamond_r\neg y$	$20 \rightsquigarrow^{\Diamond} 23$
24)	$a_1\colon\Diamond_r a_4$	$21 \rightsquigarrow^{\Diamond} 24$
25)	$a_4\colon\neg a_2$	$21 \rightsquigarrow^{\Diamond} 25$
26)	$a_3\colon\Box_r G$	$(17,22,1,2) \rightsquigarrow^{\mathsf{Trans}} 26$
27)	$a_3\colon G$	$(17,22) \rightsquigarrow^{\Box} 27$
28)	$a_3\colon a_2\colon\Diamond_r\neg a_3$	$23 \rightsquigarrow^{\downarrow} 28$
29)	$a_4\colon\Box_r G$	$(11,24,1,2) \rightsquigarrow^{\mathsf{Trans}} 29$
30)	$a_4\colon G$	$(11,24) \rightsquigarrow^{\Box} 30$
31)	$a_2\colon\Diamond_r\neg a_3$	$28 \rightsquigarrow^{@} 31$
32)	$a_4\colon\Diamond_r\downarrow y.a_4\colon\Diamond_r\neg y$	$30 \rightsquigarrow^{\downarrow} 32$
33)	$a_2\colon\Diamond_r a_5$	$31 \rightsquigarrow^{\Diamond} 33$
34)	$a_5\colon\neg a_3$	$31 \rightsquigarrow^{\Diamond} 34$
35)	$a_4\colon\Box_{r^-}p$	$(6,24) \rightsquigarrow^{\mathsf{A}} 35$
36)	$a_3\colon\Box_{r^-}p$	$(6,22) \rightsquigarrow^{\mathsf{A}} 36$
37)	$a_2\colon\Box_{r^-}p$	$(6,15) \rightsquigarrow^{\mathsf{A}} 37$
38)	$a_1\colon p$	$(35,24) \rightsquigarrow^{\Box} 38$
39)	$a_2\colon p$	$(36,22) \rightsquigarrow^{\Box} 39$
40)	$a_4\colon\Diamond_r a_6$	$32 \rightsquigarrow^{\Diamond} 40$
41)	$a_6\colon\downarrow y.a_4\colon\Diamond_r\neg y$	$32 \rightsquigarrow^{\Diamond} 41$
42)	$a_6\colon\Box_{r^-}p$	$(6,40) \rightsquigarrow^{\mathsf{A}} 42$
43)	$a_4\colon p$	$(42,40) \rightsquigarrow^{\Box} 43$

Fig. 4 A complete tableau branch for $\{\Diamond_r\top \wedge \mathsf{A}\Box_{r^-}p \wedge \Box_r G, \mathsf{Trans}(r)\}$, where $G = \downarrow x.\Diamond_r\downarrow y.x\colon\Diamond_r\neg y$.

In the comments below, the notation $\mathcal{B}_n$ is used to denote the branch segment up to node n included. Note that, in this example, the formulae to be taken into account to check compatibilities are p, $\Box_{r^-}p$ and $\Box_r G$.

The root nodes are (beyond nodes labelled by assertions): 0–6 and 10, and the offspring relation is:

$5 \prec_{\mathcal{B}} \{7-9, 11-14\}$ $\quad$ $14 \prec_{\mathcal{B}} \{15-21, 37\}$

$20 \prec_{\mathcal{B}} \{22, 23, 26-28, 31, 36, 39\}$ $\quad$ $21 \prec_{\mathcal{B}} \{24, 25, 29, 30, 32, 35, 38\}$

$31 \prec_{\mathcal{B}} \{33, 34\}$ $\quad$ $32 \prec_{\mathcal{B}} \{40-43\}$

For instance, node 7 is the minor premiss of the application of the Trans rule producing 11, and it is also the minor premiss of the application of the $\Box$ rule

producing 12 and 13; therefore 7, 11, 12 and 13 are siblings (they are all offsprings of node 5). Moreover, 7 is also the first non-phantom node where a_1 occurs when the A rule is applied to produce node 9 focusing on a_1, therefore 7 is the minor premiss of the inference, thus one of the siblings of node 9.

As a further example, though node 22 is a phantom in the final branch, it is not a phantom in $\mathcal{B}_{35}$ (see below). The branch $\mathcal{B}_{35}$ is expanded by an application of the A rule focusing on a_3 and producing node 36. In this branch, 22 is the first non-phantom node where a_3 occurs, so it is the minor premiss of the A inference and 22 and 36 are siblings (in all branch segments from $\mathcal{B}_{36}$ onward).

In the whole branch $\mathcal{B} = \mathcal{B}_{43}$, the nodes 20 and 32 are blocked by 14, because a_1 is compatible with both a_2 and a_4: the relevant formulae such nominals label in the final branch are p, $\Box_{r^-}p$ and $\Box_r G$.

The fact that 20 and 32 are blocked by 14 intuitively means that a_2 and a_4 behave "like" a_1, However, though a_2 and a_4 are compatible, the presence of node 25 does not allow to identify the states they denote in a model of this open branch.

Nodes 20 and 32 being directly blocked in $\mathcal{B}$, all their descendants ($22, 23, 26$–$28, 31, 33, 34, 36, 39$–43) are phantom nodes in $\mathcal{B}$.

However, node 20 is blocked by 14 only in $\mathcal{B}_{37}$ (where a_1 and a_2 label $\Box_{r^-}p$ and $\Box_r G$) and from $\mathcal{B}_{39}$ onward, when both (38) $a_1{:}\,p$ and (39) $a_2{:}\,p$ are added. In particular, 20 is not blocked in $\mathcal{B}_i$ for $i \leq 36$, therefore, it is expanded, and its descendants can also be expanded (or used as minor premisses) till node 39 is added to the branch.

Analogously, 32 is blocked by 14 in $\mathcal{B}_i$ only for $35 \leq i \leq 37$ and $i = 43$. Therefore, for instance, node 40 is not a phantom in $\mathcal{B}_{42}$, so that it can be used as the minor premiss of the application of the $\Box$ rule producing 43. Note also that a_2 and a_4 are compatible in $\mathcal{B}_i$ for $32 \leq i \leq 34$ and $i = 38$ and 20 is not blocked in these branch segments, therefore it blocks 32 (though 20 is not an ancestor of 32 w.r.t. the offspring relation).

In order for node 31 to be blocked by 21, a_1, a_2 and a_3 must be compatible. But when a_1 and a_2 are compatible, node 20 is blocked, and, in such a case, node 31, that is one of 20's children, is a phantom. Therefore 31 is never directly blocked.

The branch is complete: no further expansion are possible without violating the restrictions on blocked nodes. In particular, in the whole branch:

- the A rule cannot focus on a_5, which only occurs in phantom nodes.
- Though nodes 36 and 42, obtained by applications of the A rule, are phantoms, such a rule cannot focus again on a_3 and a_6, which only occur in phantom nodes.
- Though 26 and 27 are phantoms, the Trans and $\Box$ rules cannot use again 22 as a minor premiss, since it is a phantom too.
- Similarly, the other phantom nodes labelled by relational formulae cannot be used as minor premisses. For instance, 40 cannot be used as the minor premiss of an application of the $\Box$ rule, paired with 29.

5 The Prover

The Sibyl prover is written in Objective Caml and its Linux executables are available at `http://cialdea.dia.uniroma3.it/sibyl/`. It takes as input a file containing a set of assertions and a set of formulae, checks them for satisfiability and outputs the result. When called with a specific option, it also generates a LaTeX file with the explored tableau branches. The expansion strategy can be specified in a configuration file, overriding the default rule application priorities.[3]

If some formula is not in $\mathsf{HL} \setminus \Box\downarrow\Box$, then Sibyl warns the user that termination and correctness of the result are not guaranteed. In fact, in such cases, not only may obviously the proof procedure not terminate, but, in principle, it may also happen that the input problem is unsatisfiable, and yet Sibyl does not find a closed tableau for it. In fact, as already said, both the completeness and termination proofs rely on the assumption that the input formulae belong to the decidable fragment. For instance, Sibyl does not terminate on the input problem $\{r \sqsubseteq s,\ \neg p,\ \Diamond_r^- q,\ \Box_s^- \Diamond_r^- q,\ \mathsf{A}{\downarrow}x.\mathsf{E}{\downarrow}y.x{:}\,\Box_r y\}$. As a matter of fact, we are not aware of any example where Sibyl gives an incorrect "satisfiable" answer, but the assumption on which the completeness proof relies does not guarantee that a closed tableau exists for any unsatisfiable formula outside the fragment $\mathsf{HL} \setminus \Box\downarrow\Box$.

