Herod and Pilate: two Tableau Provers for Basic Hybrid Logic

Serenella Cerrito

Lab. Ibisc
Université d'Evry Val d'Essonne,
France

Marta Cialdea Mayer

Dipart. Informatica e Automazione
Università di Roma Tre
Italy

This is a draft version of a paper published on the Proceedings of IJCAR 2010. It should not be cited, quoted or reproduced.

Abstract

This work presents two provers for basic hybrid logic *HL*(@), which have been implemented with the aim of comparing the internalised tableau calculi independently proposed, respectively, by Bolander and Blackburn [3] and Cerrito and Cialdea Mayer [5]. Experimental results are reported, evaluating, from the practical point of view, the different treatment of nominal equalities of the two calculi.

1 A Brief Presentation of the Calculi P and H

The treatment of nominal equalities in proof systems for hybrid logics may easily induce many redundancies. In fact, when processing a statement of the form $@_a b$, any known property of a can potentially be copied to b (and vice-versa). This work compares, from a practical point of view, two different approaches to nominal equalities, represented by two internalised tableau calculi for *HL*(@), both terminating without loop-checking and with no restriction on rule application strategies. The two calculi were independently proposed, respectively, in [3] and [5] (a revised and extended version of the latter is [4]). The main difference between them is the treatment of nominal equalities, that is essentially carried out by means of an elegant and simple rule in the first calculus ([3], here named P), which copies formulae labelled by a to the equal nominal b (the *Id* rule), while the second ([4, 5], the calculus H) uses a more technically involved rule, requiring explicit substitution (the *Sub* rule).

An analysis and comparison of P and H, highlighting their respective similarities and differences from a theoretical point of view can be found in [7], where also an application-oriented, though abstract, reformulation of the expansion rules can be found. This presentation describes the implementations and some experimental results, run on sets of formulae randomly generated by hGen [1].

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. We omit here the definition of the syntax and semantics of *HL*(@), which can be found in any work on hybrid logic.

Since the provers only deal with the uni-modal version of *HL*(@), the (necessarily brief and informal) description of the tableau systems that follows is restricted accordingly. Differently from [4, 5, 7], for the sake of simplicity, tableaux are presented here in the "nodes as formulae" style. Assuming that input formulae are in negation normal form, tableau nodes in both P and H are labelled by *satisfaction statements*, i.e. formulae of the form $@_aF$, where F is in negation normal form. The initial tableau for a set S of formulae is a single branch tableau whose nodes are labelled by $@_aF$, for each $F \in S$, where a is a new nominal. The set $\{@_aF \mid F \in S\}$ is called the *initial set.*

The two systems share the following expansion rules:

$$\frac{@_a(F \wedge G)}{@_aF, @_aG}\ (\wedge) \qquad \frac{@_a(F \vee G)}{@_aF \quad | \quad @_aG}\ (\vee) \qquad \frac{@_a@_bF}{@_bF}\ (@)$$

$$\frac{@_a\Box F, @_a\Diamond b}{@_bF}\ (\Box) \qquad \frac{@_a\Diamond F}{@_a\Diamond b, @_bF}\ (\Diamond) \text{ where } b \text{ is new in the branch}$$

The $\Diamond$-rule is subject to some restrictions that will be given later on.

It is assumed that a formula is never added to a branch where it already occurs and that the $\Diamond$-rule, which generates new nominals, is never applied twice to the same premise on the same branch. A tableau branch is closed if it contains either $@_ap$ and $@_a\neg p$ for some nominal a and atom p, or $@_a\neg a$ for some nominal a.

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 an *accessibility formula.* 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.

