HyLoRes: A Hybrid Logic Prover Based on Direct Resolution

Carlos Areces Juan Heguiabehere
Faculty of Science. University of Amsterdam
{carlos, juanh}@science.uva.nl

Abstract: In recent years, an important number of theoretical results concerning axiomatizability, proof systems (tableaux, natural deduction, etc.), interpolation, expressive power, complexity, etc. for hybrid logics has been obtained. The next natural step is to develop provers that can handle these languages. HyLoRes is a direct resolution prover for hybrid logics implementing a sound and complete algorithm for satisfiability of sentences in $\mathcal{H}(@,\downarrow)$. The most interesting distinguishing feature of HyLoRes is that it is not based on tableau algorithms but on (direct) resolution.

1 Hybrid logics and HyLoRes

Hybrid languages are modal languages that allow direct reference to the elements of a model. The basic hybrid language ($\mathcal{H}(@)$) extends the basic modal language simply by the addition of a new set of atomic symbols called *nominals* (usually denoted as $i, j, k, \ldots$) which name particular points in the model (i.e., the interpretation of a nominal i in a model $\mathcal{M} = \langle W, R, V\rangle$ is an element $i^{\mathcal{M}} \in W$), and for each nominal i a *satisfiability operator* $@_i$. This extension already increases the expressive power of the language as we can now explicitly check whether the point of evaluation w is the specific point named i in the model $\mathcal{M}$:

$$\mathcal{M}, w \Vdash i \text{ iff } w = i^{\mathcal{M}}.$$

And from any point in the model we can check whether a point named i satisfies a given formula φ:

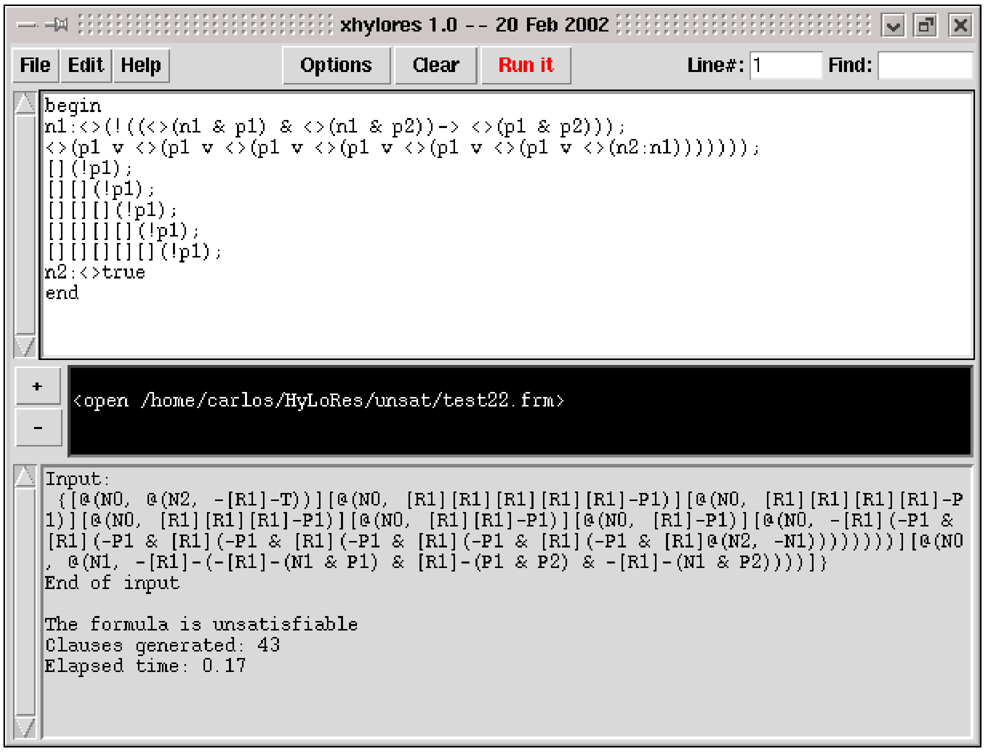
$$\mathcal{M}, w \Vdash @_i\varphi \text{ iff } \mathcal{M}, i^{\mathcal{M}} \Vdash \varphi.$$

The extended expressivity allows one to define elegant decision algorithms, where nominals and @ play the role of labels, or prefixes, which are usually needed during the construction of proofs in the modal setup [10, 5]. Note that they do so *inside* the object language. All these features we get with no increase in complexity: the complexity of the satisfiability problem for $\mathcal{H}(@)$ is the same as for the basic modal language, PSPACE [3].

When we move to very expressive hybrid languages containing binders, we obtain an impressive boost in expressivity, but usually we also move beyond the boundaries of decidability. Classical binders like $\forall$ and $\exists$ (together with @) make the language as expressive as first-order logic (FOL) while the language $\mathcal{H}(@,\downarrow)$ which includes the more "modal" binder $\downarrow$ gives a logic weaker than FOL [2] (but still undecidable). See the Hybrid Logic site at `http://www.hylo.net` for a broad on-line bibliography.