Beyond a preliminary formula simplification, Sibyl adopts two optimization techniques, that are both standard in hybrid, modal, description and first-order logic provers: *semantic branching* (see for instance [19,21,23,25]), where disjunctions are expanded by the rule shown in Figure 5, and *backjumping* (see for instance [21,25,30,44]). Both optimizations can be disabled by calling the prover with specific options.

$$\frac{a{:}\,(F \vee G)}{\begin{array}{l|l} a{:}\,F & a{:}\,G \\ & a{:}\,\neg F \end{array}}\ (\vee)$$

Fig. 5 Semantic branching

Figure 6 displays a simplified functional view of the system architecture, showing the decomposition of the system into modules and their mutual dependencies. Modules are grouped by functional affinity and boxes representing groups of modules have a double boundary, even when, for the sake of readability, their content is not shown. Dependencies between modules (or groups of modules) are represented by arrows, with the standard semantics: $\mathsf{A} \rightarrow \mathsf{B}$ means that module A depends on (the interface of) module B. Arrows originating from a group of modules mean that several modules in the corresponding

[3] By default, (1) the equality rule and branch closure checks are applied with the highest priority. Following come (2) the @ rule, (3) the A rule, (4) the $\wedge$ rule, (5) the $\downarrow$ rule, (6) the other universal rules, $\Box$, Trans and Link, (7) the blockable rules $\Diamond$ and E and, finally, (8) the $\vee$ rule.

box depend from the module/group pointed to by the arrow tail. Analogously, arrow heads pointing to a group of modules represent a dependency from more than a module of the pointed box. More details on the functionalities provided by the main modules and the data types used in the implementation can be found in Appendix A.

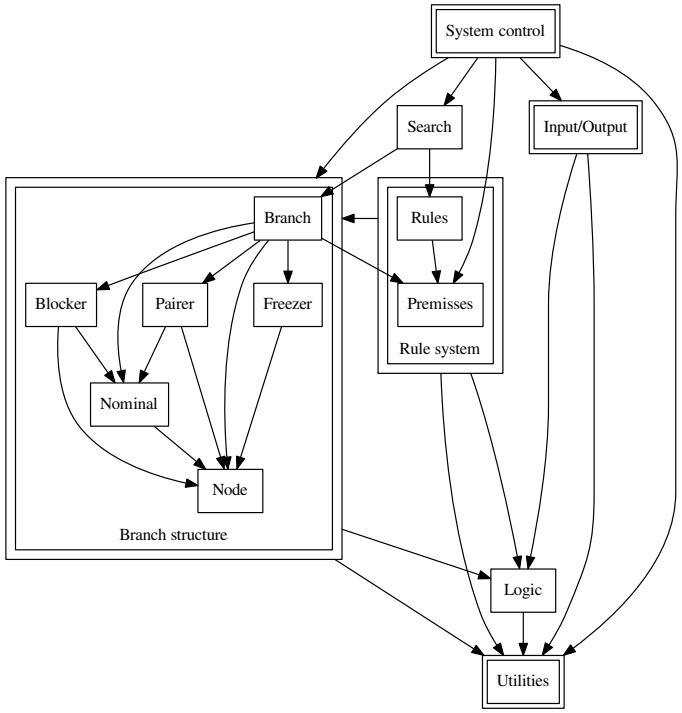


Fig. 6 Simplified architecture of the prover

The main challenges in implementing the calculus presented in Section 4 are due to the fact that many properties and relations holding in a branch are dynamic: the set of top nominals and nominal compatibilities may change during branch construction, hence blocks are dynamic, and so is the set of phantom nodes. The latter fact, in particular, is delicate when managing applications of the A rule. Another important aspect to be addressed is how to give an efficient implementation of the equality rule. The rest of this section describes how these issues are dealt with by the Sibyl prover.

5.1 Nominal Substitution

Since a naive application of nominal substitution when applying the equality rule (i.e. actually replacing a nominal in every formula where it occurs) would be computationally quite expensive, a different mechanism is adopted.

In the representation of formulae, nominals are represented by pointers, so that different occurrences of the same nominal (in the same or different formuale) point to a shared value. The latter is a structure that in turn points to the possibly replacing nominal (a structure of the same type). Substitution only affects such a shared structure and formulae are never directly modified. In other terms, nominals are arranged in a forest of trees, implemented by a *parent pointer representation*. Each tree represents an equivalence class of nominals: every node (nominal) in such a tree points to the nominal replacing it (its parent in the tree of nominals), and the root is the nominal replacing all the other nominals in its tree. Applications of the equality rule just add pointers to the structures representing nominals.

5.2 Branches and Dynamic Blocking

A branch is a structure made up essentially of:

- a list of its nodes, each of them storing information on its label, its status w.r.t. blocking, its relatives according to the offspring relation and other data useful to enforce the restrictions on the expansion rules;
- the set of nominals occurring in the branch, each of them being associated to the formulae it labels in the branch, and bearing information on its top/non-top status;
- a priority queue containing the applicable rules, i.e. suitable, yet-to-be-processed rule premisses (the *expansion queue*);
- structures helping the branch to deal with direct and indirect blocks and applications of the A rule.

A Branch is responsible for enforcing restrictions **R1**-**R4** on rule application. Restriction **R2** (blockable nodes can be expanded at most once in a branch) is simply achieved by the fact that, once a rule application has been carried out, it never re-enters the queue.

Restriction **R1** requires that no node labelled by a formula already occurring in $\mathcal{B}$ as the label of a non-phantom node is ever added to $\mathcal{B}$. Since the property of being a phantom may dynamically change during branch construction, it may be the case that a node n cannot be added to the branch because another non-phantom node m with the same label already exists. In such a case, we say that n is *frozen* and m is the *freezing* node. However, m may afterwards become a phantom, so that n must be added to the branch. Frozen nodes (together with the respective freezing nodes) are stored in a structure, called Freezer, until they are allowed to be added to the branch.

Restrictions **R3** (a phantom node cannot be expanded by means of a single-premiss rule, nor can it be used as the minor premiss of a universal rule)

and **R4** (a blockable node cannot be expanded if it is directly blocked in $\mathcal{B}$) are enforced with the help of two structures: the first, called Blocker, stores potential blocking relations; the other, called Pairer and described in the next subsection, is responsible for choosing focused nominals and minor premisses of the A rule.

A Blocker is responsible for handling most of the aspects related to the blocking relation. This structure groups together all nodes that might pairwise be affected by a blocking, because their labels have the same structure, i.e. they are equal modulo nominal renaming. Hence if $n < m$ are in the same group, n may block m once nominal top/non-top status and compatibilities are checked. This allows a Blocker to quickly determine which nodes are blocked and which are not at the request of its owner Branch.

The main function implemented in a Blocker is the update function: whenever nominal compatibilities or a top/non-top nominal status may have changed, the existence of a mapping between formulae in the same list of the Blocker is checked, and the fields recording the blocking status of the involved nodes are updated accordingly. When a node is blocked, all its descendants w.r.t. the offspring relation are marked as phantoms, and when it results not to be blocked any longer, its descendants cease to be phantoms. When a node is unblocked, it re-enters the expansion queue of the Branch to which the Blocker belongs. Moreover, the update operation may affect the Freezer: if some freezing node n becomes a phantom, the nodes frozen by n exit the Freezer and are added to the branch.