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 we say that b is a child of a, and use the notation $a \prec_\Theta b$. The relation ${\prec_\Theta}^+$ is the transitive closure of $\prec_\Theta$. 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 essential difference between the two calculi consists in the treatment of nominal equalities. In P, such formulae are expanded by means of the two premises rule *Id*, which is applicable only if $@_aF$ is not an accessibility formula:

$$\frac{@_aF, @_ab}{@_bF}\ (Id)$$

The system H treats equalities by means of a more complex rule, with side effects, the Substitution rule (*Sub*). When expanding a formula of the form $@_ab$ (where $a \neq b$) by means of *Sub* in a branch Θ, the whole branch is modified as follows: every occurrence of a is replaced by b, and every formula containing a descendant of a is deleted.

The two rules for the treatment of equalities are apparently very different. However, they bear strong similarities, which are highlighted in [7]. We only observe here that the restriction on the *Id* rule, and nominal deletion in the *Sub* rule, are crucial to ensure strong termination of the respective systems. And their role is similar: avoiding the "adoption" of a replaced nominal's children by the replacing one. 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.

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.

2 Pilate and Herod

The system Pilate ("What crime has he committed?") implements the calculus P, and Herod is the implementation of the slaughter of the innocents represented by H. The two systems are implemented in Objective Caml [12] and are available at Herod web page [13]. In the rest of this section a description of the implementations is given, with some simplifications and abstractions.

Both systems take as input a file specifying a set of hybrid formulae and build a tableau for them in a depth-first manner. If a complete and open branch is found, then a model of the initial set of formulae is extracted from it and given as output. Otherwise, the set is declared unsatisfiable. At any stage of the search process, the systems consider a single *active branch*, the others being kept in a stack, where the branching points are stored and retrieved upon backtracking.

A branch Θ is represented by a set (implemented by a hash table) of *worlds*. Each world w corresponds to a nominal occurring in the branch and is a structure (a record) with the following fields: *name(w)* stores the nominal a corresponding to w; *pending(w)* contains the formulae labelled by a (*a=name(w)*) in Θ, that still have to be processed; *memory(w)* contains the formulae labelled by a in Θ that have already been processed but have to be kept in memory for future use (i.e. literals and formulae of the form $\Box F$ and $\Diamond F$); *children(w)* is a set of pointers to the children of a in Θ, i.e. the nominals b such that $a \prec_\Theta b$.

In the implementation of Herod, beyond the set of its worlds, each branch is associated a *table of replacements*, that is updated with the application of the substitution rule, and used whenever accessing a nominal, looking for its presently replacing nominal. Substitution is in fact treated in a lazy way. In the following, we shall denote the nominal replacing a by *replaces(a)*, meaning that *replaces(a)* $= a$ if a has not been replaced in the branch.

The expansion loop obeys the following basic principle: for each world w, *memory(w)* is "saturated" with respect to any rule application, in the sense that every formula (or pair of formulae) in *memory(w)* has been expanded. At each stage, a world w is chosen for expansion and a formula F selected from *pending(w)*. According to the form of F, different operations are performed.

- If F is a literal ℓ, then its consistency is checked with respect to the literals in *memory(w)*. If the branch is closed, then the systems backtrack;

otherwise, if ℓ is not a nominal, it is moved to *memory(w)* and if it is a nominal, the respective rules for equalities, described below, are fired.

- If F is either a disjunction, or a conjunction or a satisfaction statement, its expansion (or, in the case of a disjunction, one of its expansions, the other being recorded in the stack) is added to *pending(w)* and F is deleted.

The treatment of the other rules differs in the two systems. We begin by describing Herod.

H1. The expansion of a formula of the form $\Diamond F$ in a world w causes the moving of $\Diamond F$ from *pending(w)* to *memory(w)*, the creation of a new world w' with $F \in$*pending(w′)*, and the addition of w' to *children(w)*.

H2. The $\Box$-rule in Herod is fired whenever the selected formula has the form of one of its premises: if it has the form $\Box F$, then the $\Box$-rule is applied to it and each formula of the form $\Diamond b \in$*memory(w)*. Moreover, it is applied to each $\Diamond b$ implicitly represented by *children(w)*. Symmetrically, if the extracted formula has the form $\Diamond b$, then the $\Box$-rule is applied to it and each formula of the form $\Box F \in$*memory(w)*. The obtained expansions are in both cases added to *pending(w)*.