In recent years, an important number of theoretical results concerning axiomatizability, proof systems (tableaux, natural deduction, etc), interpolation, expressive power, complexity, etc. for hybrid logics has been obtained. The next natural step is to develop provers that can handle these languages. HyLoRes is a direct resolution prover for hybrid logics implementing a sound and complete algorithm for satisfiability of sentences in $\mathcal{H}(@,\downarrow)$; it implements the algorithm presented in [5]. The most interesting distinguishing feature of HyLoRes is that it is not based on tableau algorithms but on (direct) resolution. HyLoRes implements a version of the "given clause" algorithm, which has become the skeleton underlying most first-order provers. In contrast to translation based provers like MSPASS [17], HyLoRes performs resolution directly on the modal (or hybrid) input, with no translation into background logics.

It is often said that hybrid logics combine interesting features from both modal and first-order logics. In the same spirit, HyLoRes fuses ideas from state-of-the-art first-order proving with the simple representation of the hybrid object language.



HyLoRes and the Tcl/Tk interface xHyLoRes shown in the picture above are available for downloading at `http://www.illc.uva.nl/~carlos/HyLoRes`, where you can also use HyLoRes on-line.

2 Direct resolution for hybrid logics

Designing resolution methods that can directly (without translation into large background languages) be applied to modal logics, received some attention in the late 1980s and early 1990s. But even though modal languages are sometimes viewed as "simple extensions of propositional logic," direct resolution for modal languages has proved a difficult task. Intuitively, in basic modal languages the resolution rule has to operate *inside* boxes and diamonds to achieve completeness. This leads to more complex systems, less elegant results, and poorer performance, ruining the "one-dumb-rule" spirit of resolution. In [5] we presented a resolution calculus that uses the hybrid machinery to "push formulas out of modalities" and in this way, feed them into a simple and standard resolution rule.

To handle the hybrid operators, we need just notice that nominals and @ introduce a limited form of equational reasoning: a formula like $@_i j$ is true in a model iff i and j are nominals for the *same* state. A paramodulation rule similar to the one used by Robinson and Wos [22] lets us handle nominals and @.

Briefly, our resolution algorithm works as follows. Define the following rewriting procedure *nf* on formulas of $\mathcal{H}(@,\downarrow)$. Let φ be a formula in $\mathcal{H}(@,\downarrow)$, $nf(\varphi)$ is obtained by repeated application of the following rewrite rules until none is applicable.

$$\begin{array}{rcl}
\varphi[\![\psi \vee \theta]\!] & \overset{nf}{\leadsto} & \varphi[\![\neg(\neg\psi \wedge \neg\theta)]\!] \\
\varphi[\![\langle R\rangle\psi]\!] & \overset{nf}{\leadsto} & \varphi[\![\neg[R]\neg\psi]\!] \\
\varphi[\![\neg\neg\psi]\!] & \overset{nf}{\leadsto} & \varphi[\![\psi]\!] \\
\varphi[\![\neg @_t\psi]\!] & \overset{nf}{\leadsto} & \varphi[\![@_t\neg\psi]\!] \\
\varphi[\![\neg{\downarrow}x.\psi]\!] & \overset{nf}{\leadsto} & \varphi[\![{\downarrow}x.\neg\psi]\!]
\end{array}$$

Clauses are sets of formulas of this form. To determine the satisfiability of a formula $\varphi \in \mathcal{H}(@,\downarrow)$ we first notice that φ is satisfiable iff $@_t\varphi$ is satisfiable, for a nominal t not appearing in φ. Define the clause set *ClSet* corresponding to φ to be $ClSet(\varphi) = \{\{@_t nf(\varphi)\}\}$. Next, let $ClSet^*(\varphi)$ – the saturated clause set corresponding to φ – be the smallest set containing $ClSet(\varphi)$ and closed under the rules in Figure 1.

$$(\wedge)\quad \frac{Cl \cup \{@_t(\varphi_1 \wedge \varphi_2)\}}{\begin{array}{c}Cl \cup \{@_t\varphi_1\}\\ Cl \cup \{@_t\varphi_2\}\end{array}} \qquad (\vee)\quad \frac{Cl \cup \{@_t\neg(\varphi_1 \wedge \varphi_2)\}}{Cl \cup \{@_t nf(\neg\varphi_1), @_t nf(\neg\varphi_2)\}}$$

$$(\mathrm{RES})\quad \frac{Cl_1 \cup \{@_t\varphi\} \quad Cl_2 \cup \{@_t\neg\varphi\}}{Cl_1 \cup Cl_2}$$