5.3 The Universal Global Rule

The treatment of the A rule is delicate. Not only can in fact the A rule be fired several times on the same node labelled by a formula of the form $a{:}\,\mathsf{A}F$ (briefly called an A-node in the sequel), but the same A-node can be expanded multiple times on the same focused nominal (with a different minor premiss, when the results of previous expansions become phantoms), or with the same minor premiss, on different focused nominals. Moreover, A-nodes must be expanded as long as a suitable, previously unused pair $\langle f, m\rangle$ (where f is the focused nominal and m its companion minor premiss) can be found in the branch. The matches between an A-node n and $\langle f, m\rangle$ will be called *pairings*, and denoted by $\langle n, \langle f, m\rangle\rangle$.

The treatment of the A rule must ensure that:

- for each A-node n, all suitable pairings $\langle n, \langle f, m\rangle\rangle$ are eventually consumed;
- the A rule is never applied twice to the same $\langle n, \langle f, m\rangle\rangle$;
- pairings $\langle n, \langle f, m\rangle\rangle$ that cannot be used because of restriction **R3** are not completely discarded, because the minor premiss may cease to be a phantom at some later point.

Moreover, each A-node to be expanded should be paired with a suitable $\langle f, m\rangle$ as efficiently as possible. It is worth reminding that

- the A rule can focus on an arbitrary nominal, provided that it occurs in at least one non-phantom node;
- the first (i.e. earliest occurring in the branch) non-phantom node where the focused nominal occurs, in the branch where the rule is applied (i.e. just before the addition of the new node) plays the role of minor premiss of the A rule.

These concerns are addressed with the help of a Pairer structure, that keeps track of the possible applications of the A rule. Its responsibilities are to pair A-nodes to focused nominals and minor premisses, to return a suitable pairing whenever one is called for, and to inform the caller when it is not possible to compute a pairing. A Pairer maps each A-node in the branch to the list of the pairs $\langle f, m\rangle$ already used to fire the A-rule with the considered node, a list of the available nominals that have never been focused on with the considered A-node and those that have been considered but could not be used because occurring only in phantom nodes. The Branch to which the Pairer belongs is responsible of informing it when new nominals are added to the branch and when some disappear because of applications of the equality rule.

The basic functionality of a Pairer is to compute a suitable minor premiss and focused nominal for a given A-node, whenever it is extracted from the priority queue of the owner Branch. This is done by scanning the list of available nominals until one is found for which a non-phantom minor premiss can be found. The branch is then expanded accordingly, and the A-node is enqueued again. The next time it is dequeued, the Pairer will compute a different pair $\langle f, m\rangle$. In case no suitable pair $\langle f, m\rangle$ can be found, the calling Branch inserts the A-node in a list of *deferred nodes*, that will be considered again when the expansion queue becomes empty. Then, for each deferred A-node, the Pairer reconsiders all the previously discarded nominals.

6 Experiments

6.1 The Test Sets

In order to analyze Sibyl's behaviour, it has been run on a set of randomly generated tests and on a set of hand-tailored ones. The test sets can be downloaded from Sibyl web page. Sibyl and the provers used for comparison (SPASS and HTab) were run on these tests on an Intel Core i3 3.3 GHz with 12GB RAM, running under Linux (Debian distribution, release 9.1)

6.1.1 The Randomly Generated Problems

Each of the randomly generated tests is based on a file generated by hGen [5], modified so as to obtain formulae in the decidable fragment of hybrid logic treated by Sibyl and with the addition of a random set of transitivity and

inclusion assertions.[4] A first group of 1620 tests (called the *basic set* in the sequel) has been generated with 30% probability for a relation to be transitive and 30% probability for any pair of relations to be related by a subrelation assertion. The tests are grouped according to their modal degree (varying from 2 to 10), each group containing 180 tests with 10 to 50 formulae (step 5, i.e. 20 tests for each number of formulae). In order to evaluate the impact of the presence of assertions on Sibyl's behaviour, other four groups of tests have been obtained from the basic set, reducing the number of assertions in each file, respectively to 75%, 50%, 25% and no assertions at all (therefore obtaining a total of 8100 tests).[5] On these tests, the provers were given one minute. Globally, 49.14% of the tests were recognized by some prover as satisfiable, 49.44% as unsatisfiable, and the remaining 1.42% problems were solved by no prover in the allowed time.[6] The distribution of satisfiable and unsatisfiable tests in dependence of the percentage of assertions is shown in the diagram of Figure 7.

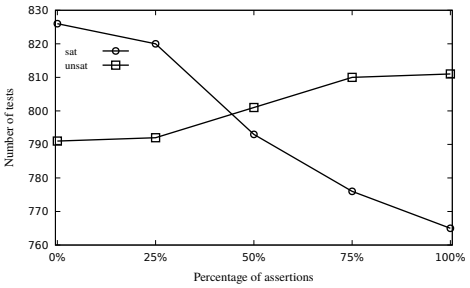


Fig. 7 Number of satisfiable and unsatisfiable problems for the basic set (100%) and those in the 75%, 50%, 25% and 0% reductions of assertions

Within each main group of tests, modal depth has been taken as the parameter for the horizontal axes and provers' performances have been evaluated in terms of percentage of timeouts (i.e. problems that could not be solved in the allowed time limit). In fact, running times are mostly very low: almost 95% of the 8100 tests were solved by Sibyl in less than 10 seconds, and the prover ran out of time in little less than 4% of the tests. Therefore, a very small percentage of solved problems required more than 10 seconds to be decided. Consequently, also the median times are always very low. Unfortunately, we

[4] Since hGen generated only formulae with forward relations, they were randomly replaced by the corresponding converse ones.

[5] The first experiments with Sibyl, briefly reported in [14], used the same random test set described here. Unfortunately, in fact, hGen is no longer maintained, and could not be used to generate new problems.

[6] The percentage of unsolved tests grows almost linearly from 0.2% in the test set with no assertions to 2.72% in the basic set.

have not been able to adjust hGen parameters so that tests of a more gradually increasing hardness were generated.

6.1.2 The Hand-Tailored Formulae

Since the random generation did not produce a fully satisfactory test set, a set of simple hand-written problems of regularly increasing difficulty has been defined, involving the binder, the global modalities, as well as transitivity and relation hierarchies. The problem of size 1 consists of the unsatisfiable set of formulae and assertions $S_1 = \{\mathsf{Trans}(r),\ r \sqsubseteq s,\ \mathsf{A}\Diamond_r p,\ \Box_s a_1,\ a_1{:}\,\Box_s \neg a_1\}$ (every state is r-related to some state where p holds, the current state is s-related at most to a_1, that is not s-related to itself). The problem of size n consists of set of formulae and assertions listed in Figure 8.

$$
\begin{array}{rl}
(1) & \mathsf{A}\Diamond_r^{n-1} p \\
(2) & \Box_s(a_1 \vee \cdots \vee a_n) \\
(3) & a_1 : \Box_s(\neg a_1 \wedge \cdots \wedge \neg a_n) \\
 & \cdots \\
(n+2) & a_n : \Box_s(\neg a_1 \wedge \cdots \wedge \neg a_n) \\
(\text{assertions}) & \mathsf{Trans}(r),\ r \sqsubseteq s
\end{array}
$$

Fig. 8 The unsatisfiable problem of size n

The first formula implies that every state is r-related to at least n different states. In the translations into other formats, the graded modality is treated like an abbreviation, as defined in Section 3, so it actually contains the binder when $n > 1$. The second formula states that the current world is s-related at most to $a_1, \ldots, a_n$, and the third one that a_1 is not s-related to any of $a_1, \ldots, a_n$. This is enough for the set S_n to be unsatisfiable.[7] The other formulae are added because they actually ease the provers' task.

A corresponding set of satisfiable problems has also been considered, each consisting of the first two formulae only and the assertions of Figure 8.

When run on these problems, the provers were given 5 minutes time.

