Hydra: a tableau-based prover for basic hybrid logic (System Report)

Patrick Blackburn	Aljoscha Burchardt	Stephan Walter
Langue et Dialogue	Computerlinguistik	Computerlinguistik
INRIA, Lorraine	Saarland University	Saarland University
patrick@aplog.org	albu@coli.uni-sb.de	stwa@coli.uni-sb.de

November 23, 2001

1 Introduction

Basic hybrid logic is an extension of modal logic in which it is possible to name states and to assert that a formula is true at a named state. It does this with the help of *nominals* and *satisfaction operators*. Nominals are special propositional variables (typically written i, j, k, and l), that are true at exactly one point in any model: nominals 'name' the unique point they are true at. Satisfaction operators have the form $@_i$, where i is a nominal. The meaning of a formula of the form $@_i\phi$ is "ϕ is true at the point named by i". For more on hybrid logic, see the Hybrid Logic Manifesto [2], Chapter 7 of Blackburn, de Rijke and Venema [3], or take a look at `www.hylo.net`, the hybrid logic webpage.

Hydra is a tableau-based prover for the basic hybrid language written in SWI Prolog. It was implemented by Aljoscha Burchardt and Stephan Walter as an undergraduate programming project under the supervision of Patrick Blackburn. The goals of the project were to gain experience in designing and implementing a larger piece of software, and to gain a deeper knowledge of Prolog. It was decided that (a) writing a propositional logic tableau prover in Prolog that was more efficient than Fitting's [4], and then (b) extending this to a prover for the basic hybrid language, would be a good way of meeting these goals. Hydra was the result.

Hydra is not — and was not intended to be — competitive with standard modal theorem provers. Its interest lies in the fact that (as far as we are aware) it was the first prover ever implemented for basic hybrid logic: it can prove interesting theorems displaying the interplay between nominals, satisfaction operators and the orthodox operators. In what follows we briefly describe the type of tableau reasoning performed by Hydra, make some remarks on Hydra's implementation, and note an important problem. Hydra can be experimented with over the internet: it is accessible from the hybrid logic homepage (`www.hylo.net`).

2 The tableau system

Hydra is based on the tableau systems described in [1, 2]. Space restrictions make a systematic exposition impossible here, but the following example should make the basic ideas clear. Consider the formula

$$\Diamond(i \wedge p) \wedge \Diamond(i \wedge q) \rightarrow \Diamond(p \wedge q).$$

This is valid, and hence should be provable. It is:

1		$\neg @_j(\Diamond(i \wedge p) \wedge \Diamond(i \wedge q) \rightarrow \Diamond(p \wedge q))$	
2		$@_j(\Diamond(i \wedge p) \wedge \Diamond(i \wedge q))$	$1, \neg\rightarrow$
$2'$		$\neg @_j \Diamond(p \wedge q)$	Ditto
3		$@_j \Diamond(i \wedge p)$	$2, \wedge$
$3'$		$@_j \Diamond(i \wedge q)$	Ditto
4		$@_j \Diamond k$	$3, \Diamond, k$
$4'$		$@_k(i \wedge p)$	Ditto
5		$@_k i$	$4', \wedge$
$5'$		$@_k p$	Ditto
6		$@_j \Diamond l$	$3', \Diamond, l$
$6'$		$@_l(i \wedge q)$	Ditto
7		$@_l i$	$6', \wedge$
$7'$		$@_l q$	Ditto
8		$@_i l$	$7'$, Sym
9		$@_k l$	$5, 8$, Nom
10		$@_k q$	$9, 7'$, Nom
11		$\neg @_k(p \wedge q)$	$2', 4, \neg\Diamond$
12	$\neg @_k p$	$\mid$ $\neg @_k q$	$11, \neg\wedge$
	✠ $5', 12$ ✠	✠ $10, 12$ ✠	

- Every item in the tableau is a formula prefixed by a satisfaction operator. The system works by using the nominals and satisfaction operators to make assertions, and draw conclusions, about specific named states.

- For example, the first line is an assertion that the formula we are trying to prove is false at an arbitrary state, which we have christened j. We prove the formula by showing that this is impossible.

- Note that line 2 has the form $@_j(\phi \wedge \psi)$ (that is "at j a conjunction holds") and that the conclusions we draw from this (at lines 3 and $3'$) have the form $@_j\phi$ and $@_j\psi$ (that is, "at j both conjuncts must hold").

