Università degli Studi di Roma Tre
Dipartimento di Informatica e Automazione
Via della Vasca Navale, 79 – 00146 Roma, Italy.

Two tableau provers for basic hybrid logic

Marta Cialdea Mayer,[1] Serenella Cerrito,[2]
Emanuele Benassi,[1] Fabio Giammarinaro,[1]
Chiara Varani[1]

RT-DIA-145-2009 **March 2009**

[1] Università di Roma Tre, Italy
[2] Ibisc, FRE CNRS 3190, Université d'Evry Val d'Essonne

ABSTRACT

This work compares two internalized tableau calculi for HL(@), both terminating without loop-checking and with any rule application strategy. The two systems were independently proposed, respectively, by Bolander and Blackburn [1] and Cerrito and Cialdea Mayer [2]. The comparison is carried out both from the theoretical point of view and on the practical side. The two calculi bear strong similarities that are highlighted in the paper. They do differ, however, in the treatment of nominal equalities, which in [1] is elegant and simple, while in [2] is more technically involved, using, in fact, explicit substitution and nominal deletion. As a matter of fact, nominal deletion is the crucial difference with the treatment of equalities in the tableau system for hybrid logic previously proposed by van Eijck [10].

In order to evaluate the impact of the different approaches to nominal equalities of the considered calculi, they have been implemented and their performances compared. This work describes the implementations and the results of the empirical evaluation, which shows that substitution and nominal deletion, although unelegant from the theoretical point of view, has meaningful practical advantages.

1 Introduction

This work analyses and compares two internalized tableau calculi for HL(@), both terminating without loop-checking and with no restriction on rule application ordering. The two calculi were independently proposed, respectively, by Bolander and Blackburn [1] and Cerrito and Cialdea Mayer [2]. The first of the two tableau systems will be called P in the sequel, the second H. As usual, the notation H(@) stands for basic hybrid logic, endowed with unary direct modal operators, nominals and the satisfaction operator.[1]

The two calculi P and H are compared in order to highlight their respective similarities and differences, both from a theoretical and from a practical point of view. The main difference between the two calculi is the treatment of nominal equalities, i.e. formulae of the form $@_a b$ where b is a nominal, that is essentially carried out by means of an elegant and simple rule in P, which copies formulae labelled by a to the equal nominal b (Id rule), while H uses a more technically involved rule, requiring explicit substitution (Sub rule).

Section 2 briefly introduces the syntax and semantics of the considered logic. The core of the theoretical comparison between P and H is Section 3, where differences and similarities between the respective rules treating nominal equalities are illustrated. In order to better explain the nature of Sub, we also briefly compare such a rule with a related one proposed in [10] (*Nominal Substitution*), that however, differently from Sub, does not guarantee termination w.r.t. every rule application strategy.

The practical impact of the different approaches to nominal equalities of the considered calculi has been evaluated by means of two systems, implementing, respectively, P and H. Section 4 describes the implementations, mostly in an abstract fashion, by means of an application-oriented reformulation of the expansion rules. In Section 5 their performances are evaluated on a series of empirical tests. The experiments show that the approach to nominal equalities in H, although theoretically less elegant than P's, has important practical advantages. The systems are also briefly compared to other implemented tableau provers for HL(@). Section 6 concludes this paper pointing out directions for future work.

2 Syntax and semantics of $HL(@)$

In this section, we present the syntax and semantics of the "uni-modal" version of $HL(@)$, but obviously it is straightforward to extend the definitions to the multimodal case.

Let Nom and P be disjoint sets of propositional letters. Elements of Nom are called *nominals*. We shall use lowercase letters from the beginning of the alphabet, possibly with indexes, as metavariables for nominals. An *atom* is an element of $Nom \cup P$. The set of formulae in $HL(@)$ is defined by the following grammar:

$$F := p \mid \neg F \mid F \wedge F \mid F \vee F \mid \Box F \mid \Diamond F \mid @_a F$$

where p is an atom and a a nominal.

An *interpretation* $\mathcal{M}$ is a quadruple $\langle W, R, N, I \rangle$ where W is a non-empty set (whose elements are called the *states* of the interpretation), $R \subseteq W \times W$ (called the *accessibility relation*), N is a function $Nom \to W$ and I is a function $W \to 2^P$. We shall write wRw' as a shorthand for $\langle w, w' \rangle \in R$.

[1]The calculi presented here are uni-modal, although the comparison easily carries over into their multi-modal versions.

If $\mathcal{M} = \langle W, R, N, I\rangle$ is an interpretation, $w \in W$ and F a formula, the relation $\mathcal{M}, w \models F$ ($\mathcal{M}$ satisfies F at w) is inductively defined as follows:

1. $\mathcal{M}, w \models p$ if $p \in I(w)$, for $p \in P$.
2. $\mathcal{M}, w \models a$ if $N(a) = w$.
3. $\mathcal{M}, w \models \neg F$ if $\mathcal{M}, w \not\models F$.
4. $\mathcal{M}, w \models F \wedge G$ if $\mathcal{M}, w \models F$ and $\mathcal{M}, w \models G$.
5. $\mathcal{M}, w \models F \vee G$ if either $\mathcal{M}, w \models F$ or $\mathcal{M}, w \models G$.
6. $\mathcal{M}, w \models \Box F$ if for each w' such that wRw', $\mathcal{M}, w' \models F$.
7. $\mathcal{M}, w \models \Diamond F$ if there exists w' such that wRw' and $\mathcal{M}, w' \models F$.
8. $\mathcal{M}, w \models @_a A$ if $\mathcal{M}, N(a) \models A$.

A formula F is *satisfiable* if there exist an interpretation $\mathcal{M}$ and a state w of $\mathcal{M}$, such that $\mathcal{M}, w \models F$. Two formulae F and G are logically equivalent ($F \equiv G$) iff for every interpretation $\mathcal{M}$ and state w of $\mathcal{M}$, $\mathcal{M}, w \models F$ if and only if $\mathcal{M}, w \models G$.

It is worth pointing out that, for any nominal a and formula F:

$$\neg @_a F \equiv @_a \neg F \qquad \neg\Diamond F \equiv \Box\neg F \qquad \neg\Box F \equiv \Diamond\neg F$$

This allows one to restrict attention to formulae in negation normal form (where negation dominates only atoms) withous loss of generality.

3 A presentation of the calculi P and H

Tableaux can be presented in two different styles: either labelling each tableau node by a single formula or by a set of formulae (in a sequent-like fashion). The first style is suited to systems where there are no rules affecting the whole branch (or the whole tableau). Its easiest instance is the usual presentation of classical propositional tableaux, where, for example, the rules that expand conjunctions and disjunctions are presented as follows:

$$\frac{F \wedge G}{\begin{array}{c} F \\ G \end{array}}\ (\wedge) \qquad\qquad \frac{F \vee G}{F \qquad G}\ (\vee)$$

This is also the case of P, the calculus presented in [1], where in fact nodes are labelled by single formulae.