6.2 Evaluation of Sibyl's Behaviour

This section is devoted to analyse the impact of the optimization techniques, the percentage of assertions, and the relation of running time with other interesting parameters. The evaluation considers the randomly generated tests.

As far as the two optimization techniques implemented in Sibyl are concerned, semantic branching does not affect significantly the performances of

[7] Consider, for instance S_2: the current state a is r-related, hence s-related, to exactly the two states a_1 and a_2 (from the first and second formulae); a_1 is s-related (hence r-related) to neither a_1 nor a_2 (Formula 3), hence it must be r-related to some state b different from both a_1 and a_2. By transitivity, a is also r-related, hence s-related, to b, contradicting 2.

the prover. On the whole random test set, in fact, the prover either with the default settings or by disabling semantic branching could not solve about 4% of the problems in the allowed one minute (semantic branching only allows Sibyl to solve 5 tests more). This is not surprising: the evaluation of other provers has already pointed out that semantic branching is not necessarily beneficial (see, for instance, [18,27]).

On the contrary, like it has been observed in other cases (like [27,30]), backjumping has a greater impact on Sibyl performances: disabling backjumping doubles the number of timeouts. The diagram in Figure 9 plots the percentage of timeouts against the problem modal depth, showing that semantic branching has almost no impact on the prover performances, that instead improve significantly with backjumping.

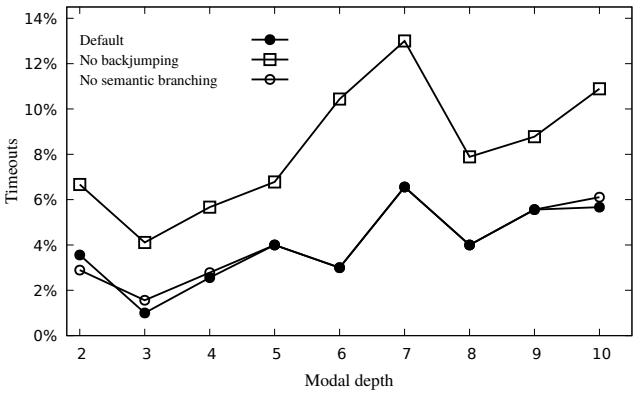


Fig. 9 Impact of backjumping and semantic branching on Sibyl behaviour

The influence of the presence of assertions on Sibyl behaviour (called with the default options) can be evaluated by looking at the diagram in Figure 10, that plots the percentage of timeouts on the test sets against the modal depth. It clearly shows that the number of assertions has no significant impact on the behaviour of the prover. This is confirmed by the fact that the correlation between running times (where timeouts are considered as 60 seconds runs) and the number of assertions is very weak (correlation coefficient 0.07, that is about the same between running times and transitivity/inclusion axioms taken separately).

The problems in the randomly generated sets differ according to different coordinates: number of assertions, modal depth and number of formulae. A different interesting parameter that can be used to classify the problems in the test set is the number of top and non-top nominals they contain. While top nominals (which are strongly related to the problem size) have scarce influence on running time, the correlation between the latter and the number of top and non-top nominals is stronger (correlation coefficient 0.42), just a

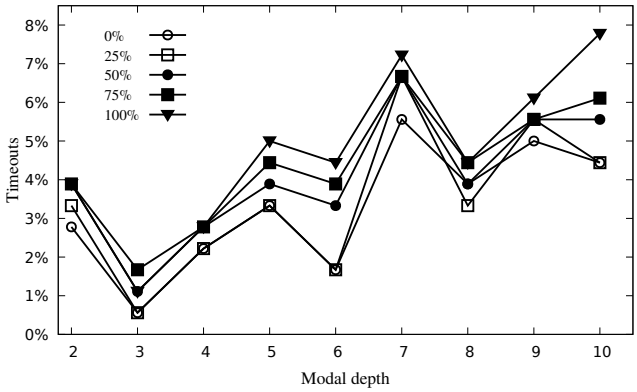


Fig. 10 Impact of the number of assertions on Sibyl behaviour

little weaker than the correlation with the number of application of nominal generating rules.

Other parameters have also been considered. Those that are most strongly related to running time are the number of applications of the $\vee$ rule, the number of expanded branches in the tableau and the number of applications of the equality rule (with correlation coefficients 0.72, 0.66 and 0.63, respectively). These are in turn pairwise very strongly related, so that these results do not necessarily give a significant measure of how efficiently Sibyl handles equality reasoning.

6.3 Comparison with Other Provers

In order to compare Sibyl with other provers, the expressive power of the logics they treat ought to be considered. To the author's knowledge, the only modal prover where the hybrid binder and relation hierarchies coexist is HTab [24,25].[8] With respect to the language accepted by Sibyl, HTab allows for declaring a relation to be reflexive (such a property can however be expressed in $\mathsf{HL} \setminus \Box\downarrow\Box$, as shown by formula 5 of Figure 1), as well as equal to, or a subset of the union of a set of relations. On the other hand, HTab admits neither the converse modalities nor the graded ones. It is worth pointing out that the calculus underlying HTab [9] does not guarantee termination in the presence of the binder, even when it does not occur in the pattern $\Box\downarrow\Box$. Furthermore, even in the absence of the binder, termination has not been proved for the underlying calculus when relation hierarchies are part of the problems.

The provers HyLoRes [4] and HyLoTab [46], already introduced in Section 1, were shown to be less performant than an old version of HTab [24]. On the other hand, Spartacus [21] has not been considered in the evaluation because,

[8] HTab sources can be found at `https://hackage.haskell.org/package/HTab`.

though able to deal with transitive relations and the global modalities, it admits neither relation hierarchies, nor the converse modalities nor, above all, the binder. Needless to say, the binder is absent also in description logic provers.

Beyond HTab, Sibyl has also been compared with the SPASS prover [47].[9] Considering in fact that at present there are no specialized provers for the full hybrid language admitted by Sibyl, a comparison of Sibyl performances w.r.t classical logic provers is of interest.

To this aim, the tool HLtranslate has been developed.[10] The graded modalities possibly present in Sibyl files are eliminated as shown in Section 3. The resulting formulae are applied (a simplification of) the standard translation of hybrid logic into first order logic, according to the following definition: a formula F is translated into $ST_a(F)$, where a is a fresh constant; for every term t, ST_t distributes over boolean operators and, for the others, behaves like shown in Figure 11. This definition differs from the standard translation

$$
\begin{aligned}
ST_t(p) &= p(t)\\
ST_t(a) &= a = t\\
ST_t(t' \colon F) &= ST_{t'}(F)\\
ST_t(\downarrow z.F) &= ST_t(F)[t/z]\\
ST_t(\Box_r F) &= \forall x(r(t,x) \rightarrow ST_x(F))\\
ST_t(\Diamond_r F) &= \exists x(r(t,x) \wedge ST_x(F))\\
ST_t(\Box_{r^-} F) &= \forall x(r(x,t) \rightarrow ST_x(F))\\
ST_t(\Diamond_{r^-} F) &= \exists x(r(x,t) \wedge ST_x(F))\\
ST_t(\mathsf{A}F) &= \forall x ST_x(F)\\
ST_t(\mathsf{E}F) &= \exists x ST_x(F)
\end{aligned}
$$

where the variable x on the right hand sides is a fresh one.

Fig. 11 Translation of hybrid into first-order formulae

given in [43], that uses only two new variables and is defined by mutual recursion between two functions, ST_x and ST_y.[11] When fresh variables are used in the translation of formulae of the form $\Box_R F$, $\Diamond_R F$, $\mathsf{A}F$ of $\mathsf{E}F$, the cases of the binder and the satisfaction operator can be simplified as shown in Figure 11. When fed with this translation, SPASS behaves slightly better than with the two-variable one.