- The most characteristic rule is the $\Diamond$ rule. The formula at line 3 has the form $@_j\Diamond\phi$. The conclusions we draw from this (at lines 4 and $4'$) are (a) $@_j\Diamond k$, and (b) $@_k\phi$. Now, as k is a nominal that makes its first appearance at line 4, in effect we are saying: from $@_j\Diamond\phi$ conclude that (a) there is some successor of j which we christen k, and (b) at k ϕ holds.

- The $\neg\Diamond$ rule works by branch saturation. From a formula of the form $\neg@_j\Diamond\phi$ ("no successor of j makes ϕ true") and $@_j\Diamond k$ ("k is a successor of j") we conclude $\neg@_k\phi$ ("ϕ is not true at k"). This rule is applied at line 11 to lines $2'$ and 4.

Note that the proof also makes use of two rules, namely Sym (from $@_i j$ deduce $@_j i$) and Nom (from $@_i j$ and $@_j p$ deduce $@_i p$) which *aren't* tableau-style rules. (In fact the full system contains four such rules, the other two being Ref and Bridge). In essence, these are rewrite rules. Adding nominals and satisfaction operators to modal logic is in effect to enrich modal logic with an 'equational logic' of states and state succession. The Sym, Nom, Ref and Bridge rules embody this equational logic, and it would be more accurate to describe the system not as a pure tableau system, but as as a tableau + rewrite system. This has undesirable consequences for computational efficiency that we discuss below.

3 Hydra

As we mentioned earlier, Hydra grew out of a propositional theorem prover designed to improve on the performance of the Prolog provers presented in Fitting [4]. Three main changes were made to Fittings' design:

1. Fitting represents branches as lists, and a tableau as a list of such lists. Hydra works with a tree representation. Every formula is associated with an explicit path (coded as a series of left and right moves) from the root of the tree.
2. Hydra maintains tableau by asserting and retracting information into the Prolog database, not by list manipulation.
3. Hydra always tests for branch closure before it adds information to the tableau.

Hydra is typically much faster at propositional logic than the Fitting prover. This seems to be mainly due to the closure strategy. (Whether database manipulation is more efficient than list manipulation seems to depend on the Prolog implementation used. With Sicstus, timing experiments revealed no significant difference. With SWI, database manipulation yielded a useful advantage.)

The benefits of the explicit tree representation made themselves felt when the prover was extended to the basic hybrid language. Several of the rules used (for example, $\neg\Diamond$ and Nom) take two inputs from a branch and then calculate an output (in essence, they are branch saturation rules). None of the propositional rules work this way. The explicit path representation makes it possible to perform the lookup required for such rules by matching the path patterns. Paths are searched from leaf to root. A predicate `leaf/1` indexes the leaf node of each path. Retraction of such a formula codes branch closure.