H3. Finally, if the selected formula is a nominal b, the substitution rule is fired. If w' is the world representing b, then: (a) every formula in *pending(w)* is copied to *pending(w′)*; (b) every formula of the form $\Diamond F \in$*memory(w)* is copied to *pending(w′)*; (c) every formula of the form $\Box F \in$*memory(w)* is copied to *memory(w′)* and the $\Box$-rule is fired against each $\Diamond c \in$*memory(w′)* and the children of w'; (d) the literals in *memory(w)* are added to *memory(w′)*, after a consistency check. Finally, the table of replacements is updated, and the world w and, recursively, all its descendants are deleted. This suffices to implement nominal deletion; in fact, every information related to a descendant w'' of w is contained either in w'' itself, or in its parent, which is either w or, in turn, a descendant of w (that is deleted altogether).

Let us now turn to consider the implementation of the Id rule in Pilate. The basic principle is always that the set in *memory(w)* has to be maintained saturated with respect to every rule application. Therefore:

P1. When an equality $@_a b$ is processed, i.e. a nominal b is chosen from *pending(w)* for some world w with *name(w)=a*, it is moved to *memory(w)* and the Id rule is fired against $@_a b$ and: (a) any already processed formula of the form $@_a F$ that is not an accessibility formula, yielding $@_b F$ (i.e. F is added to to the *pending* field of the world w' representing b – provided that F is not already in *memory(w′)*); (b) any already processed formula of the form $@_a c$, producing $@_c b$ (i.e. for all nominal c in *memory(w)*, b is added to the *pending* field of the world w' representing c – provided that b is not already in *memory(w′)*); and, finally (c) Id is applied to $@_a b$ and $@_a b$ itself, if $a \neq b$ (producing $@_b b$, i.e. b is added to the *pending* field of the world representing b); such an operation is in fact necessary for completeness. The membership tests are needed in order to avoid that, in the presence of a looping chain of equalities, formulae are copied forever, passing from the memory of a node to the pending formulae of another one and then back to the pending formulae of the first one.

The symmetric case is subtler, since the leftmost premise of the Id rule can have any form. Pilate's treatment consists in firing the Id rule against any memorised equality every time a formula is moved to *memory(w)*:

P2. when processing a literal or a formula of the form $\Box F$ or $\Diamond F$ chosen from a world w with *name(w)=a*, beyond the operations performed by Herod (H1 and H2), the Id rule is fired against such a formula and any already processed equality of the form $@_a b$ (i.e. any nominal in *memory(w)*, and the results are added to the *pending* field of the world representing b).

3 Experimental Results

The relative performances of Herod and Pilate have been evaluated running the provers on a set of 1600 sets of formulae, randomly generated by hGen [1], and approximately equally partitioned into satisfiable and unsatisfiable. The modal depth (greatest number of nested modal operators) of the tests varies from 10 to 40, and the number of clauses varies from 60 to 200. The sets of formulae used for the benchmarks and the parameters used for their generation can be found at Herod web page [13]. The experiments were run on an Intel Pentium 4 3GHz, with 3Gb RAM, running under Linux, and the provers were given 1 minute timeout.

Considering the 1274 tests that both Pilate and Herod solved in the allowed time, Pilate is in the average more than 50 times slower than Herod, and the median run time of Pilate is more than 5 times Herod's one.[1] Moreover, Pilate runs out of time almost 50% more often than Herod. The diagram on the left in Figure 1 plots the average run time of the two systems against the number of clauses of the tests solved by both provers. In the the diagram, points on the X-axis group all sets with the same number of clauses, independently of their modal depth. Pilate seems to be more sensible to the phase transition in the easy-hard-easy pattern of the benchmarks [8].