Obviously, the behaviour of a first-order prover fed with the translation of modal formulae depends on the translation itself. Optimized translations for hybrid logics exist, such as the optimized functional translation [38, 3] and

[9] SPASS had already been used to test Sibyl for correctness, find out and correct bugs [14]. At that time, the language accepted by HTab was more restricted, since it did not include transitivity and relation inclusion assertions, and consequently HTab and Sibyl were not compared.

[10] When called with a specific option, HLtranslate converts files from Sibyl input syntax to HTab one. The tool can be downloaded from Sibyl web page.

[11] Moreover, the simplified translation of the global modalities given in Figure 11 omits the trivial guards of the form $x = x$ or $y = y$.

the tree layered one [2]. However, such optimization techniques do not extend to the converse modalities nor to the global ones. The axiomatic translation method [41] can be used for expressive modal/description logics, including the universal and converse modalities, transitivity and role inclusions. However, the expansion rules for the satisfaction operator and the binder, involving substitution, seem to be critical. The axiomatic translation method may probably be extended to these cases, but this is still an open issue.

Consequently, the experiments fed SPASS (v. 3.9) with the standard translation presented above. SPASS was run in default mode, since, from some preliminary tests, other flag settings appeared either to degrade its performance or have no significant effect.

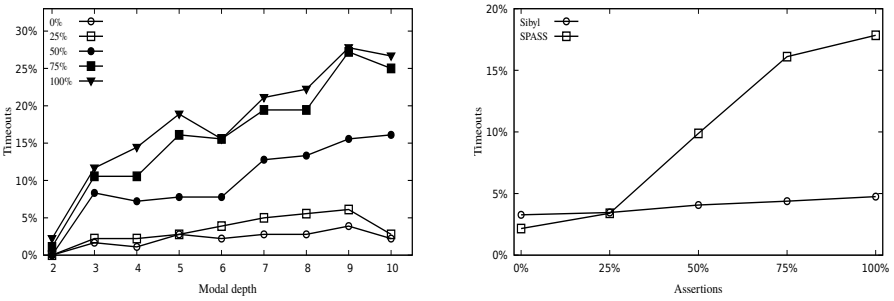
6.3.1 Behaviour of the Provers on the Randomly Generated Tests

Since HTab does not support the converse modalities, only Sibyl and SPASS could be compared on test set described in Section 6.1.1. A first interesting observation can be drawn by comparing the diagram of Figure 10 with diagram (a) of Figure 12: SPASS appears to be much more sensible than Sibyl to the presence of transitivity assertions and relation hierarchies. While in the tests with no assertions SPASS performs better than Sibyl, the performances of the former considerably degrade when the number of assertions increases, like shown by diagram (b) of Figure 10. As a matter of fact, SPASS reaches almost 28% timeouts (for the formulae of modal depth 9 of the basic set), while Sibyl maximum (in all depths) is below 8%.

A clearer view of the relative behaviour of the provers is obtained by analyzing the relation between the percentage of failures and the problem modal degree separately in the test sets with different numbers of assertions. The diagrams of Figure 13 consider the sets with no assertions (a), 50% of the assertions w.r.t. the basic set (b), and the basic set itself (c). Each of them plots the percentage of timeouts of the two provers against the problem modal depth in each group of problems. They clearly show that, with the exception of the tests with no assertions, Sibyl turned out to outperform SPASS. On the whole test set, as already said, Sibyl could not solve nearly 4% of the problems in the allowed time, while SPASS reached almost 10% of failures.

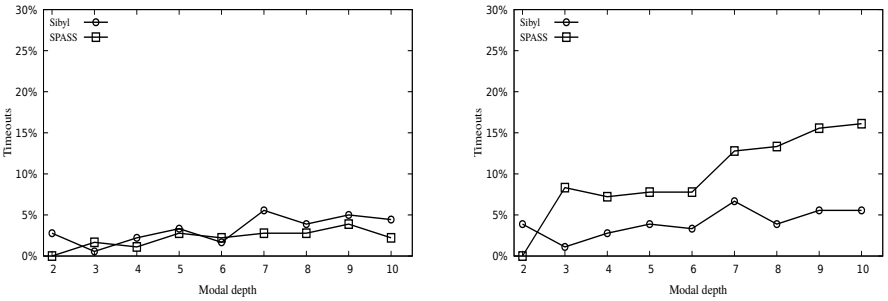
It may be interesting to notice that out of the 800 tests that SPASS could not solve in one minute, only one is an unsatisfiable problem, while the others are either satisfiable or unsolved by Sibyl either (thus unknown). Satisfiable problems are harder for Sibyl too, however, out of its 323 cases of failure, 42 were on unsatisfiable problems and 166 on satisfiable ones (the others being unknown).

In order to extend the comparison to HTab, the test files described in Section 6.1.1 have been processed and every backward relation has been replaced by the corresponding forward one. The experiments have been limited to the basic test set, the problems with 50% assertions w.r.t. the basic set and those with no assertions at all.

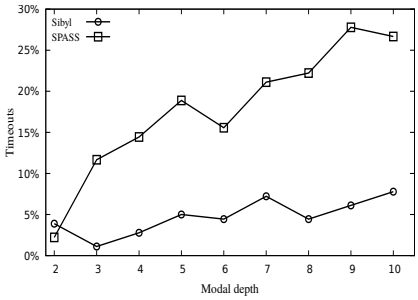


(a) SPASS (b) SPASS and Sibyl

Fig. 12 Impact of the number of assertions on SPASS and Sibyl performances



(a) No assertions at all (b) 50% assertions



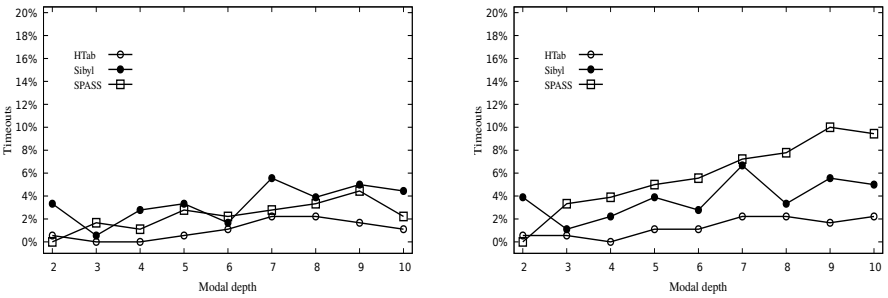
(c) The basic set (100% assertions)

Fig. 13 Sibyl and SPASS performances on the problems with converse modalities

On these tests, HTab (v. 1.6.3) performs much better than Sibyl. In fact, HTab's percentage of timeouts is 1.42% while Sibyl's one is 3.72% (and SPASS failed to solve 6.52% of the 4860 problems). This is not surprising, since, beyond backjumping and semantic branching, HTab implements other optimization techniques, such as unit propagation and lazy branching [21].

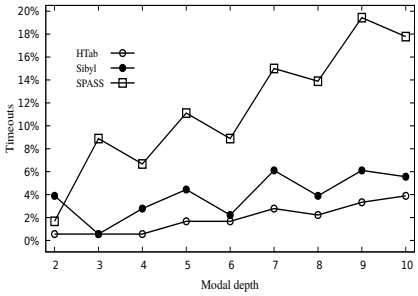
The diagrams in Figure 14 show the behaviour of the three considered provers on the tests without converse modalities. They separately analyse the test sets with different numbers of assertions, plotting, for each of them, the

percentage of timeouts of the three provers against the formulae modal depth, for each of the considered groups of problems: the sets with no assertions (a), 50% of the assertions w.r.t. the basic set (b), and the basic set itself (c). These data clearly confirm that HTab behaves much better than both Sibyl and SPASS in all cases.[12]



(a) No assertions at all

(b) 50% assertions



(c) The basic set (100% assertions)

Fig. 14 HTab, Sibyl and SPASS performances on the problems without converse modalities

The meaning of these experimental results should obviously be tempered by considering that, being a prover for full first-order logic, SPASS does not take advantage of the decidability of the considered fragment of hybrid logic, hence its failures might be actual non-termination cases. A similar remark can be applied to HTab: being based on a different tableau calculus, it does not guarantee termination for formulae in the fragment $\mathsf{HL} \setminus \Box\downarrow\Box$.