In the second presentation style, what is a branch $\mathcal{B}$ in the first style becomes a node, containing all the formulae in $\mathcal{B}$. The leaves of the tableau correspond to the branches of the first style, and intermediate nodes represent an history of the expansions. In this setting, the classical rules for conjunction and disjunction are reformulated as follows:

$$\frac{F \wedge G, S}{F, G, F \wedge G, S}\ (\wedge) \qquad\qquad \frac{F \vee G}{F, F \vee G, S \qquad G, F \vee G, S}\ (\vee)$$

Here, comma-separated sequences of formulae represent sets of formulae, S is a set of formulae and comma represents set union, i.e., for instance $F, F \vee G, S$ stands for $\{F, F \vee G\} \cup S$. This approach is suited to systems where some rule not only adds formulae to the branch, but modifies the existing ones, for example deleting some of them. The rule on the left below is an instance of a modifying rule in modal tableaux [3, 5]; the rule on the right is the "next" rule in tableaux for linear temporal logic [12]. Both rules represent the transition from the analysis of a state to a new one.

$$\frac{\Diamond F, S}{F, \{G \mid \Box G \in S\}}\ (\Diamond) \qquad\qquad \frac{S, \bigcirc F_1, ..., \bigcirc F_k}{F_1, ..., F_k}\ (\bigcirc)$$

(The "next" rule actually assumes that S is a set of literals). Since H, the calculus presented in [2], has a rule with "side effects", it is formulated in this style.

In order to ease the comparison between the two calculi, they will be both formulated in the "nodes as sets" style. It is worth pointing out, moreover, that such a style is in some sense closer to implementation, for two reasons. First of all, automated tableau constructions do not generally keep in memory the whole tree, but the expansion loop acts on sets of formulae (the "active" formulae in the branch). Secondly, the "nodes as sets" approach allows one to focus on which formulae can be forgotten after expansion. For instance, the two boolean rules above could be restated as shown below, meaning that the expanded formula will not be needed any more in the branch.

$$\frac{F \wedge G, S}{F, G, S}\ (\wedge) \qquad\qquad \frac{F \vee G}{F, S \qquad G, S}\ (\vee)$$

We shall depart from the original presentation of [1] also in three other respects. Firstly, in order to reduce the number of expansion rules, we assume that input formulae are in negation normal form (nnf). Secondly, we eliminate the so-called $\neg$ rule:

$$\frac{@_a \neg b}{@_b b}$$

and replace it with a stronger notion of branch closure. Finally, in this work we consider only unary modalities and one accessibility relation.

In the presentation that follows, tableau nodes are labelled by sets of *satisfaction statements*, i.e. assertions of the form $@_a F$. Obviously, some rule must necessarily conserve its premise(s) to ensure completeness, and a memory of already expanded formulae must be hold in order to avoid trivial cases of non-termination. As a matter of fact, in both [1] and [2] it is assumed that a formula is never added to a branch (a node) where it already occurs and that rules generating new nominals are never applied twice to the same premise on the same branch. In this section, differently from the next one, rules are formulated so that their application preserves all formulae in the premise, except for the only rule with side effects of H (the *Sub* rule). In Section 4 we shall turn to consider how such a memory can be economized.

The initial tableau for a set S of formulae is a node labelled by $S_a = \{@_a A \mid A \in S\}$, where a is a new nominal. S_a is called the *root set*. Nominals occurring in S_a are called *root nominals*.

Table 1 contains the expansion rules which are common to the two systems. The $\Diamond$ rule is subject to some restrictions that will be given later on.

Boolean Rules	
$\dfrac{@_a(F \wedge G), S}{@_aF, @_aG, @_a(F \wedge G), S}\ (\wedge)$	
$\dfrac{@_a(F \vee G), S}{@_aF, @_a(F \vee G), S \qquad @_aG, @_a(F \vee G), S}\ (\vee)$	
Label Rule	
$\dfrac{@_a@_bF, S}{@_bF, @_a@_bF, S}\ (@)$	
Modal rules	
$\dfrac{@_a\Box F, @_a\Diamond b, S}{@_bF, @_a\Box F, @_a\Diamond b, S}\ (\Box)$	$\dfrac{@_a\Diamond F, S}{@_a\Diamond b, @_bF, @_a\Diamond F, S}\ (\Diamond)$ where b is a new nominal
Closure rules	
$\dfrac{@_aF, S}{\bot}\ (\bot_1)$ if $@_a\overline{F} \in S$	$\dfrac{@_a\neg a, S}{\bot}\ (\bot_2)$

Table 1: Expansion rules common to the two systems. In the $\bot_1$ rule, $\overline{F}$ is the nnf of $\neg F$.

A tableau node S is *closed* if it contains $\bot$ (see the Closure Rules of Table 1). A tableau branch is open if all its nodes are open (otherwise it is closed) and it is *complete* if no rule can be applied to expand it further. A tableau is closed if all its branches are closed, otherwise it is open.

A formula of the form $@_aF$ will be called *labelled by a*. If $@_aF \in S$, where S is a tableau node, we say that F is true at a in S. A formula of the form $@_a\Diamond b$, where b is a nominal, is a *relational formula*. A relational formula generated by application of the $\Diamond$-rule is called and *accessibility formula*.

If a nominal b is introduced in a branch Θ by application of the $\Diamond$-rule to a premise of the form $@_a\Diamond F$, then $a \prec_\Theta b$ (and we say that b is a child of a). The relation ${\prec_\Theta}^*$ is the transitive closure of $\prec_\Theta$.[2] If $a \prec_\Theta^* b$ we say that b is a descendant of a and a an ancestor of b in the branch Θ.

The system P restricts the applicability of the $\Diamond$ rule to cases where its premise $@_a\Diamond F$ is not an accessibility formula, while in H it is restricted to cases where $@_a\Diamond F$ is not a relational formula.

[2]It is worth pointing out that, in [1], ${\prec_\Theta}^*$ denotes the *reflexive* and transitive closure of $\prec_\Theta$.

The essential difference between the two calculi consists however in the treatement of "nominal equalities", i.e. formulae of the form $@_a b$, where b is a nominal. In P, such formulae are expanded by means of the two premises rule Id:

$$\frac{@_a F, @_a b, S}{@_b F, @_a F, @_a b, S}\ (Id)$$

The rule is applicable only if $@_a F$ is not an accessibility formula. The general requirement that a formula is never added to a node where it already occurs forbids firing the Id rule twice on the same pair of formulae.

The system H treats equalities by means of a more complex (and less elegant) rule, with side effects, the Substitution rule (Sub), that is applicable only if $a \neq b$:

$$\frac{@_a b, S}{S^{\#}[a \mapsto b]}\ (Sub)$$

In such a rule, $S^{\#}[a \mapsto b]$ is obtained from S by:

1. deleting every formula containing a descendant of a;
2. replacing every occurrence of a with b.

When the substitution rule is applied, a is said to be *replaced in the branch* and the descendants of a are called *deleted in the branch.* Let's point out that root nominals are never deleted (they cannot be children of any nominal), although obviously they may be replaced.

The two rules for the treatement of equalities are apparently very different. However, they bear strong similarities, that can also be recognized by inspection of the respective termination and completeness proofs. First of all, we observe that in both rules equalities are "directional": $@_a b$ is not the same as $@_b a$. Now, the following key property holds in both systems: for any formula $@_a F$ (other than accessibility formulae) occurring in a tableau node, F is a subformula of a formula in the root set (and therefore does not contain any non-root nominal). As a consequence, in the premise $@_a b$ of the Id and Sub rules, b is a root nominal. This means that only root nominals can "inherit" formulae from other (equal) ones.