$$([R])\quad \frac{Cl_1 \cup \{@_t[R]\varphi\} \quad Cl_2 \cup \{@_t\neg[R]\neg s\}}{Cl_1 \cup Cl_2 \cup \{@_s\varphi\}} \qquad (\langle R\rangle)\quad \frac{Cl \cup \{@_t\neg[R]\varphi\}}{\begin{array}{c}Cl \cup \{@_t\neg[R]\neg n\}\\ Cl \cup \{@_n nf(\neg\varphi)\}\end{array}}, \text{ for } n \text{ new.}$$

$$(@)\quad \frac{Cl \cup \{@_t @_s\varphi\}}{Cl \cup \{@_s\varphi\}}$$

$$(\mathrm{SYM})\quad \frac{Cl \cup \{@_t s\}}{Cl \cup \{@_s t\}} \qquad (\mathrm{REF})\quad \frac{Cl \cup \{@_t\neg t\}}{Cl} \qquad (\mathrm{PARAM})\quad \frac{Cl_1 \cup \{@_t s\} \quad Cl_2 \cup \{\varphi(t)\}}{Cl_1 \cup Cl_2 \cup \{\varphi(t/s)\}}$$

Figure 1: Resolution Rules

The computation of $ClSet^*(\varphi)$ is in itself a sound and complete algorithm for checking satisfiability of $\mathcal{H}(@)$, in the sense that φ is unsatisfiable if and only if the empty clause $\{\}$ is a member of $ClSet^*(\varphi)$ (see [5]).

The hybrid binder $\downarrow$ binds variables to the point of evaluation, i.e., for a model $\mathcal{M}$, an assignment g and a state w,

$$\mathcal{M}, g, w \models \downarrow x.\varphi \text{ iff } \mathcal{M}, g^x_w, w \models \varphi,$$

where g^x_w is the assignment that coincides with g, but maps x to w. For example, a state w satisfies the formula $\downarrow x.\Diamond x$ if and only if w can reach itself through the accessibility relation. Extending the system to account for hybrid sentences using $\downarrow$ is fairly straightforward. Consider the rule $(\downarrow)$ below

$$(\downarrow) \quad \frac{Cl \cup \{@_t \downarrow x.\varphi\}}{Cl \cup \{@_t \varphi(x/t)\}}.$$

As $\downarrow$ is self dual (i.e., $\neg\downarrow x.\varphi$ is equivalent to $\downarrow x.\neg\varphi$) we don't need a rule for its negation (the same is true for @). Notice also that the rule transforms hybrid sentences into hybrid sentences. The full set of rules is a sound and complete calculus for checking satisfiability of sentences in $\mathcal{H}(@,\downarrow)$.

Example. We prove that $\downarrow x.\langle R\rangle(x \wedge p) \to p$ is a tautology. Consider the clause set corresponding to the negation of the formula:

1. $\{@_i((\downarrow x.\neg[R]\neg(x \wedge p))\underline{\wedge}\neg p)\}$ by $(\wedge)$
2. $\{@_i\underline{\downarrow x}.\neg[R]\neg(x \wedge p)\}, \{@_i\neg p\}$ by $(\downarrow)$
3. $\{@_i\underline{\neg[R]}\neg(i \wedge p)\}, \{@_i\neg p\}$ by $(\langle R\rangle)$
4. $\{@_i\neg[R]\neg j\}, \{@_j(i\underline{\wedge}p)\}, \{i{:}\neg p\}$ by $(\wedge)$
5. $\{@_j\underline{i}\}, \{@_j\underline{p}\}, \{@_i\neg p\}$ by (PARAM)
6. $\{@_i\underline{p}\}, \{@_i\underline{\neg p}\}$ by (RES)
7. $\{\}$.

3 The "given clause" algorithm for hybrid resolution

HyLoRes implements a version of the "given clause" algorithm [24] shown in Figure 2. The implementation preserves the soundness and completeness of the calculus introduced in Section 2, and ensures termination for $\mathcal{H}(@)$. As an example of the execution of the prover, we show how HyLoRes solves the formula in the previous example:

Example The following are dumps of the input formula and the execution of the prover (minimally formatted for presentation).