6.3.2 Behaviour of the Provers on the Hand-Tailored Formulae

The experiments run on the hand-written formulae detected an anomalous behaviour of the HTab prover. In fact, it gives a "satisfiable" outcome for the simple unsatisfiable test of size 1, while it cannot solve any of the others in

[12] It is fair to remark that for one of the 4860 considered problems HTab gives a "satisfiable" answer, while both Sibyl and SPASS declare it to be unsatisfiable. See also Section 6.3.2.

the allowed 5 minutes. Regarding the satisfiable hand-tailored problems, HTab could only solve the test of size 1, and ran short of time in all the others.

The incorrect answer is due to some problem in dealing with relation hierarchies, since HTab correctly recognizes the unsatisfiability of the problem of size 1, when substituting r for s in the formulae and deleting the inclusion axiom. However, strangely enough, with this modification, HTab still exceeded the allowed time when fed with the problem of size 2.[13] These results lead to believe that HTab's treatment of the binder, at least when coexisting with the global modalities and/or relation hierarchies, is at present not completely satisfactory, beyond being sometimes not reliable. Presumably, the problems that the prover was unable to solve are actually non termination cases, and this fact supports the importance of proof procedures guaranteeing termination for decidable fragments of HL.

The comparison on the hand-tailored formulae can consequently be made only between Sibyl and SPASS. The two provers have quite similar behaviours on the unsatisfiable problems: both could solve the problems up to those of size 22 and ran short of time (5 minutes) for the greater ones. The diagram of Figure 15 plots the running times of the two provers against the problem size, showing that in fact Sibyl and SPASS performances are comparable.

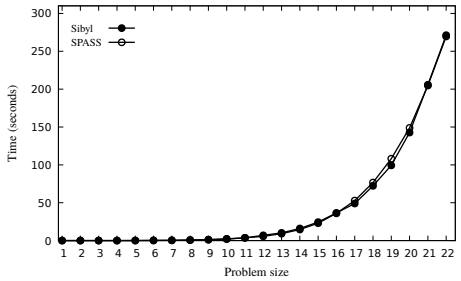


Fig. 15 SPASS and Sibyl on the hand-tailored formulae

On the contrary, on the set of satisfiable problems (that are harder for both provers), SPASS suffers more than Sibyl: the former could solve them only up to the problem of size 4 in the allowed time, while Sibyl solved the problem of size 13 in 77 seconds (and ran short of time for the greater ones).

The above reported experiments are of a very preliminary nature: it would be important to devise sets of problems of increasing difficulty for the logic considered in this work, analogously to what has been done for classical and modal logic [7,37,39,42].

[13] After preprocessing, the problem of size 2 without the relation inclusion assertions becomes $S_2 = \{\mathsf{Trans}(r),\ \mathsf{A}{\downarrow}x.\Diamond_r(p \wedge {\downarrow}y_0.x{:}\,\Diamond_r(p \wedge \neg y_0)),\ \Box_r(a_1 \vee a_2),\ a_1{:}\,(\neg a_1 \wedge \neg a_2),\ a_2{:}\,(\neg a_1 \wedge \neg a_2)\}$.

7 Concluding remarks

This work presents an implementation of a tableau calculus for an expressive hybrid logic, that is provably terminating when a critical pattern involving the binder and universal modalities does not occur in the input formulae. To the author's knowledge, the Sibyl prover is the only one ensuring termination for such a fragment of hybrid logic with the binder. Comparison with two other provers shows that Sibyl has reasonable performance: not as fast as the hybrid logic prover HTab, but generally better than SPASS on first-order translations.

Directions for future work include the implementation of other tableau optimization techniques and extensions of the prover. Among the latter ones, an interesting possibility derives from the consideration that HL subsumes the description logic $\mathcal{SHOIQ}$ [29], and restricted uses of the binder are of interest also in the context of description logics [26,34]. The Sibyl prover can therefore be turned into a prover for a highly expressive description logic with the addition of the binder. Once both the optimization and the extension aims are attained, the prover can be compared with description logic provers dealing with logics that, except for the binder, are at least as expressive as HL. In this regard, it would be important to devise significant description logic domains and examples exploiting the hybrid binder, in order to check whether the operator can be useful notwithstanding the decidability preserving restrictions on its occurrences.

The experimental results described in Section 6 show that, notwithstanding the involved aspects of the implemented proof method, the prover behaves reasonably well. However, the experiments that have been carried out are quite restricted. First of all, as already remarked above, Sibyl should be compared with description logic provers, that are at present the specialized provers dealing with logics closest to HL. Secondly, other first-order provers should be considered in the comparison, beyond SPASS. A further interesting issue to face is whether the axiomatic translation method [41] can be extended to deal with the proof system implemented by Sibyl.

Another important gap to be bridged concerns the test sets used for the experiments. The set of randomly generated problems was in fact devised originally only with the aim of assessing Sibyl's correctness, and is clearly not fully adequate, since the average and median running times of the provers run on it are always very low and the test set, in its whole, does not reflect any phase transition or hard-easy-hard pattern [20]. On the other hand, different and more significant sets of hand-written benchmarks formulae (in HL or its sublanguages) should be conceived, like has been done for classical and modal logics [7,37,39,42], and used for the experiments.

Acknowledgments. The present version of the Sibyl prover extends and improves the work done by some students in their bachelor or master projects. Beyond them, the author wishes to thank Renate Schmidt for her ready answers to my questions on the SPASS prover and translation issues.

References

1. C. Areces, P. Blackburn, and M. Marx. A road-map on complexity for hybrid logics. In *Computer Science Logic*, pages 307–321. Springer, 1999.
2. C. Areces, R. Gennari, J. Heguiabere, , and M. de Rijke. Tree-based heuristics in modal theorem proving. In *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI 2000)*, pages 199–203, 2000.
3. C. Areces and D. Gorín. Unsorted functional translations. *Electronic Notes in Theoretical Computer Science*, 278:3–16, 2011.
4. C. Areces and J. Heguiabehere. Hylores 1.0: Direct resolution for hybrid logics. In *Proceedings of the 18th International Conference on Automated Deduction (CADE-18)*, pages 156–160, 2002. Previously presented in the Proceedings of Methods for Modalities 2, 2001.
5. C. Areces and J. Heguiabehere. hGen: A random CNF formula generator for hybrid languages. In *Proceedings of the 3rd Workshop on Methods for Modalities (M4M-3)*, Nancy, France, 2003.
6. C. Areces and B. ten Cate. Hybrid logics. In *Handbook of Modal Logics*, pages 821–868. Elsevier, 2007.
7. P. Balsiger, A. Heuerding, and S. Schwendimann. A benchmark method for the propositional modal logics K, KT, S4. *Journal of Automated Reasoning*, 24(3):297–317, 2000.
8. P. Blackburn and J. Seligman. Hybrid languages. *Journal of Logic, Language and Information*, 4:251–272, 1995.
9. T. Bolander and P. Blackburn. Termination for hybrid tableaus. *Journal of Logic and Computation*, 17(3):517–554, 2007.
10. S. Cerrito and M. Cialdea Mayer. An efficient approach to nominal equalities in hybrid logic tableaux. *Journal of Applied Non-classical Logics*, 1-2(20):39–61, 2010.
11. S. Cerrito and M. Cialdea Mayer. Nominal substitution at work with the global and converse modalities. In *Advances in Modal Logic*, volume 8, pages 57–74. College Publications, 2010.
12. S. Cerrito and M. Cialdea Mayer. A tableaux based decision procedure for a broad class of hybrid formulae with binders. In *Automated Resoning with Analytic Tableaux and Related Methods (TABLEAUX 2011)*, volume 6793 of *LNAI*, pages 104–118. Springer, 2011.
13. S. Cerrito and M. Cialdea Mayer. A tableau based decision procedure for a fragment of hybrid logic with binders, converse and global modalities. *Journal of Automated Reasoning*, 51(2):197–239, 2013.
14. M. Cialdea Mayer. A proof procedure for hybrid logic with binders, transitivity and relation hierarchies. In *Proceedings of the 24th Conference on Automated Deduction (CADE-24)*, number 7898 in LNCS, pages 76–90. Springer Verlag, 2013.
15. M. Cialdea Mayer. A proof procedure for hybrid logic with binders, transitivity and relation hierarchies (extended version). Technical report, arXiv:1312.2894, 2013.
16. M. Cialdea Mayer. Extended decision procedure for a fragment of HL with binders. *Journal of Automated Reasoning*, 53(3):305–315, 2014.
17. M. Cialdea Mayer and S. Cerrito. Herod and Pilate: two tableau provers for basic hybrid logic. In *Proceedings of the 5th International Joint Conference on Automated Reasoning (IJCAR 2010)*, pages 255–262. Springer, 2010.
18. F. M. Donini and F. Massacci. EXPTIME tableaux for $\mathcal{ALC}$. *Artificial Intelligence*, 124(1):87–138, 2000.
19. J. W. Freeman. Hard random 3-SAT problems and the Davis-Putnam procedure. *Artificial Intelligence*, 81(1):183–198, 1996.
20. I. P. Gent and T. Walsh. The SAT phase transition. In *Proceedings of the Eleventh European Conference on Artificial Intelligence (ECAI'94)*, pages 105–109, 1994.
21. D. Götzmann, M. Kaminski, and G. Smolka. Spartacus: A tableau prover for hybrid logic. *Electronic Notes in Theoretical Computer Science*, 262:127–139, 2010. Proceedings of the 6th Workshop on Methods for Modalities (M4M-6 2009).
22. E. Grädel. On the restraining power of guards. *Journal of Symbolic Logic*, 64:1719–1742, 1998.