Forbidding the application of the Id rule to accessibility formulae means that the nominal b inherits all formulae true at a in the node, except for those representing outcoming links to its children. The Substitution rule directly replaces (instead of copying) a with b, but, again, its children are not "adopted" by b. They are deleted, together with all their descendants. In both cases we have that, treating an equality of the form $@_a b$:

- the children of a must not become children of b (this is necessary in order to ensure termination);
- in the branch, b can safely "take the place of" a.

The second assertion is literally true for H, while, in the case of P, its meaning is a little subtler. In fact, in the completeness proof, both a and b are "represented" by a same nominal c (that may or may be not one of a or b), and only the descendants of c are of use when showing how to build a model from a complete and open tableau branch. Hence, b can take the place of a in the sense that a will be represented by the same nominal as b. In any case, at least one between a and b (and its respective descendants) could be ignored.

So, in simple words we can say that the main difference between the two systems is that P is more tolerant than H: even when the descendants of a nominal are of no use any longer, they are left alive, since they are not harmful, either. H, on the contrary, is radical and bloody: when a nominal becomes useless, it is killed with all its descent.

The reason why, in both systems, when expanding an equality $@_a b$, the children of a must not be adopted by b is a crucial property used in the respective termination proofs. If S a node (a branch, in [1]) and b a nominal occurring in S, the measure of b w.r.t. S, $m_S(b)$, is the maximal length of a formula F such that $@_b F \in S$. The important property used to prove termination is that, for any nominal a and c, if c is a child of a, then $m_S(c) < m_S(a)$. If, when processing $@_a b$, the nominal b inherited also accessibility formulae – i.e. for any child c of a, $@_b \Diamond c$ were derivable –, then, the Box rule could be applied to expand formulae of the form $@_b \Box F$ and $@_b \Diamond c$, generating $@_c F$. This could make $m_S(c)$ greater than $m_S(a)$.

Figures 1 and 2 show in fact that, without the restriction on the *Id* rule, or without nominal deletion in the *Sub* rule, respectively, infinite tableaux can be built. In the figures, the expansion rule applied at each step and the formula(e) to which it is applied are shown on the right. The modified *Id* and *Sub* rules are marked by an asterisk. It is easy to check that, with the Id rule of P and the Substitution rule of H, the same rule application strategy used in Figures 1 and 2 builds finite tableaux for the same root set.

It is worthwile pointing out that the tableau in Figure 2 is a tableau in the internalized calculus for H(@) proposed in [10]. There, in fact, nominal equalities are handled by means of a substitution rule that differs from *Sub exactly because nominal deletion is not performed.* As a consequence, such a calculus does not terminate w.r.t. *every* rule application strategy.

Nominal deletion in H has a practical advantage, avoiding the employment of resources to expand formulae labelled by "useless" nominals. It can therefore be conjectured that the treatment of nominal equalities in H might be more efficient than in P. In order to verify such an hypothesis, the calculi P and H have been implemented, as described in the next section. The experimental results are presented in Section 5.

4 Implementation issues

In this section we describe an implementation of the two tableaux calculi considered in this paper: the system Pilate ("What crime has he committed?") implements the calculus P, while Herod is the implementation of the slaughter of the innocents represented by H. The two systems are implemented in Objective Caml [9] and are available at `http://cialdea.dia.uniroma3.it/herod/`.

As we have seen in the previous section, some memory of already expanded formulae has to be maintained in order to ensure both completeness and termination. Here, we turn in particular to consider how to economize on the number of formulae to be kept in memory, so as to lighten also the membership tests required by some rule. The description of the systems will abstract from many details and a reformulation of the expansion rules

$$
\begin{array}{c l}
@_a\Diamond(a\wedge\Diamond p), @_a\Box\Diamond(a\wedge\Diamond p) & \\ \hline
@_a\Diamond b_0, @_{b_0}(a\wedge\Diamond p), @_a\Diamond(a\wedge\Diamond p), @_a\Box\Diamond(a\wedge\Diamond p) & \Diamond : @_a(a\wedge\Diamond p) \\ \hline
@_{b_0}a, @_{b_0}\Diamond p, @_{b_0}(a\wedge\Diamond p), @_a\Diamond b_0, & \wedge : @_{b_0}(a\wedge\Diamond p) \\
@_a\Diamond(a\wedge\Diamond p), @_a\Box\Diamond(a\wedge\Diamond p) & \\ \hline
@_{b_0}\Diamond b_1, @_{b_1}p, @_{b_0}a, @_{b_0}\Diamond p, @_{b_0}(a\wedge\Diamond p), & (\Diamond : @_{b_0}\Diamond p) \\
@_a\Diamond b_0, @_a\Diamond(a\wedge\Diamond p), @_a\Box\Diamond(a\wedge\Diamond p) & \\ \hline
@_a\Diamond b_1, @_{b_0}\Diamond b_1, @_{b_1}p, @_{b_0}a, @_{b_0}\Diamond p, @_{b_0}(a\wedge\Diamond p), & (Id^* : @_{b_0}a, @_{b_0}\Diamond b_1) \\
@_a\Diamond b_0, @_a\Diamond(a\wedge\Diamond p), @_a\Box\Diamond(a\wedge\Diamond p) & \\ \hline
@_{b_1}\Diamond(a\wedge\Diamond p), @_a\Diamond b_1, @_{b_0}\Diamond b_1, & (\Box : @_a\Diamond b_1, @_a\Box\Diamond(a\wedge\Diamond p)) \\
@_{b_1}p, @_{b_0}a, @_{b_0}\Diamond p, @_{b_0}(a\wedge\Diamond p), & \\
@_a\Diamond b_0, @_a\Diamond(a\wedge\Diamond p), @_a\Box\Diamond(a\wedge\Diamond p) & \\ \hline
@_{b_1}\Diamond b_2, @_{b_2}(a\wedge\Diamond p), @_{b_1}\Diamond(a\wedge\Diamond p), @_a\Diamond b_1, & (\Diamond : @_{b_1}\Diamond(a\wedge\Diamond p)) \\
@_{b_0}\Diamond b_1, @_{b_1}p, @_{b_0}a, @_{b_0}\Diamond p, @_{b_0}(a\wedge\Diamond p), & \\
@_a\Diamond b_0, @_a\Diamond(a\wedge\Diamond p), @_a\Box\Diamond(a\wedge\Diamond p) & \\ \hline
@_{b_2}a, @_{b_2}\Diamond p, @_{b_1}\Diamond b_2, @_{b_2}(a\wedge\Diamond p), @_{b_1}\Diamond(a\wedge\Diamond p), & (\wedge : @_{b_2}(a\wedge\Diamond p)) \\
@_a\Diamond b_1, @_{b_0}\Diamond b_1, @_{b_1}p, @_{b_0}a, @_{b_0}\Diamond p, @_{b_0}(a\wedge\Diamond p), & \\
@_a\Diamond b_0, @_a\Diamond(a\wedge\Diamond p), @_a\Box\Diamond(a\wedge\Diamond p) & \\ \hline
@_{b_2}\Diamond b_3, @_{b_3}p, @_{b_2}a, @_{b_2}\Diamond p, @_{b_1}\Diamond b_2, @_{b_2}(a\wedge\Diamond p), & (\Diamond : @_{b_2}\Diamond p) \\
@_{b_1}\Diamond(a\wedge\Diamond p), @_a\Diamond b_1, @_{b_0}\Diamond b_1, @_{b_1}p, @_{b_0}a, @_{b_0}\Diamond p, & \\
@_{b_0}(a\wedge\Diamond p), @_a\Diamond b_0, @_a\Diamond(a\wedge\Diamond p), @_a\Box\Diamond(a\wedge\Diamond p) & \\ \hline
@_a\Diamond b_3, @_{b_2}\Diamond b_3, @_{b_3}p, @_{b_2}a, @_{b_2}\Diamond p, @_{b_1}\Diamond b_2, @_{b_2}(a\wedge\Diamond p), & (Id^* : @_{b_2}a, @_{b_2}\Diamond b_3) \\
@_{b_1}\Diamond(a\wedge\Diamond p), @_a\Diamond b_1, @_{b_0}\Diamond b_1, @_{b_1}p, @_{b_0}a, @_{b_0}\Diamond p, & \\
@_{b_0}(a\wedge\Diamond p), @_a\Diamond b_0, @_a\Diamond(a\wedge\Diamond p), @_a\Box\Diamond(a\wedge\Diamond p) & \\ \hline
\vdots & (\Box : @_a\Diamond b_3, @_a\Box\Diamond(a\wedge\Diamond p))
\end{array}
$$