4 A problem

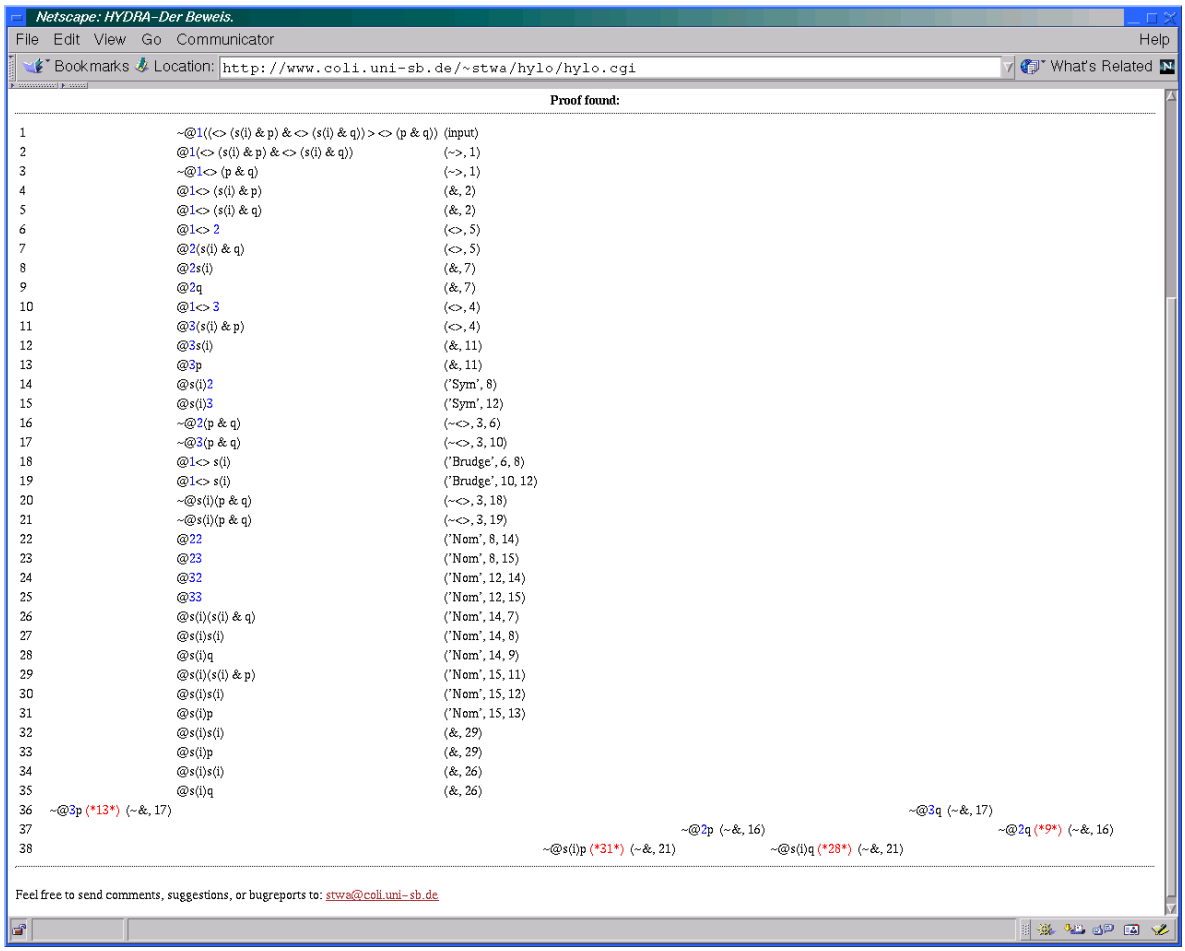
It is hoped to reimplement Hydra, extending its coverage and improving its performance: it is clear that standard ideas from modal and description logic proving could be applied to improve its efficiency. But because Hydra has to deal with *hybrid* logic, rather than ordinary modal logic, there is also a novel efficiency problem, and it is less clear how to cope with this. The inefficiency is not one of implementation: rather it concerns the design of a computationally efficient tableau system for hybrid logic.

Compare, for example, the (not quite complete) screen shot on the following page of Hydra's proof of $\Diamond(i \wedge p) \wedge \Diamond(i \wedge p) \rightarrow \Diamond(p \wedge q)$. The (handwritten) tableau proof given earlier is 18 lines long; Hydra's is 38. Why? Because Hydra applies all the genuine *tableau* rules it can, and then blindly starts applying the *rewrite* rules (note that 14 lines of the proof are taken up by applications of Sym, Nom and Bridge, most of which lead nowhere).

This is a real problem. Any complete system for hybrid logic has to be able to reason about state equality and succession. The question is, how can this part of the reasoning best be controlled? The tableau + rewrite system on which Hydra is based is conceptually clear, but as this example shows, it can lead to unnecessary work.

An interesting alternative would be to experiment with a Tzakova-style tableau calculus (see [5]). Rather than having an explicit 'state equality and succession' component, such systems make use of more complex branch closure rules. This strategy might be easier to

control computationally, perhaps leading to smaller tableau. Another possibility would be to restate the state equality and succession as a constraint task, and turn this component over to a dedicated constraint solver. One way or another, this problem has to be addressed, for it lies right at the heart of genuinely hybrid reasoning.



References

[1] P. Blackburn. Internalizing labelled deduction. *Journal of Logic and Computation*, 10:136–168, 2000.

[2] P. Blackburn. Representation, reasoning, and relational structures: a Hybrid Logic manifesto. In C. Areces, E. Franconi, R. Goré, M. de Rijke, and H. Schlingloff, editors, *Methods for Modalities, 1*, volume 8(3), pages 339–365. Logic Journal of the IGPL, 2000.

[3] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001.

[4] M. Fitting. *First Order Logic and Automated Theorem Proving (second edition)*. Springer Verlag, 1996.

[5] M. Tzakova. Tableaux calculi for hybrid logics. In N. Murray, editor, *Proceedings of the Conference on Tableaux Calculi and Related Methods (TABLEAUX), Saratoga Springs, USA*, volume 1617 of *LNAI*, pages 278–292. Springer Verlag, 1999.