23. J. Hladik. Implementation and evaluation of a tableau algorithm for the guarded fragment. In *Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2002)*, pages 145–159. Springer, 2002.
24. G. Hoffmann. *Tâches de raisonnement en logiques hybrides*. PhD thesis, Université Henri Poincaré – Nancy I, 2010.
25. G. Hoffmann and C. Areces. HTab: A terminating tableaux system for hybrid logic. *Electronic Notes in Theoretical Computer Science*, 231:3–19, 2007. Proceedings of the 5th Workshop on Methods for Modalities (M4M-5).
26. I. Horrocks, B. Glimm, and U. Sattler. Hybrid logics and ontology languages. *Electronic Notes in Theoretical Computer Science*, 174:3–14, 2007.
27. I. Horrocks and P. F. Patel-Schneider. Optimizing description logic subsumption. *Journal of Logic and Computation*, 9:267–293, 1999.
28. I. Horrocks and U. Sattler. A description logic with transitive and inverse roles and role hierarchies. *Journal of Logic and Computation*, 9(3):385–410, 1999.
29. I. Horrocks and U. Sattler. A tableau decision procedure for $\mathcal{SHOIQ}$. *Journal of Automated Reasoning*, 39(3):249–276, 2007.
30. U. Hustadt and R. A. Schmidt. Simplification and backjumping in modal tableau. In *Automated Reasoning with Analytic Tableaux and Related Methods. TABLEAUX 1998.*, pages 187–201. Springer Berlin Heidelberg, 1998.
31. M. Kaminski, S. Schneider, and G. Smolka. Terminating tableaux for graded hybrid logic with global modalities and role hierarchies. *Logical Methods in Computer Science*, 7(1), 2011.
32. M. Kaminski and G. Smolka. Hybrid tableaux for the difference modality. *Electronic Notes in Theoretical Computer Science*, 231:241–257, 2007. Proceedings of the 5th Workshop on Methods for Modalities (M4M-5).
33. M. Kaminski and G. Smolka. Terminating tableau systems for hybrid logic with difference and converse. *Journal of Logic, Language and Information*, 18(4):437–464, 2009.
34. M. Marx. Narcissists, stepmothers and spies. In *Proceedings of the 2002 International Workshop on Description Logics (DL 2002)*, 2002. CEUR Workshop Proceedings Volume 53.
35. M. Mundhenk and T. Schneider. Undecidability of multi-modal hybrid logics. *Electronic Notes in Theoretical Computer Science*, 174(6):29–43, 2007.
36. M. Mundhenk, T. Schneider, T. Schwentick, and V. Weber. Complexity of hybrid logics over transitive frames. *Journal of Applied Logic*, 8(4):422–440, 2010.
37. C. Nalon, U. Hustadt, and C. Dixon. KSP: A resolution-based prover for multimodal K, Abridged Report. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017*, pages 4919–4923, 2017.
38. H. Ohlbach and R. Schmidt. Functional translation and second-order frame properties of modal logics. *Journal of Logic and Computation*, 7(5):581–603, 1997.
39. F. J. Pelletier. Seventy-five problems for testing automatic theorem provers. *Journal of Automated Reasoning*, 2(2):191–216, 1986.
40. K. Schild. A correspondence theory for terminological logics: Preliminary report. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 466–471, 1991.
41. R. A. Schmidt and U. Hustadt. A principle for incorporating axioms into the first-order translation of modal formulae. In *Proceedings of the 19th International Conference on Automated Deduction (CADE-19)*, pages 412–426, 2003.
42. G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0. *Journal of Automated Reasoning*, 59(4):483–502, 2017.
43. B. ten Cate and M. Franceschet. On the complexity of hybrid logics with binders. In *Proceedings of Computer Science Logic 2005*, pages 339–354. Springer, 2005.
44. D. Tsarkov, I. Horrocks, and P. F. Patel-Schneider. Optimizing terminological reasoning for expressive description logics. *Journal of Automated Reasoning*, 39(3):277–316, 2007.
45. J. van Eijck. Constraint tableaux for hybrid logics. Manuscript, CWI, Amsterdam, 2002.
46. J. van Eijck. HyLoTab – Tableau-based theorem proving for hybrid logics. Manuscript, CWI, Amsterdam, available at `https://homepages.cwi.nl/~jve/hylotab/Hylotab.pdf`, 2002.

47. C. Weidenbach, D. Dimova, A. Fietzke, R. Kumar, M. Suda, and P. Wischnewski. SPASS version 3.5. In *Proceedings of the 22nd International Conference on Automated Deduction (CADE-22)*, pages 140–145. Springer, 2009.

A Architecture of the Sibyl Prover

This appendix is devoted to describe the main architecture of the prover and the modules and data types used to implement the techniques described in Section 5.

A.1 Main Architecture and System Dynamics

The architectural decomposition of the system is shown in Figure 6. Here, the execution lifecycle is presented, showing the functionalities provided and the responsibilities assigned to the components of the system, as well as the interaction among them.

All the modules depend on a set of Utilities, containing modules that provide various kinds of services, and on the module Logic. The latter contains the definition of the data type representing formulae, as well as a collection of functions to manipulate them. In particular, it contains a function implementing formula preprocessing, as described in Section 3, and a test verifying whether the input formulae belong to the decidable fragment $\mathsf{HL} \setminus \Box\downarrow\Box$. Before being fed to the tableau construction procedure, formulae are simplified and transformed into negation normal form. Such operations are also under the responsibility of the module Logic. Another important task carried out by the module is nominal substitution, implemented as described in Section 5.1.

The main System control is under the responsibility of a group of modules, arranging all the necessary operations for the startup and the lifecycle of the program. It handles all aspects of Sibyl's simple command-line interface, it reads, if present, a configuration file defining rule application priorities, it initializes the computation environment and collaborates with the input/output handling modules. Moreover, it initializes all the data structures in memory that will be used in the computation. In particular, it builds the initial segment of the tableau and passes it to the main module handling tableau expansion, i.e. Search.