Figure 1: An infinite tableau in P without the restriction on the Id rule

in an implementation-oriented fashion will be used.

It is common practice in the implementation of a tableau system that cannot delete all its expanded formulae to represent a tableau node by means of a structure containing a set of formulae that still have to be processed and a *memory*, storing information that cannot be forgotten. In both Pilate and Herod, the memory is itself a set of formulae, which can either be re-used for expansions or used to restrict some rule application in the sequel. In the abstract representation of node structures we are going to use, a node is represented by a set of formulae, containing both *pending* formulae, that still have to be processed, and the memorized ones. The latter will be marked by an asterisk.

The boolean rules, the label rule and (obviously) the closure rules are destructive in both systems: the main formula in the premise can safely be deleted. In fact, it is not necessary to expand such formulae more than once in a branch and they are not needed for other purposes. The destructive rules are represented in Table 2, where S stands for

$$
\begin{array}{cl}
@_a\Diamond(a\wedge\Diamond p),\ @_a\Box\Diamond(a\wedge\Diamond p) & \\
\cline{1-1}
@_a\Diamond b_0,\ @_{b_0}(a\wedge\Diamond p), & (\Diamond : @_a\Diamond(a\wedge\Diamond p)) \\
@_a\Diamond(a\wedge\Diamond p),\ @_a\Box\Diamond(a\wedge\Diamond p) & \\
\cline{1-1}
@_{b_0}a,\ @_{b_0}\Diamond p,\ @_a\Diamond b_0,\ @_{b_0}(a\wedge\Diamond p), & (\wedge : @_{b_0}(a\wedge\Diamond p)) \\
@_a\Diamond(a\wedge\Diamond p),\ @_a\Box\Diamond(a\wedge\Diamond p) & \\
\cline{1-1}
@_{b_0}\Diamond b_1,\ @_{b_1}p,\ @_{b_0}a,\ @_{b_0}\Diamond p,\ @_a\Diamond b_0, & (\Diamond : @_{b_0}\Diamond p) \\
@_{b_0}(a\wedge\Diamond p),\ @_a\Diamond(a\wedge\Diamond p),\ @_a\Box\Diamond(a\wedge\Diamond p) & \\
\cline{1-1}
@_a\Diamond b_1,\ @_{b_1}p,\ @_a\Diamond p,\ @_a\Diamond a,\ @_a(a\wedge\Diamond p), & (Sub^* : @_{b_0}a) \\
@_a\Diamond(a\wedge\Diamond p),\ @_a\Box\Diamond(a\wedge\Diamond p) & \\
\cline{1-1}
@_{b_1}\Diamond(a\wedge\Diamond p),\ @_a\Diamond b_1,\ @_{b_1}p,\ @_a\Diamond p,\ @_a\Diamond a, & (\Box : @_a\Diamond b_1, @_a\Box\Diamond(a\wedge\Diamond p)) \\
@_a(a\wedge\Diamond p),\ @_a\Diamond(a\wedge\Diamond p),\ @_a\Box\Diamond(a\wedge\Diamond p) & \\
\cline{1-1}
@_{b_1}\Diamond b_2,\ @_{b_2}(a\wedge\Diamond p),\ @_{b_1}\Diamond(a\wedge\Diamond p), & (\Diamond : @_{b_1}\Diamond(a\wedge\Diamond p)) \\
@_a\Diamond b_1,\ @_{b_1}p,\ @_a\Diamond p,\ @_a\Diamond a, & \\
@_a(a\wedge\Diamond p),\ @_a\Diamond(a\wedge\Diamond p),\ @_a\Box\Diamond(a\wedge\Diamond p) & \\
\cline{1-1}
@_{b_2}a,\ @_{b_2}\Diamond p,\ @_{b_1}\Diamond b_2,\ @_{b_2}(a\wedge\Diamond p), & (\wedge : @_{b_2}(a\wedge\Diamond p)) \\
@_{b_1}\Diamond(a\wedge\Diamond p),\ @_a\Diamond b_1,\ @_{b_1}p,\ @_a\Diamond p,\ @_a\Diamond a, & \\
@_a(a\wedge\Diamond p),\ @_a\Diamond(a\wedge\Diamond p),\ @_a\Box\Diamond(a\wedge\Diamond p) & \\
\cline{1-1}
@_{b_2}\Diamond b_3,\ @_{b_3}p,\ @_{b_2}a,\ @_{b_2}\Diamond p,\ @_{b_1}\Diamond b_2, & (\Diamond : @_{b_2}\Diamond p) \\
@_{b_2}(a\wedge\Diamond p),\ @_{b_1}\Diamond(a\wedge\Diamond p),\ @_a\Diamond b_1,\ @_{b_1}p,\ @_a\Diamond p, & \\
@_a\Diamond a,\ @_a(a\wedge\Diamond p),\ @_a\Diamond(a\wedge\Diamond p),\ @_a\Box\Diamond(a\wedge\Diamond p) & \\
\cline{1-1}
@_a\Diamond b_3,\ @_{b_3}p,\ @_a\Diamond p,\ @_{b_1}\Diamond a,\ @_a(a\wedge\Diamond p), & (Sub^* : @_{b_2}a) \\
@_{b_1}\Diamond(a\wedge\Diamond p),\ @_a\Diamond b_1,\ @_{b_1}p, & \\
@_a\Diamond a,\ @_a\Diamond(a\wedge\Diamond p),\ @_a\Box\Diamond(a\wedge\Diamond p) & \\
\cline{1-1}
\vdots & (\Box : @_a\Diamond b_3, @_a\Box\Diamond(a\wedge\Diamond p))
\end{array}
$$

Figure 2: An infinite tableau in H without nominal deletion in the Substitution rule

a set that may contain both marked and unmarked formulae. The $\vee$-rule, where $\overline{F}$ is the nnf of $\neg F$, implements *semantic branching* [4], that is in fact used by both systems. In the $\bot_1$ rule, ℓ is a literal (that can be built either from a nominal or from a propositional letter), and $\overline{\ell}$ is its complement.

The implementation of rules with two premises (such as the Box rule and the Id rule of P) requires attention. In both Herod and Pilate, whenever a formula matching one of the premises is chosen from the pending formulae and processed, the rule is fired against any formula in the memory matching the second premise. We describe the implementation of such rules starting from Herod, since it has only one double-premise rule.