Input file:

```
begin
!((down (x1 dia (x1 & p1) )) -> p1)
end
```

Execution:

```
(carlos@guave 149) hylores -f test.frm -r
Input:
   {[@(N0, (-P1 & Down(X1, -[R1]-(P1 & X1))))]}
End of input

Given: (1, [@(N0, (-P1 & Down(X1, -[R1]-(P1 & X1))))])
```

```
CON: {[@(N0, -P1)][@(N0, Down(X1, -[R1]-(P1 & X1)))]}
Given: (2, [@(N0, -P1)])
Given: (3, [@(N0, Down(X1, -[R1]-(P1 & X1)))])
ARR: {[@(N0, -[R1]-(P1 & N0))]}
Given: (4, [@(N0, -[R1]-(P1 & N0))])
DIA: {[@(N-2, (P1 & N0))][@(N0, -[R1]-N-2)]}
Given: (5, [@(N-2, (P1 & N0))])
CON: {[@(N-2, P1)][@(N-2, N0)]}
Given: (6, [@(N-2, N0)])
PAR (0,-2): {[@(N-2, (P1 & N-2))][@(N-2, -[R1]-(P1 & N-2))]
[@(N-2, Down(X1, -[R1]-(P1 & X1)))][@(N-2, -P1)]
[@(N-2, (-P1 & Down(X1, -[R1]-(P1 & X1))))]}
Given: (7, [@(N-2, P1)])
Given: (8, [@(N-2, -P1)])
RES: (7, [])
```

We discuss now some of the salient characteristics of the prover.

Programing language. HyLoRes is implemented in Haskell, and compiled with the Glasgow Haskell Compiler (GHC) Version 5.02. GHC generates fairly efficient C code which is afterwards compiled into an executable file. Thus, users need no additional software to use the prover.

The HyLoRes site provides executables for Solaris (tested under Solaris 8) and Linux (tested under Red Hat 7.0 and Mandrake 8.0). The original Haskell code is also made publicly available under the GPL license [12] (the code, though, is still unstable, being under active development). We will soon provide also the intermediate C source which could then be compiled under a wider range of platforms.

In addition to HyLoRes, a graphical interface called xHyLoRes implemented in Tcl/Tk was developed. It uses HyLoRes in the background and provides full file access and editing capabilities, and a more intuitive control of the command line parameters of the prover.

Data structures. The design of the algorithm is modular with respect to the internal representation of the different kinds of data. We have used the Edison package (a library of efficient data types provided with GHC) to implement most of the data types representing sets. But while we represent clauses directly as `UnbalancedSet`, we have chosen different representations for each of the clause sets used by the algorithm: *new* and *inuse* are simply lists of clauses (as they always have to be examined sequentially, one by one) and *clauses* is an UnbalancedSet of clauses.[1] In particular, *clauses* is ordered by our selection criterion, which makes for an efficient selection of the given clause. The selection order can be specified from the command line. Four complexity measures are computed for each clause C: v the maximum number of propositional variables in a formula in C, d the maximum modal depth of a formula in C, p the minimum prefix of a formula in C and s the size of the C (number of formulas in C). Different combinations of this measures can be chosen as order for *clauses*. Selection of different orders can have an important effect on the performance of the prover (see Figure 3 *b)*).

The internal state of the given clause algorithm (the sets *clauses*, *inuse* and *new*, the data structures used for subsumption checking, etc) is represented as a combination

[1]While `List` provides efficient sequential access to their elements, `UnbalancedSet` implements sets as unbalanced search trees to optimize search of single elements.

input: init: set of clauses
var: new, clauses, inuse: set of clauses
var: given: clause

clauses := $\{\}$; inuse := $\{\}$; new := init
new := normalize(new)
if $\{\} \in$ new **then return** "unsatisfiable"
clauses := computeComplexity(new)
while clauses $\neq \{\}$ **do**
    $\{$* *Selection of given clause* *$\}$
    given := select(clauses)
    clauses := clauses – $\{$given$\}$
    $\{$* *Inference* *$\}$
    new := infer(given, inuse)
    new := normalize(new)
    if $\{\} \in$ new **then return** "unsatisfiable"
    $\{$* *Subsumption deletion* *$\}$
    new := simplify(new, inuse $\cup$ clauses)
    inuse := simplify(inuse, new)
    clauses := simplify(clauses, new)
    $\{$* *Initialization for next cycle* *$\}$
    if notRedundant(given) **then**
        inuse := inuse $\cup$ $\{$given$\}$
    clauses := clauses $\cup$ computeComplexity(new)
return "satisfiable"

- *normalize(A)* applies *nf* to formulas in *A* and handles trivial tautologies/contradictions.
- *computeComplexity(A)* determines length, modal depth, number of literals, etc. for each of the formulas in *A*; these values are used by *select* to pick the given clause.
- *infer(*given*,A)* applies the resolution rules to the given clause and each clause in *A*. If the rules $(\wedge)$, $(\vee)$, $(\langle R\rangle)$ or $(\downarrow)$ are applicable, no other rule is applied as the clauses obtained as conclusions by their application subsume the premises.
- *simplify(A,B)* performs subsumption deletion, returning the subset of *A* which is not subsumed by any element in *B*.
- *notRedundant(*given*)* is true if none of the rules $(\wedge)$, $(\vee)$, $(\langle R\rangle)$ or $(\downarrow)$ was applied to given.