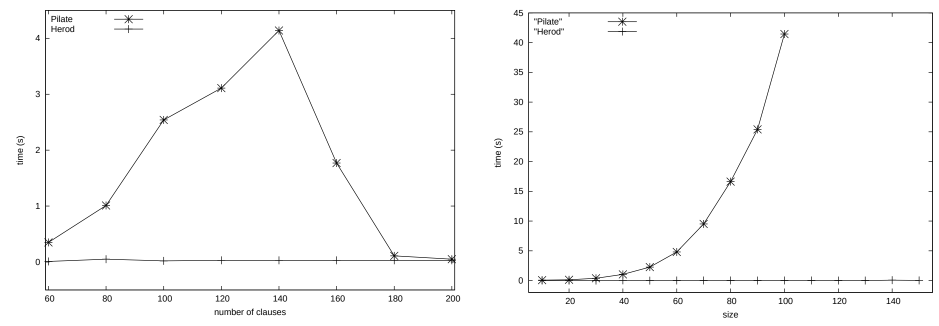

Figure 1: Pilate and Herod average run times

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 $@_{a_1}\Diamond^n(@_{a_1}a_2 \wedge ... \wedge @_{a_n}a_{n+1} \wedge \Diamond F)$, where $\Diamond^n$ is a

[1]The median times are computed counting timeouts as values greater than all the others.

sequence of n $\Diamond$'s, dominating n nominal equalities, and F is a (non trivially) unsatisfiable formula. Processing such formulae forces the provers to generate n new worlds before processing the equalities. The size of such formulae is taken to be n. The results are represented by the diagram on the right in Figure 1. As can be seen, Pilate can only solve problems up to size 100 in the allowed time of one minute, while Herod solved the problems up to the maximal tested size (600), within 0.11 seconds.

The empirical results described above confirm that Pilate consumes, in general, more resources than Herod. It is important to point out, moreover, that the different performances are effectively due to the different treatment of equalities. In fact, on a set of 400 modal formulae (without nominals and satisfaction operators) randomly generated by hGen, of modal depth varying from 30 to 70, the two provers had the same cases of timeouts and the same average and median execution times. The same results, showing that that all the difference between the systems is due to their treatment of equalities, are obtained when running the two provers on the hand-tailored collection of modal formulae proposed in [2], where in fact Pilate and Herod show the same behaviour.

It must be remarked, however, that Pilate constitutes a more or less "ad litteram" implementation of P and that more effective ways of treating its Id rule can be conceived. For this reason, Herod has been compared with HTab [11], which implements the prefixed calculus presented in [3]. HTab implements equality by means of equivalence classes of nominals and prefixes, that are created, enlarged and merged while the tableau construction proceeds. Many redundancies are avoided by processing only formulae true at the representative of each class. However, in HTab, *the descendants of any nominal are always expanded,* thus consuming time and space. The comparison has also considered another prover, Spartacus [10].[2] Spartacus processes nominal equalities by merging the content of the corresponding "nodes", and electing one of them as the representative of both [9]. Both HTab and Spartacus are much more mature provers than Herod, handling a richer logic and implementing many important optimisation strategies. On the contrary, at present, the only simple rule application strategy adopted by Herod, beyond semantic branching, consists of delaying tableau branching as far as possible. And in fact, HTab and Spartacus behave definitely much better than Herod when run on the set of hand-tailored collection of modal formulae presented in [2], as well as on hybrid formulae of a very low modal depth (personal communication by the maintainer of HTab). The experiments reported below aimed at testing whether Herod gains any advantage from its treatment of equalities.

The three provers were run on the same sets of random formulae used for the comparison with Pilate. As can be seen in the tables below, HTab could solve 95% of the tests in the allowed time (one minute) and space (7% more than Herod), and Spartacus failed to solve only one test in the allowed one minute time. Although the number of Herod's failures is the highest one, surprisingly enough its median run time is much better than HTab's and Spartacus's. Moreover, the average execution times on all the problems (modal depth 10 to 40) solved by both Herod and HTab are in favour of Herod. In the average, moreover, the execution times of Herod and Spartacus are comparable.