Boolean Rules	
$\dfrac{@_a(F \wedge G), S}{@_aF, @_aG, S}\ (\wedge)$	$\dfrac{@_a(F \vee G), S}{@_aF, S \qquad @_aG, @_a\overline{F}, S}\ (\vee)$

Label Rule	Closure rules	
$\dfrac{@_a@_bF, S}{@_bF, S}\ (@)$	$\dfrac{@_a\ell, S}{\bot}\ (\bot_1)$ if $@_a\overline{\ell}^* \in S$	$\dfrac{@_a\neg a, S}{\bot}\ (\bot_2)$

Table 2: Destructive rules, common to Pilate and Herod

4.1 Herod

The implementation-oriented reformulation of the non-destructive expansion rules of H, that is at the same time an abstract description of Herod, is presented in Table 3. Here too, S stands for a set that may contain both marked and unmarked formulae. In the sequel, the set of marked formulae in S will be called the memory of S. In all such rules, the main formula is unmarked in the upper node and marked in the lower one, meaning that when such a formula is processed it is moved from the set of pending formulae to the memory of the node.

Let us consider first the $\Box$-rule of Table 1. The rule is applied to a pair of formulae, none of which can be deleted, since it may be necessary to re-apply the same rule to any of them paired with a different formula. However, it must be ensured that the rule is never applied twice to the *same* pair of formulae. This can be done in different ways. A first possibility consists in storing a table of pairs of formulae to which the $\Box$-rule has already been applied, but this approach would make the memory structure (and implementations) more complicated. A second possibility is to save a copy of $@_bF$ in the history (i.e. add $@_bF^*$ to the lower node) and restrict the applicability of the rule to cases where $@_bF^* \notin S$. The approach followed by Herod, however, is to treat the $\Box$-rule as a kind of *quasi-immediate rule*: when a formula of the form $@_a\Box F$ is processed, the $\Box$-rule is applied to $@_a\Box F$ and each relational formula $@_a\Diamond b^*$ that has already been processed (and stored in the memory); and, when a relational formula $@_a\Diamond b$ is processed, the $\Box$-rule is fired to $@_a\Diamond b$ and each formula of the form $@_a\Box F^*$ that is in the node memory. The Box rules in Table 3 formalize this approach. This mechanism ensures that, for any node S, if the memory of S contains a pair of formulae $@_a\Diamond b^*$ and $@_a\Box F^*$, the $\Box$-rule of Table 1 has been applied to them above S. Note that, when no relational or $\Box$ formula is marked yet, firing any of these rules just induces a storage in memory.

Note that the Box rules of Table 3 have a single *main formula*, that which fires the application of the rule when chosen from the pending formulae. The second premise is chosen (in every possible way) from the node memory.

The $\Diamond$-rule cannot be destructive either, although for different reasons. This holds for both systems Pilate and Herod. Let us assume in fact that the $\Diamond$-rule is applied to expand $@_a\Diamond F$ and that, below in the branch, either the Id rule or the the Substitution rule (depending on the system) is applied to $@_a c$. As we have seen, in neither case are the children of a adopted by c, so, in order for the systems to be complete, the generating formula $\Diamond F$ has to be passed to c, producing, by application of one of the two rules treating equalities, $@_c\Diamond F$. In that way, c will be able to generate its corresponding child. The Diamond rule of table 3, used by Herod, formalizes this approach.

Box rules	
$\dfrac{@_a\Box F, S}{@_a\Box F^*, S, \{@_b F \mid @_a\Diamond b^* \in S\}}$ $(\Box_1)$	$\dfrac{@_a\Diamond b, S}{@_a\Diamond b^*, S, \{@_b F \mid @_a\Box F^* \in S\}}$ $(\Box_2)$
Diamond rule	**Literal rule**
$\dfrac{@_a\Diamond F, S}{@_a\Diamond b, @_b F, @_a\Diamond F^*, S}$ $(\Diamond)$ where b is a new nominal (applicable only if $@_a\Diamond F$ is not a relational formula)	$\dfrac{@_a\ell, S}{@_a\ell^*, S}$ (lit) if ℓ is not a nominal and $@_a\overline{\ell}^* \notin S$
Substitution rule	
$\dfrac{@_a b, S}{S^\dagger[a \mapsto b]}$ (Sub) (if $a \neq b$ and $@_a\neg b^* \notin S$) where $S^\dagger[a \mapsto b]$ is obtained from S by: 1. deleting every formula containing a descendant of a; 2. unmarking every formula of the form $@_a F$ that is marked in S; 3. replacing every occurrence of a with b.	

Table 3: Abstract description of the implementation of the non-destructive expansion rules in Herod

The Literal rule, together with the two closure rules, describe what happens when a formula of the form $@_a\ell$, where ℓ is a literal, is chosen for expansion: its consistency with the already memorized literals is checked and, if ℓ is not a nominal (that would fire the Substitution rule), the formula is moved to the memory.

Last comes the Substitution rule. Note, before all, that such a rule is not applicable to $@_a b$ when $@_a\neg b^*$ is in the memory: in that case the $\bot_1$ rule is applied and the node is closed. In other words, consistency of $@_a b$ with the current node memory is checked before firing the *Sub* rule. Considering the remarks about the Diamond rule made above, it is easy to realize that every formula of the form $@_a\Diamond F^*$ occurring in the upper node must be unmarked, producing, when the Substitution rule is applied, $@_b\Diamond F[a \mapsto b]$. And the same must be done with formulae of the form $@_a\Box F^*$, $@_a\Diamond c^*$ and $@_a\ell^*$. In fact, the

$\Box$-rule may still be applied to $@_b\Box F[a \mapsto b]$ and some $@_b\Diamond c^*$ in the memory, or vice-versa. Moreover, consistency of $@_b\ell[a \mapsto b]$ with the other marked formulae of the form $@_b\ell'^*$ must be checked.

Actually, in the implementation, only formulae of the form $@_b\Diamond F[a \mapsto b]^*$ are unmarked. In fact, when the Substitution rule is applied, the $\Box$-rule is immediately applied to each new formula of the form $@_b\Box F$ obtained from the substitution and each relational formula $@_b\Diamond c^*$ already present in S; it is also applied to each new formula of the form $@_b\Diamond c$ and each $@_b\Box F^* \in S$. Analogously, literals of the form $@_b\ell$ obtained from the substitution are immediately checked for consistency with the marked literals in S, and are left marked. In other terms, it is as if any application of the Substitution rule were followed by a series of applications of the $\Box_1$, $\Box_2$, *Lit*, $\bot_1$ and $\bot_2$ rules.

We describe here in some details the deletion and substitution mechanisms used by Herod. The information in a node is actually organized as a set of worlds, each of which corresponds to a nominal a and contains the formulae true at a (marked and unmarked formulae being stored in distinct fields). Moreover, each world has pointers to its children. Substitution is treated in a lazy way: a table of replacements is kept aside each node, that is updated with the application of the Substitution rule, and used whenever accessing a nominal, looking for its presently replacing nominal. It can be proved (see [2]) that every non-root nominal c can only occur in formulae of the form $@_cF$ (where F does not contain c) and in a single accessibility formula $@_a\Diamond c$, where $a \prec c$ (c is a child of a). As a consequence, when processing $@_ab$ we only need to update the replacement table, delete the world representing a and, recursively, all its descendants. In fact, every information related to a descendant c of a is contained either in c itself, or in its parent, which is either a or, in turn, a descendant of a (that will be deleted altogether).