Figure 2: Given clause algorithm implemented in HyLoRes

of a state and an output monads (see [25]); the former provides transparent access to the internal state of the algorithm from the monadic functions that perform inference, while the latter handles all printing services with no need of further parameters in the function signatures. In addition, the use of monads allows the addition of further structure (hashing functions, etc.) to optimize search, with minimum re-coding. We have already experienced the advantages of the monad architecture as we have been able to check different data structures and improve the performance of some of the most expensive functions with great ease.

Extensibility. With respect to the addition of further resolution rules, our main aim was to take advantage of the inherent modularity of the given clause algorithm. New rules can simply be added in the *infer* function without the need for any further modification of the code. One of the main extensions we are planning for version 2.0 of the prover is the addition of the universal modality A [13]. The logic $\mathcal{H}(\mathsf{A},\downarrow)$ is as expressive as FOL and would turn HyLoRes into a full first-order prover. But interestingly, the language is now split in a radically new way: a decidable part ($\mathcal{H}(\mathsf{A})$ with an EXPTIME-complete satisfiability problem, see [3]) and an undecidable, but intrinsically local part ($\mathcal{H}(@,\downarrow)$ is equivalent in expressivity to the fragment of FOL invariant under generated submodels, see [4]).

Subsumption Checking. Subsumption checking (i.e., deciding whether a clause in the clause set is redundant and can be deleted) is one of the – or perhaps "the" – most expensive operations in resolution based theorem provers [23]. HyLoRes uses a simple version of subsumption checking where a clause C_1 subsumes a clause C_2 if $C_1 \subset C_2$. Early versions of the prover implemented this test very inefficiently, checking the subset relation element by element, and clause by clause. Since Version 0.9 a set-at-a-time subsumption checking algorithm which uses a clause repository structured as a trie [23] is implemented, with notorious improvements (see Figure 3 *a)*).

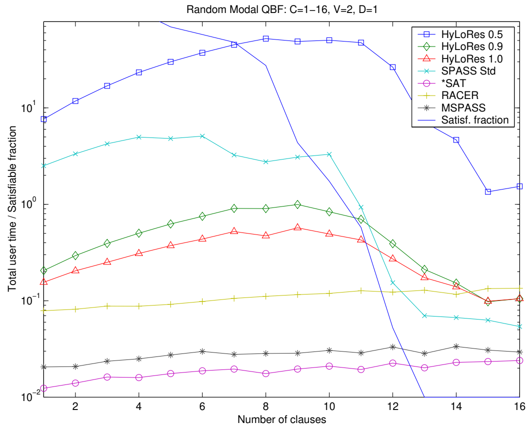
Paramodulation. As we said in Section 2, we need some kind of paramodulation to handle nominals and @. We can once more take advantage from FOL experience in resolution based theorem proving here. In [7], Bachmair and Ganzinger develop in detail the modern theory of equational reasoning for first-order saturation based provers. Many of the ideas and optimizations discussed there can and should be implemented in HyLoRes. In the current version, paramodulation is done naively. The only "optimization" being the orientation of equalities so that we always replace nominals by nominals which are lower in a certain ordering.

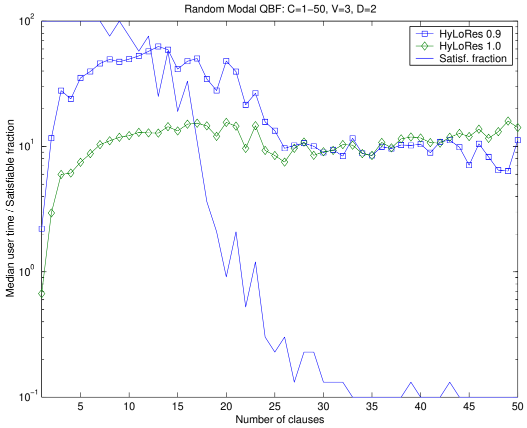
4 Comparison and Testing

The prototype is not yet meant to be competitive when compared with state of the art theorem provers for modal and description logics like MSPASS [17], DLP [20], FaCT [16], RACER [14] or *SAT [11]. On the one hand, the system is still in a preliminary stage of development (only very simple optimizations for hybrid logics have been implemented), and on the other hand hybrid and description languages are related but different. $\mathcal{H}(@,\downarrow)$ is undecidable while the implemented description languages are mostly decidable. And even when comparing the fragment $\mathcal{H}(@)$ for which HyLoRes implements a decision algorithm, the expressive powers are incomparable ($\mathcal{H}(@)$ permits free Boolean combinations of @ and nominals but lacks the limited form of universal modality available in the T-Box of DL provers [2]).