[2]In the tests, HTab 1.4.0 and Spartacus 1.0.1 were used, both run with the respective default options.

modal	Number of failures			Median times		
depth	Herod	HTab	Spartacus	Herod	HTab	Spartacus
10	56	22	1	0.02	0.07	0.07
20	47	12	0	0.02	0.14	0.12
30	44	24	0	0.03	0.20	0.16
50	42	24	0	0.04	0.29	0.21
total	189	82	1	0.03	0.16	0.13

Average times						
	Herod vs HTab			Herod vs Spartacus		
modal	number of tests	Average time		number of tests	Average time	
depth	solved by both	Herod	HTab	solved by both	Herod	Spartacus
10	340/400	0.16	0.12	344/400	0.16	0.06
20	348/400	0.03	0.21	353/400	0.03	0.11
30	347/400	0.03	0.27	356/400	0.04	0.16
50	346/400	0.25	0.31	358/400	0.24	0.20
total	1381/1600	0.12	0.23	1411/1600	0.12	0.13

The efficiency of Herod's treatment of equalities is apparent when comparing the three systems on the set of hand-written formulae earlier defined. The interest of such tests relies on the fact that they are meant not so much to compare the provers in themselves, but rather their different treatments of nominal equalities. The provers were run on formulae up to size 600. Herod employed 10 seconds to solve the problem of maximal size, while the execution times of Spartacus and HTab were, respectively, of 24 and 28 seconds.

4 Concluding Remarks

The experimental results are encouraging and suggest that, although an important refinement and optimisation work still has to be done, Herod's approach to nominal equalities is algorithmically interesting. In fact, although Herod is still at a very early stage of development, the very nature of the treatment of nominal equalities in H already gives rather good results, thanks to nominal deletion that allows one to economize on the number of processed nominals. Therefore it seems worth going 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, Herod has to be extended to handle different accessibility relations and the global and converse modalities [6].

Acknowledgements. The very first versions of Herod and Pilate were developed as a bachelor project at "Università di Roma Tre" by Emanuele Benassi, Fabio Giammarinaro and Chiara Varani.

References

[1] C. Areces and J. Heguiabehere. hGen: A random CNF formula generator for hybrid languages. In *Methods for Modalities 3 (M4M-3)*, Nancy, France, 2003.

[2] 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.

[3] T. Bolander and P. Blackburn. Termination for hybrid tableaus. *Journal of Logic and Computation*, 17(3):517–554, 2007.

[4] S. Cerrito and M. Cialdea Mayer. An efficient approach to nominal equalities in hybrid logic tableaux. *Journal of Applied Non-classical Logics*. To appear.

[5] 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. (`http://www.ibisc.univ-evry.fr/Vie/TR/2007/IBISC-RR2007-07.pdf`).

[6] S. Cerrito and M. Cialdea Mayer. Tableaux with substitution for hybrid logic with the global and converse modalities. Technical Report RT-DIA-155-2009, Dipartimento di Informatica e Automazione, Università di Roma Tre, 2009.

[7] M. Cialdea Mayer, S. Cerrito, E. Benassi, F. Giammarinaro, and C. Varani. Two tableau provers for basic hybrid logic. Technical Report RT-DIA-145-2009, Dipartimento di Informatica e Automazione, Università di Roma Tre, 2009.

[8] 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.

[9] D Götzmann. Spartacus: A tableau prover for hybrid logic. Master's thesis, Saarland University, 2009.

[10] D. Götzmann, M. Kaminski, and G. Smolka. Spartacus: A tableau prover for hybrid logic. In *M4M6*, number 128 in Computer Science Research Reports, pages 201–212. Roskilde University, 2009.

[11] G. Hoffmann and C. Areces. HTab: A terminating tableaux system for hybrid logic. *Electronic Notes in Theoretical Computer Science*, 231:3–19, 2009. Proceedings of the 5th Workshop on Methods for Modalities (M4M-5), 2007.

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

[13] Herod web page. `http://cialdea.dia.uniroma3.it/herod/`, 2009.