4.2 Pilate

Let us now turn to consider Pilate. Its particular treatment of nominal equalities has been implemented in a somewhat more involved way at first, and then it has been simplified. The simplified version turned out to be also slightly more efficient than the previous one. We are going to describe here both versions.

4.2.1 First version

In the description of the first implementation of Pilate, we assume, at first, that Box, Diamond and Literal rules are the same as Herod's. Similarly to the Box rule, the rule Id is treated as an immediate rule: every time an unmarked formula of the form $@_ab$ is processed, the Id rule is fired with all the *marked and unmarked* formulae of the form $@_aF^{[*]}$ (that are not accessibility formulae) occurring in the node, and such that $@_bF$ is not in the node. Here, $@_aF^{[*]}$ means that the formula can be either marked or unmarked.

$$\frac{@_ab, S}{@_ab^*, S, \{@_bF \mid @_aF^{[*]} \in S \text{ and } @_bF \notin S\}}\ (Id_1)$$

Moreover, the information passed from a to b is immediately propagated to all nominals c_i that are "reachable" from b by means of a chain $@_bc_1^*, @_{c_1}c_2^*, ...@_{c_{i-1}}c_i^*$ in the memory of S. In other terms, it is as if any fresh application of the Id rule to an unmarked $@_ab$ were followed by a series of applications of the rule (with the same restriction to avoid loops). I.e., let $b = c_0$ and let $@_{c_0}c_1^*, @_{c_1}c_2^*, ..., @c_{n-1}c_a^*$ be any chain of "equalities" in the memory of S. Then any application of the Id_1 rule is followed by as many applications as possible of the rule:

$$\frac{@_{c_i}c_{i+1}^*, S}{@_{c_i}c_{i+1}^*, S, \{@_{c_{i+1}}F \mid @_{c_i}F \in S \text{ and } @_{c_{i+1}}F \notin S\}}\ (Id_2)$$

(note that the Id_2 rule differs from the Id_1 rule only because its premise is marked). The restriction to the applicability of the rule ensures termination also when a nominal is reachable from itself. A sequence of (correct) applications of the Id_2 rule that adds $@_{c_i}F$ for all the nominals c_i such that $@_bc_1^*, @_{c_1}c_2^*, ...@_{c_{i-1}}c_i^* \in S$ will be called a *complete chain of applications of* Id_2 *saturating equalities of* b *in* S *with respect to* F.

Next, it must be ensured that, if $@_ab^*$ is in the node, when processing new formulae of the form $@_aF$, the Id rule is fired again with $@_ab^*$. Specifically, every time the expansion of a formula $@_cG$ causes the addition of a formula $@_aF$ with $a \neq c$ (and $@_aF$ is not an accessibility formula), then the possibility to fire the Id rule with $@_aF$ as one of the premises is considered: for every $@_ab^*$ in S, if $@_bF \notin S$, then $@_bF$ is added to the lower node. Moreover, like above, the procedure is recursively repeated to $@_bF$. In other terms, it is as if any expansion of a formula $@_cG$ causing the addition of a formula $@_aF$ with $a \neq c$ (except for the $\Diamond$ rule) were followed by a complete chain of applications of the Id_2 rule saturating equalities of b in S with respect to F.

Finally, we observe that the only rules that need to be followed by a chain of applications of the Id_2 rule are the Label rule and the two Box rules. In fact, of the two formulae added by the $\Diamond$ rule, the first is an accessibility formula and in the second, $@_bF$, b is a fresh nominal that does not occur in S. In the boolean rules, if equalities are saturated on the premise, there is no need to saturate them on the conclusion(s).

4.2.2 Second version

The description of the second implementation of Pilate, being the presently adopted one, will be more accurate and all rules will be given their abstract implementation-oriented reformulation.

Since the treatment of the Id rule will cause some modifications in the other non-destructive rules, we begin by describing the implementation of such a rule. It is treated similarly to the $\Box$-rule: every time an unmarked formula of the form $@_ab$ is processed, the Id rule is fired with premises $@_ab$ and:

- $@_ab$ itself, if $a \neq b$, producing $@_bb$;
- any marked formula of the form $@_aF^*$ (except for accessibility formulae), yielding $@_bF$, provided $@_bF^*$ is not already in the node;
- any marked formula of the form $@_ac^*$ occurring in the node, yielding $@_cb$, provided $@_cb^*$ is not already in the node.

This corresponds to the Id_1 rule of Table 4. Of course, like in the case of Herod's Sub, the rule cannot be fired when $@_a\neg b^*$ is in the node.

Similarly to the $\Box$-rule, a symmetric rule is needed, but in this case the matter is more complicated: in fact, the leftmost premise of the Id rule of Section 3 can have any

Id rule
$\dfrac{@_a b, S}{@_a b^*, [@_b b]^{(1)}, S, \{@_c b \mid @_a c^* \in S \text{ and } @_c b^* \notin S\}, \{@_b F \mid @_a F^* \in S, @_a F \text{ is not an accessibility formula, and } @_b F^* \notin S\}}\ (Id_1)$ if $@_a \neg b^* \notin S$ $^{(1)}$ $@_b b$ is added only if $a \neq b$
Diamond rule
$\dfrac{@_a \Diamond F, S}{@_a \Diamond b, @_b F, @_a \Diamond F^*, S, \{@_c \Diamond F \mid @_a c^* \in S \text{ and } @_c \Diamond F^* \notin S\}}\ (\Diamond)$
Box rules
$\dfrac{@_a \Box F, S}{@_a \Box F^*, S, \{@_b F \mid @_a \Diamond b^* \in S\}, \{@_c \Box F \mid @_a c^* \in S \text{ and } @_c \Box F^* \notin S\}}\ (\Box_1)$
$\dfrac{@_a \Diamond b, S}{@_a \Diamond b^*, S, \{@_b F \mid @_a \Box F^* \in S\}}\ (\Box_2)$ if $@_a \Diamond b$ is an accessibility formula
$\dfrac{@_a \Diamond b, S}{@_a \Diamond b^*, S, \{@_b F \mid @_a \Box F^* \in S\}\{@_c \Diamond b \mid @_a c^* \in S \text{ and } @_c \Diamond b^* \notin S\}}\ (\Box_3)$ if $@_a \Diamond b$ is not an accessibility formula
Literal rule
$\dfrac{@_a \ell, S}{@_a \ell^*, S, \{@_b \ell \mid @_a b^* \in S \text{ and } @_b \ell^* \notin S\}}\ (lit)$ if ℓ is not a nominal and $@_a \overline{\ell}^* \notin S$

Table 4: Abstract description of the implementation of the non-destructive expansion rules in Pilate

form (excluding, obviously, accessibility formulae). Since $@_a b$ must necessarily be marked in the lower node of the Id_1 rule of Table 4, it must be ensured that, when processing new formulae of the form $@_a F$, the Id rule is fired again with $@_a b^*$. In fact, what is to be ensured is that for any node S, if $@_a b^* \in S$ and $@_a F^* \in S$ (where $@_a F$ is not an accessibility formula), then the Id rule of Section 3 has been applied to them above S. So, the $\Diamond$, $\Box_1$, $\Box_2$ and Lit rule used by Herod (see Table 3) have to be modified. These are in fact the rules where some formula passes from the set of pending formulae to the memory of the node (i.e. a formula is unmarked in the premise and marked in the conclusion). The corresponding rules used by Pilate are in Table 4.