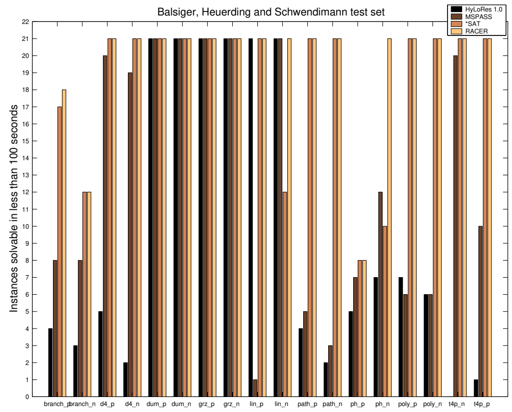
Figure 3 shows ongoing work on preliminary testing with the random modal QBF generator (plots *a)* and *b)*), the hand-tailored set of Balsiger et al. (plots *c)* and *d)*), and the random modal CNF generator (plots *e)* and *f)*).

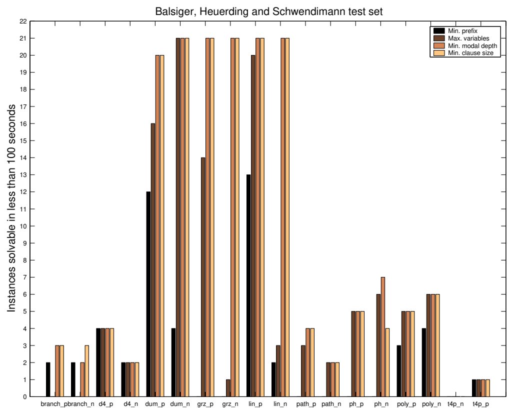
The first two plots use the random modal QBF generator described in [19]. This generator produces random formulas by first generating a random quantified Boolean



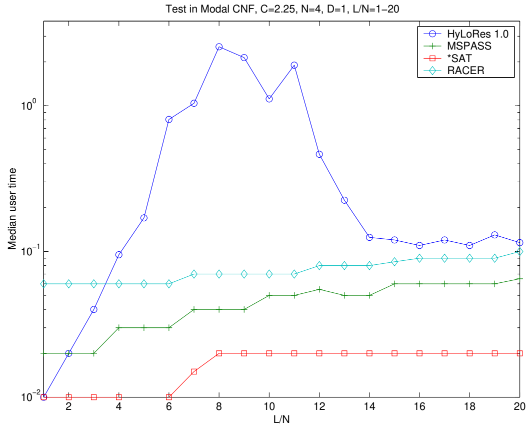


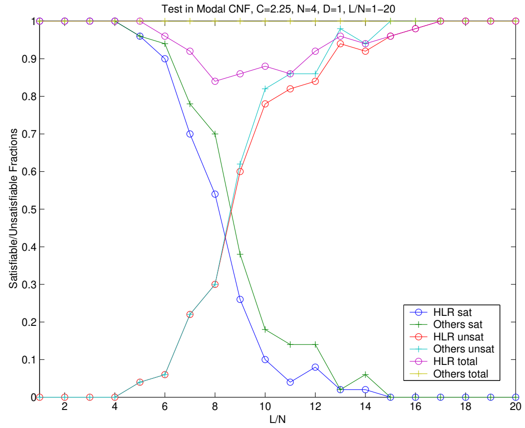
a) Comparison with different provers - Modal QBF *b) Comparison with different versions - Modal QBF.*





c) Comparison with different provers - BHS *d) Comparison with different orderings - BHS*





e) & f) Comparison with different provers - Modal CNF.

Figure 3: Testing HyLoRes on random and hand-tailored formulas

formula and then using different satisfaction preserving encodings into the modal language. The random modal QBF generator has been previously used for system comparison, but see [15] for its limitations. Different parameters allow the generation of formulas in different areas of the input space (the parameters used are specified at the top of each plot). The x-axis varies with the number of clauses generated (from under-constrained, mostly satisfiable problems to the left to over-constrained, mostly unsatisfiable problems to the right). The y-axis shows the median CPU-time in a set of 25 instances for each setting. The unmarked line shows the relative frequency of satisfiable formulas (moving from 1 in the left to 0 in the right).

In plot *a)* we investigate very simple modal problems and we compare HyLoRes with *SAT [11] (a modal prover that interleaves modal steps with efficient propositional reasoning), RACER [14] (one of the most developed description logic provers), SPASS [1] (one of the best first-order provers based on resolution) and MSPASS [17] (which implements an optimized translation of modal formulas into first-order logic, and then calls SPASS on this output with fine tuned configuration options). The performance of three different versions of HyLoRes is shown. HyLoRes 0.5 uses the clause-at-a-time subsumption checking strategy. HyLoRes 0.9 already implements a clause-at-a-time subsumption checking strategy. HyLoRes 1.0 simply improves the performance of some of the most expensive functions by fine-tuning the Haskell code. It is interesting to notice how the performance of HyLoRes improves when the problem is over-constrained (as expected in a resolution based theorem prover).

In plot *b)* we compare HyLoRes 0.9 and HyLoRes 1.0 on harder instances of the random modal QBF test, observing a similar behavior as in plot *a)*. The differences between HyLoRes 0.9 and HyLoRes 1.0 peak on satisfiable problems, where the prover needs to perform a full exploration of the clause set to reach saturation.