Tableau branches are expanded one at a time, in a depth-first manner, until either a contradiction is found (the branch closes) or the branch is complete. The Search module keeps a stack of the branches to be expanded, pops them out when the currently explored branch closes, and pushes new branches into the stack when the branching $\vee$-rule is applied. Its direct collaborators are the modules Branch and Rules: while the selected branch can be expanded, it is passed to the Branch module, which selects the correct rule instance to be fired and module Rules applies it. When a branch closes, Search *backjumps* and pops out another branch stored in the stack (the backjumping mechanism is described in Section A.4).

The core of the system is made up by the modules cooperating in branch expansion, that can be partitioned into two classes: the modules taking care of the Rule system and those constituting the Branch structure.

A.2 Rule System

In the Rule system, Premisses defines the type used to represent (sets of) tableau nodes [and assertions] constituting suitable premisses for a rule application,[14] so each element of this type represents a possible rule application. The representation of an application of the A rule, beyond its "main" premiss, contains the name of the focused nominal. Each rule is associated a *priority* (either the default one or the priority read from the configuration file),

[14] A node is a value of the main data type defined in the module Node, described later on.

so that Premisses can compare two rule applications accordingly. A priority queue of possible rule applications is in fact part of the Branch structure, and Premisses helps Branch in its management.

The main responsibility of the module Rules is firing a selected rule application and modifying the current branch accordingly. When the current branch forks, Rules returns the two branches to Search, which stores them both into the stack and resumes the first one to be expanded further. If the application of the selected rule causes a clash, Rules informs Search, communicating also the dependency points of the clash, used by Search for backjumping (see Section A.4).

A.3 Branch Structure

The Branch structure groups modules containing all the elements constituting a tableau branch and the functions to handle its dynamics. Each module in this group defines an abstract data type, whose values will be referred to with the same name of the corresponding module.

The lowest-level module of this group is Node. A Node structure represents a tableau node and its main responsibility is to encapsulate its state. Node values play an important role by supporting other modules, and supplying them with the state information they need. A Node has a unique numeric identifier (identifiers reflect node order in the branch), and maintains the label of the node. Moreover, a set of fields supports operations taking into account the restrictions on the applicability of the expansion rules:

- flags signaling whether the node is directly blocked or a phantom;
- the node's relatives according to the offspring relation;
- a list of nodes whose labels have the same structure of its own label, that are therefore potentially blockers for it; in other terms a node $(n)F$ belongs to the list of potential blockers of $(m)G$ whenever there exists a matching θ such that $\theta(F) = G$ (matchings are defined in Section 4.2).
- a flag *frozen*, that is set when the node cannot be added to the branch because of restriction **R1** (but can possibly be resumed afterwards), and a list of nodes that are "frozen" by the current node for the same reason;
- a flag *deferred*, that is set whenever the node label has the form $a{:}\,\mathsf{A}F$ and no minor premiss for its expansion currently exists (all the potential ones being phantoms); the node cannot therefore be expanded because of restriction **R3** (but may be resumed afterwards).

Finally, an optimization support field stores the branching points (in the form of a list of branch identifiers) causing the presence of the node in the branch. They are the *dependency points* that will be used on backjumping, when a clash is detected involving the current node.

A Nominal structure represents a nominal occurring in the branch. It stores the nodes labeled by satisfaction statements where the nominal is the outermost one,[15] as well as those where the nominal simply occurs, and the nodes labelled by relational formulae involving it. Moreover, a Nominal dynamically maintains its top/non-top status and caches a list of compatible nominals.

The functionalities of the module Nominal include clash detection, which is "internal" to nominals (a clash is caused by a pair of nodes of the form $a{:}\,p$ and $a{:}\,\neg p$, or a node of the form $a{:}\,\neg a$). Furthermore, since the module also maintains a list of all the nominals occurring in the branch, it can determine whether a matching $\{a_1 \mapsto b_1, \ldots, a_n \mapsto b_n\}$ is a *mapping* for the branch.

Let us recall that a partial injective function $\theta = \{a_1 \mapsto b_1, \ldots, a_n \mapsto b_n\}$ maps a formula F to G if:

- $\theta(F) = G$, i.e. F and G have the same structure;

[15] Propositional letters and formulae of the form $\Box_R F$ labelled by the nominal are stored in specific fields, so as to ease compatibility checks.

– for all i, a_i and b_i are compatible non-top nominals.

The first property is structural and pertains to formulae while the second pertains to nominals. Accordingly, the responsibility of checking the existence of a mapping between two formulae is split between Logic and Nominal.

Branch defines the abstract data type of branches in a tableau. A Branch-typed value stores most of the current state of the computation. It maintains the list of its nodes and the nominals occurring in it, and is responsible for the rule expansion queue. A Branch is a complex structure whose main components are:

– a unique identifier used to implement backjumping.
– The set of all the Nodes constituting the branch. The set of nodes labelled by formulae of the form $a\colon \mathsf{A}F$ is stored apart so that they can be easily retrieved when new nominals are added to the branch.
– The set of the directly or indirectly blocked Nodes, that have not been expanded but might be resumed afterwards.
– The set of Nominals occurring in the branch.
– A priority queue containing the applicable rules (Premisses-typed values), i.e. suitable, yet-to-be-processed rule premisses.
– The set of Nodes labelled by formulae of the form $a\colon \mathsf{A}F$ presently lacking a minor premiss, whose application is *deferred*. The applicability of the A rule to such nodes is checked again before terminating the branch construction.
– Structures devoted to help the branch in enforcing the restrictions on branch expansion: a Blocker, a Freezer and a Pairer. The corresponding modules are described below.

It is worth pointing out that, at first, a node is inserted in the expansion queue as if it were the premiss of a single premiss rule. Then, when, for instance, a node labelled by a relational formula is dequeued, all the possible tuples of companion premisses are looked for (by module Rules) and enqueued again. When a previously (directly or indirectly) blocked node is unblocked, it re-enters the queue, and this mechanisms deals with the dynamic applicability of all the expansion rules, except for the A rule.

A Blocker, whose role and main functionalities are described in Section 5.2, is a hash table where keys are *denominalized formulae*, i.e. formulae where nominals are replaced by a special "dummy" one, so that a denominalized formula represents a formula structure. The values associated to each of such formula structures are lists of blockable nodes (in descending order, i.e. higher node identifiers first) whose labels have the structure determined by the key. Consequently, the labels of all the nodes in the same list have the same structure. Every node is potentially blocked by those that occur later in the list (i.e. before it in the branch), depending on whether a mapping actually exists. Nodes enter the data structure as soon as they are created.

A Freezer, whose role is described in Section 5.2, is concretely implemented by a hash table. The values of the table are pairs of lists $\langle \mathit{freezing}, \mathit{frozen} \rangle$. All freezing and frozen nodes in the same map entry share the exact same label; thus the pairs are indexed and retrieved by the labels of the nodes they store. A Freezer is updated as soon as the frozen status of a node may have changed, due to some freezing node becoming a phantom, as a consequence of an update of the corresponding Blocker.

Finally, a Pairer, the branch helper managing applications of the A rule, is concretely implemented by a hash table of records, where the keys are the Node identifiers of A-nodes and each record contains, beyond the corresponding A-node, the information described in Section 5.3.

A.4 Backjumping

This section gives a brief description of the (standard) mechanism used to implement backjumping. As already seen, each Branch is identified by a unique, progressive number, called its identifier. Branching points are labelled by the identifier of the right branch (pushed into the stack by Search) and each Node has a *dependency set*, that contains the branching points on which it depends, determined according to the following rules: a node always inherits

the dependency set(s) of the node(s) that have caused its addition to the branch; when the $\vee$ rule is applied and the branch splits, the identifier of the right branch is added to the dependency set of the node added to the left; finally, when the equality rule is applied to a node n, every node whose outermost nominal is affected by the substitution inherits the dependency set of n.

When a branch closes because a contradiction is detected, the dependency sets of the Nodes involved in the clash are merged in descending order; the first dependency point identifies the branch where the search will resume.