We conclude this section saying that, at present, the only simple rule application strategy adopted by both Pilate and Herod consists of delaying tableau branching (i.e. applications of the $\vee$ rule) as far as possible.

5 Experimental results

In order to evaluate the performances of Pilate and Herod, the two provers have been run on a set of 1400 randomly generated sets of formulae (not necessarily in CNF), approximately equally partitioned into satisfiable and unsatisfiable. The test size, in number of logical operators, is up to 4000 and their modal depth (greatest number of nested modal operators) varies from 0 to 22. The experiments were run on an Intel Pentium 4 3GHz, with 3Gb RAM, running under Linux, and the provers were given 1 minute timeout.

The table below summarizes the results. It can be noted that Pilate has about 20% cases of error (i.e. out of either time or memory) more than Herod. The average time (in seconds) has been computed on the set of 1071 tests that both provers solved in the allowed time and space.

	Pilate	Herod
Total errors (timout/memory out)	306	259
Timeout	302	259
Out of memory	4	0
Average time	0.60	0.10

The diagram in Figure 3 shows how the average execution time of each system varies in relation to the modal depth of the tests.

Maybe more interesting is the comparison between the two systems on a set of hand-written formulae which involve many $\Diamond$'s and equalities, where the differences in treating equalities should be pushed to the limit. The formulae we have used have the form:

$$@_{a_1}\Diamond^n(@_{a_1}a_2 \wedge \ldots \wedge @_{a_n}a_{n+1} \wedge \Diamond F)$$

where $\Diamond^n$ is a sequence of n $\Diamond$'s, dominating n nominal equalities. F is an unsatisfiable formula of modal degree 2, with 3 nominals and 3 propositional letters, generating a four-branches complete tableau. The size of such formulae is taken to be n. The results are represented by the diagram in Figure 4. As it can be seen, Pilate can only solve problems up to size 100 in the allowed time of one minute, and its execution time increases exponentially.

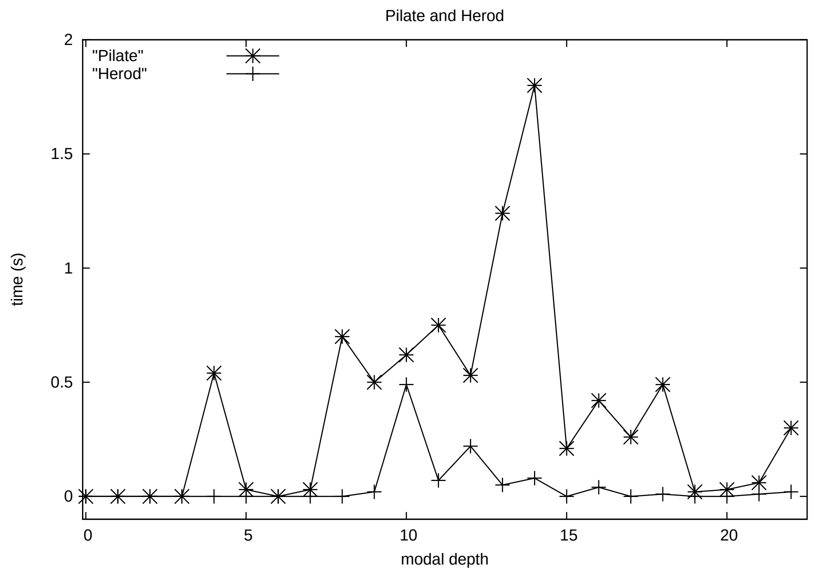

Figure 3: Performances of Pilate and Herod on random formulae

The empirical results shown above reflect what one could have expected, i.e. that Pilate consumes, in general, more time and space resources than Herod.

It must be remarked, however, that Pilate constitutes a more or less "ad literam" implementation of P and that more effective ways of treating its Id rule can be conceived. For instance, a different implementation of such a rule could use equivalence classes of *right nominals*, i.e. nominals c such that, for some nominal d, $@_d c$ occurs in the branch. In fact the completeness proof of P shows that the relation $a \sim_\Theta b$, holding between two nominals a and b iff $@_a b$ occurs in the branch Θ, is an equivalence relation on right nominals. Such an approach would certainly reduce many redundancies. However, any faithful implementation of P would consume resources in processing formulae labelled by all the descendants of equal nominals, while only a single set of such descendants is needed to decide the root set.

Although Herod is only at its first stage of development, it has been compared with other more mature provers for hybrid logic, based on tableau calculi. First of all, it has been compared with HTab [7], implementing the prefixed calculus presented in [1]. HTab implements two rules for prefix equality (that are related to restricted forms of the Id rule of P) by means of equivalence classes of nominals and prefixes, that are created, enlarged and merged while the tableau construction proceeds. Moreover, many redundancies are avoided by processing only formulae true at the representative of each class (if the prefix σ is the representative of the class of τ, conclusions of the form $\tau : F$ are directly replaced by $\sigma : F$ – except, as usual, for accessibility formulae). Such methodology, in effect, closely resembles substitution. However, in HTab like in Pilate, the descendants of any nominal are always expanded (unless they are in turn affected by nominal equality), thus consuming time and space.

HTab decides hybrid logic formulae that may also contain the universal modality (besides multiple unary modalities), and, in order to terminate, it implements a blocking condition requiring the computation of the inclusion ordering on the sets of formulae that hold at each prefix on the branch. However, the system uses no more than what is

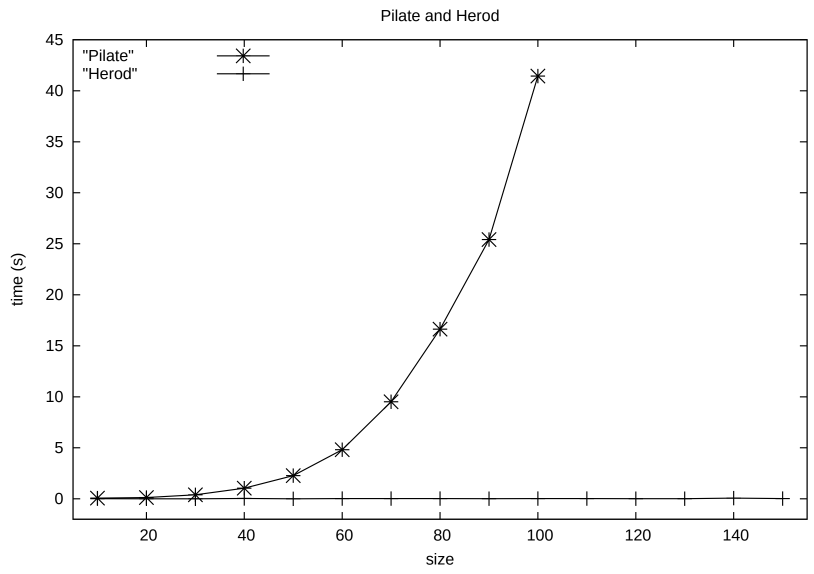

Figure 4: Performances of Pilate and Herod on hand-written formulae

needed, depending on the input formula: if it does not contain the universal modality, such expensive computations are not carried out. Therefore, the fact that HTab is a prover for a more extensive logic, should not affect its performances on H(@).