Plots *c)* and *d)* use the hand-tailored test set defined in [9]. This test set consists of 9 different kinds of problems, each kind itself split into satisfiable and unsatisfiable test cases (the different test sets are listed on the horizontal axis, the postfix _n identify satisfiable problems, while _p identify unsatisfiable problems). Finally, each of these groups of problems contains 21 instances, each one designed to be twice as difficult as the previous one. The test is run with a timeout of a 100 seconds per problem, plotting the number of problems solved in each class.

In plot *c)* we compare HyLoRes with RACER, *SAT and MSPASS. Here we can clearly see that DL provers like RACER have still a huge lead. RACER is able to solve almost all problems within the timeout of 100 seconds per problem. HyLoRes compares more favorably with MSPASS, with MSPASS surpassing HyLoRes in most test sets except in lin_p.

In plot *d)* we run HyLoRes alone, using different orderings for the selection of the given clause. We see that resolution can be quite sensitive to the order used, both on satisfiable and unsatisfiable instances (see the columns for dum_p, dum_n, grz_p and grz_n). We only plot "simple" orderings where we sort *clauses* by the minimum prefix, the maximum number of variables, the minimum modal depth or the minimum clause size. Apparently, setting the orderings to minimum modal depth and minimum clause size provides best performance. HyLoRes allows any combination of these measures, and in some cases the combination of two of them provides better results than the ones obtained using each of them separately.

Finally, plots *e)* and *f)* use the random modal CNF generator introduced in [21]. The random modal CNF directly generates sets of modal clauses (no translation is involved), and its many parameters permit the full exploration of the complete test space. Furthermore, its flexibility allows precise control of the overall problem difficulty.

The plots show a run of HyLoRes, MSPASS, *SAT and RACER. In *e)* the y-axis shows once more median CPU-time (50 instances per datapoint, timeout of 30 seconds), while the x-axis shows the ratio of number of clauses (L) and number of variables (N). In *f)* we show the ratio of satisfiable and unsatisfiable formulas for each datapoint and the sum these two ratios. The points where the ratios do not add to one show timeouts. The problems generated are very easy for the other provers, but they already cause some timeouts for HyLoRes, specially in the hard section of maximal uncertainty.

To close this section, one thing to bear in mind is that the testings we have been able to perform up to now are not really representative of the capabilities of HyLoRes, as they are purely modal and do not require hybrid reasoning. No hybrid test collection or random generator of hybrid formulas is available. We plan to develop such a resource (by extending the random modal CNF to encompase nominal, satisfiability operators and binders) in the near future.

5 Conclusions and Future Work

There remain many things to try and improve in HyLoRes, but the main goal we pursued while developing Version 1.0 of the prover has been largely achieved: direct resolution can be used as an interesting, and in the future perhaps even competitive, alternative to tableaux based methods for modal and hybrid logics. Among the things that will be improved in the next versions of HyLoRes are the following.

We are investigating both the theoretical and practical issues involved in performing direct *ordered* resolution for hybrid logics, where the resolution rules are restricted to the maximum literals in the clauses [8].

We want to make the prover much more aware of the characteristics of its input. At the moment, the prover simply check which formulas appear in the input (propositional, modal, basic hybrid, and binders) and uses the appropriate rules of the calculus. Particular rules/heuristics for certain inputs (e.g., SLD resolution and horn clauses) can provide important improvements in performance. Some of the heuristics presented in [6] can be adapted for our calculus. Also, more effective normal form transformations should be applied to the input before proceeding to resolution. An example of this: formulas prefixed by @ are global and can be pushed outside modalities. This transformation can greatly affect the performance of paramodulation.

Another important extension for future versions is to extend the language with the universal modality A. As we said above, this would give us full FOL expressivity. Also the language $\mathcal{H}(\mathsf{A})$ is interesting as it let us perform inference in terms of full Boolean knowledge bases of the description logic $\mathcal{ALC}$ in HyLoRes (see [2]).

Finally, we would like to improve the output of the prover, making in able to display a concise refutation proof in case it finds one, or a model satisfying the input otherwise.

As we said in the introduction, HyLoRes fuses nicely some ideas from state-of-the-art first-order proving with the simplicity of hybrid languages; and it provides the basis for future developments on computational tools for hybrid logic. Already in its actual state, users find the tool useful for better understanding the formalisms.

Acknowledgment. C. Areces is supported by the NWO project # 612.069.006.

References

[1] B. Afshordel, U. Brahm, C. Cohrs, T. Engel, E. Keen, C. Theobalt, D. Topić, and C. Weidenbach. System description: SPASS Version 1.0.0. In *Automated deduction—CADE-16 (Trento, 1999)*, pages 187–201. Springer, Berlin, 1999.

[2] C. Areces. *Logic Engineering. The Case of Description and Hybrid Logics*. PhD thesis, Institute for Logic, Language and Computation, University of Amsterdam, Amsterdam, The Netherlands, October 2000.

[3] C. Areces, P. Blackburn, and M. Marx. A road-map on complexity for hybrid logics. In J. Flum and M. Rodríguez-Artalejo, editors, *Computer Science Logic*, number 1683 in LNCS, pages 307–321. Springer, 1999. Proceedings of the 8th Annual Conference of the EACSL, Madrid, September 1999.

[4] C. Areces, P. Blackburn, and M. Marx. Hybrid logics: characterization, interpolation and complexity. *The Journal of Symbolic Logic*, 66(3):977–1010, 2001.

[5] C. Areces, H. de Nivelle, and M. de Rijke. Resolution in modal, description and hybrid logic. *Journal of Logic and Computation*, 11(5):717–736, 2001.

[6] Y. Auffray, P. Enjalbert, and J. Hebrard. Strategies for modal resolution: results and problems. *Journal of Automated Reasoning*, 6(1):1–38, 1990.

[7] L. Bachmair and H. Ganzinger. Equational reasoning in saturation-based theorem proving. In *Automated deduction—a basis for applications, Vol. I*, pages 353–397. Kluwer Acad. Publ., Dordrecht, 1998.

[8] L. Bachmair and H. Ganzinger. Resolution theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 2, pages 19–99. Elsevier Science, 2001.

[9] 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.

[10] P. Blackburn. Internalizing labelled deduction. *Journal of Logic and Computation*, 10(1):137–168, 2000.

[11] F. Giunchiglia and R. Sebastiani. A sat-based decision procedure for $\mathcal{ALC}$. In L. Aiello, J. Doyle, and S. Shapiro, editors, *Proceedings of the 5th International Conference on Principles of Knowledge Representation and Reasoning (KR'96)*, pages 304–314, Cambridge, USA, 1996.

[12] GNU General Public License. `http://www.gnu.org/copyleft/gpl.html`.

[13] V. Goranko and S. Passy. Using the universal modality: gains and questions. *Journal of Logic and Computation*, 2:5–30, 1992.

[14] V. Haarslev and R. Möller. RACE system description. In Lambrix et al. [18], pages 130–132.

[15] J. Heguiabehere and M. de Rijke. The random modal QBF test set. In *Proceedings IJCAR Workshop on Issues in the Design and Experimental Evaluation of Systems for Modal and Temporal Logics*, pages 58–67, 2001. refereed.

[16] I. Horrocks. FaCT and iFaCT. In Lambrix et al. [18], pages 133–135. FACT is available under the GNU public license at `http://www.cs.man.ac.uk/~horrocks`.

[17] U. Hustadt, R. A. Schmidt, and C. Weidenbach. MSPASS: Subsumption testing with SPASS. In Lambrix et al. [18], pages 136–137.

[18] P. Lambrix, A. Borgida, M. Lenzerini, R. Möller, and P. Patel-Schneider, editors. *Proceedings of the 1999 International Workshop on Description Logics (DL'99)*, 1999.

[19] F. Massacci. Design and results of the Tableaux-99 Non-Classical (Modal) Systems Comparison. In N. Murray, editor, *Proceedings of the Third International Conference on Analytic Tableaux and Related Methods (TABLEAUX'99)*, volume 1617 of *Lecture Notes in Artificial Intelligence*, pages 14–18. Springer-Verlag, 1999.

[20] P. Patel-Schneider. DLP system description. In E. Franconi, G. De Giacomo, R. MacGregor, W. Nutt, and C. Welty, editors, *Proceedings of the 1998 International Workshop on Description Logics (DL'98)*, pages 87–89, 1998. DLP is available at `http://www.bell-labs.com/user/pfps`.

[21] P. Patel-Schneider and R. Sebastiani. A new very-general method to generate random modal formulae for testing decision procedures. *Journal of Artificial Intelligence Research*, 2002. Submitted.

[22] G. Robinson and L. Wos. Paramodulation and theorem-proving in first-order theories with equality. In *Machine Intelligence, 4*, pages 135–150. American Elsevier, New York, 1969.

[23] A. Voronkov. The anatomy of Vampire. *Journal of Automated Reasoning*, 15(2):237–265, 1995.

[24] A. Voronkov. Algorithms, datastructures, and other issues in efficient automated deduction. In R.Goré, A. Leitsch, and T. Nipkow, editors, *Automated Reasoning. 1st. International Joint Conference, IJCAR 2001*, number 2083 in LNAI, pages 13–28, Siena, Italy, June 2001.

[25] P. Wadler. Monads for functional programming. In J. Jeuring and E. Meijer, editors, *Advanced Functional Programming*, number 925 in LNCS. Springer Verlag, 1995.