Herod has been compared with HTab and the two provers appear to have comparable behaviours. The experimental results on the randomly generated formulae are reported in the table below. The average execution time has been computed wrt the 1062 tests which both systems solved in the allowed time (and space).

	Htab	Herod
Total errors (timout/memory out)	243	259
Timeout	227	259
Out of memory	15	0
Average time	0.28	0.20

In interpreting the figures reported above, one has to consider, on one side that the systems are based on different compilers,[3] and, on the other side, that HTab implements many important optimization strategies, such as, for instance, backjumping – paired with a rule application strategy that takes into account branching dependencies –, which, according to the authors, improved the system behaviour of an order of magnitude.

In [7], HTab has been compared with HyLoTab [11], that is reported to be far behind HTab. So, although we have not experimentally verified, HyLoTab is presumably also less efficient than Herod.

Finally, the tableau prover for hybrid logic Spartacus [6] has been considered. It is based on the calculi presented in [8] (and previous works by the same authors), whose approach to nominal equalities differs from P, H and the prefixed tableaux of [1], on which HTab is based. In the calculi underlying Spartacus, equality is in fact handled abstractly:

[3]The relative efficiency of Haskell, the compiler used by Htab, and Objective Caml, is debated. See for instance:
`http://augustss.blogspot.com/2007/11/benchmarking-ray-tracing-haskell-vs.html`

a congruence on nominals, based on the equality assertions present in a branch, is defined, but the systems does not commit to a particular representation of the congruence closure induced by the equations.

Spartacus has been run on the same set of randomly generated tests. Its average execution time – on the 1137 cases where none of the two systems (Spartacus and Herod) went in timeout – is about 22% of Herod's and run out of time only in 58 cases. Spartacus is by far the more mature of the systems we are considering. In fact, it schedules pending rule applications using a configurable priority queue, which allows for a fine-grained control over the rule application strategy, and implements a number of optimizations, besides semantic branching, which include term normalization, boolean constraint propagation, backjumping and a technique called lazy branching.

The relative performances of the three provers, Herod, HTab and Spartacus, however, seem to be significantly different when run on the set of hand-written formulae mentioned above. The interest of such tests relies on the fact that they are meant not so much to compare the provers in themselves, rather their different treatments of nominal equalities. The results are shown in Figure 5 and suggest that, although an important refinement and optimisation work still has to be done, Herod's approach to nominal equalities deserves some interest.

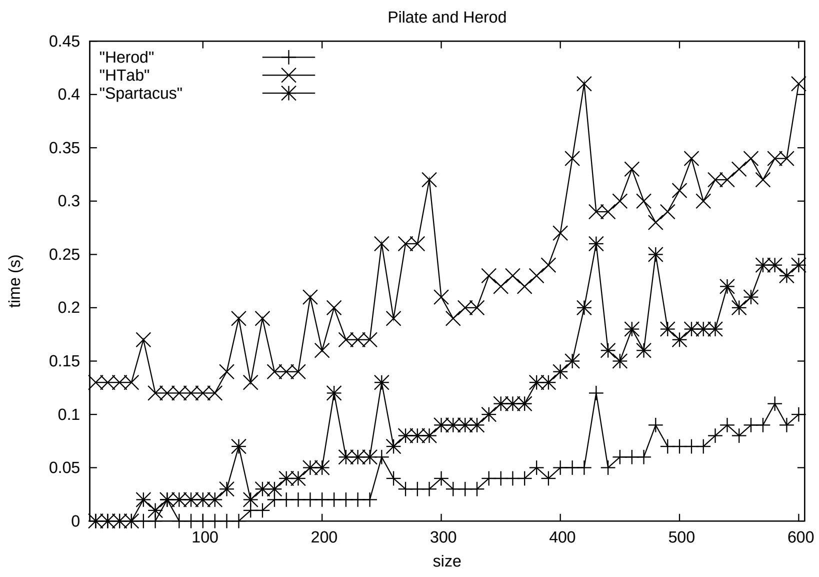

Figure 5: Performances of Herod, Htab and Spartacus on the set of hand-written formulae

As a general conclusion, it must be remarked that, in each pairwise comparison of the considered systems, the cases of error of one of them are never a subset of the errors of the other. It would be interesting to study and characterize the classes of tests that are better solved by each prover.

6 Concluding remarks

The experimental results described in Section 5 are encouraging: although Herod is still at a very early stage of development, the very nature of the treatment of nominal equalities in

H, that allows one to economize on the number of generated nominals, already gives rather good results. Therefore it seems worthwhile to go on and refine its implementation, both by some routine work that still can be done, as well as by studying and experimenting more effective rule application strategies and the implementation of basic optimisation techniques.

Moreover, in order to make the promising child grow up, extensions of H to handle at least different accessibility relations and the global modalities have to be studied and implemented. Behind implementation, a theoretical study is needed in order to check that the existing algorithms treating the global modalities can be adapted to H. In fact, in [1] for instance, they are dealt with only in the context of prefixed tableaux. A similar remark applies to possible extensions covering also converse modalities. Treating the difference modality (like in [8]) might reveal to be even more delicate.

Another subject of future work is the study of extensions of internalized calculi so as to cover the transitive closure of accessibility relations (similarly to propositional dynamic logic) and some restricted use of the binder (still allowing for terminating tableaux), Here, as far as we know, really new ideas and techniques are needed.

References

[1] T. Bolander and P. Blackburn. Termination for hybrid tableaus. *Journal of Logic and Computation*, 2007.

[2] S. Cerrito and M. Cialdea Mayer. Terminating tableaux for HL(@) without loop-checking. Technical Report IBISC-RR-2007-07, Ibisc Lab., Université d'Evry Val d'Essonne, 2007. Available at `http://www.ibisc.univ-evry.fr/Vie/TR/2007/IBISC-RR2007-07.pdf`.

[3] M. Fitting. *Proof Methods for Modal and Intuitionistic Logics*. Reidel Publishing Company, 1983.

[4] F. Giunchiglia and R. Sebastiani. A SAT-based decision procedure for ALC. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR'96)*. Morgan Kaufmann, 1996.

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

[6] D. Götzmann, M. Kaminski, and G. Smolka. Spartacus: A tableau prover for hybrid logic. Technical report, Saarland University, 2009.

[7] G. Hoffmann and C. Areces. HTab: A terminating tableaux system for hybrid logic. In *Proceedings of Methods for Modalities 5*, November 2007.

[8] M. Kaminski and G. Smolka. Terminating tableaux for hybrid logic with the difference modality and converse. In A. Armando, P. Baumgartner, and G. Dowek, editors, *Proc. of IJCAR 2008*, volume 5195 of *LNAI*, pages 210–225. Springer, 2008.

[9] X. Leroy. The Objective Caml system, release 3.11. Documentation and user's manual. `http://caml.inria.fr/`, 2008.

[10] J. van Eijck. Constraint tableaux for hybrid logics. Manuscript. CWI, Amsterdam, 2002.

[11] J. van Eijck. HyLoTab – Tableau-based theorem proving for hybrid logics. Manuscript, CWI, Amsterdam. Available at `http://www.cwi.nl/~jve/hylotab/Hylotab.pdf`, 2002.

[12] P. Wolper. The tableau method for temporal logic: An overview. *Logique et Analyse*, 1985.