HyLoRes: A Hybrid Logic Prover Based on Direct Resolution
(System Description)
Carlos Areces Juan Heguiabehere
Faculty of Science. University of Amsterdam
{carlos, juanh}@science.uva.nl

1 Hybrid logics and HyLoRes

Hybrid logics 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 $M = \langle W, R, V \rangle$ is a an element $i^M \in W$), 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 $M$:

$$M, w \vDash i \iff w = i^M.$$ 

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

$$M, w \vDash \@_i \varphi \iff M, i^M \vDash \varphi.$$ 

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 [11, 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 machin-
ery 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 [21] lets us handle nominals and @.

Very briefly, our resolution algorithm works as follows. First, we consider formulas where \( R \varphi \) and \( \varphi \lor \psi \) are rewritten as \( \neg [R] \neg \varphi \) and \( \neg (\neg \varphi \land \neg \psi) \) respectively, and where double negations have been eliminated. Clauses are sets of formulas of this form. To determine the satisfiability of a formula \( \varphi \in \mathcal{H}(\varphi) \) we first notice that \( \varphi \) is satisfiable iff @,\varphi is satisfiable, for a nominal \( t \) not appearing in \( \varphi \). Define the clause set \( \text{ClSet} \) corresponding to \( \varphi \) to be \( \text{ClSet} (\varphi) = \{ \{ \varphi, \neg (\varphi) \} \} \), where \( t \) does not appear in \( \varphi \), and \( \neg \varphi \) is \( \psi \) if \( \varphi \) is equal to \( \neg \psi \) and \( \neg \psi \) otherwise. Next, let \( \text{ClSet}^t (\varphi) \) – the saturated clause set corresponding to \( \varphi \) – be the smallest set containing \( \text{ClSet} (\varphi) \) and closed under the following rules.

\[
\begin{array}{cc}
\text{(L)} & \frac{\text{Cl} \cup \{ @1(\varphi) \land \varphi_2 \} }{\text{Cl} \cup \{ @1, \varphi_1 \} } \\
\text{(V)} & \frac{\text{Cl} \cup \{ @2, (\varphi_1 \land \varphi_2) \} }{\text{Cl} \cup \{ @2, \neg \varphi_1, \neg \varphi_2 \} } \\
\text{(RES)} & \frac{\text{Cl}_1 \cup \{ @1, \varphi \} \quad \text{Cl}_2 \cup \{ @1, \neg \varphi \} }{\text{Cl}_1 \cup \text{Cl}_2} \\
\text{([R])} & \frac{\text{Cl}_1 \cup \{ @1, [R] \varphi \} \quad \text{Cl}_2 \cup \{ @1, \neg [R] - \varphi \} }{\text{Cl}_1 \cup \text{Cl}_2 \cup \{ @2, \varphi \} } \\
\text{([R])} & \frac{\text{Cl} \cup \{ @1, [R] \varphi \} }{\text{Cl} \cup \{ @1, \neg [R] - n \varphi \} } \\
\text{(SYM)} & \frac{\text{Cl} \cup \{ @2, x \varphi \} }{\text{Cl} \cup \{ @2, x \varphi \} } \\
\text{(REF)} & \frac{\text{Cl} \cup \{ @1, \neg t \varphi \} }{\text{Cl}} \\
\text{(PARAM)} & \frac{\text{Cl}_1 \cup \{ @2, \varphi \} \quad \text{Cl}_2 \cup \{ \varphi, t \} }{\text{Cl}_1 \cup \text{Cl}_2 \cup \{ \varphi, t/s \} } \\
\end{array}
\]

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

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

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

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

\[
\Downarrow \frac{\text{Cl} \cup \{ @1, \downarrow x. \varphi \} }{\text{Cl} \cup \{ @2, \varphi, \varphi (x/t) \} }.
\]

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 $H(\forall, \downarrow)$.

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

1. $\{ \Rightarrow_i([-x.\neg[R]-(x \land p)\land \neg p)] \}$ by (\land)
2. $\{ \Rightarrow_i[-[R]-(x \land p)], \{ \Rightarrow_i\neg p \} \}$ by ($\downarrow$)
3. $\{ \Rightarrow_i[-[R]-(i \land p)], \{ \Rightarrow_i\neg p \} \}$ by ($\neg[\Rightarrow]$)
4. $\{ \Rightarrow_i[-[R]\land i], \{ \Rightarrow_i(i \land p) \}, \{ i \land \neg p \} \}$ by (\land)
5. $\{ \Rightarrow_i[i], \{ \Rightarrow_i p \}, \{ \Rightarrow_i \neg p \} \}$ by (PARAM)
6. $\{ \Rightarrow_i p \}, \{ \Rightarrow_i \neg p \}$ by (RES)
7. $\{ \}$.

3 The “given clause” algorithm for hybrid resolution

HyLoRes implements a version of the “given clause” algorithm [23] shown in Figure 1. The implementation preserves the soundness and completeness of the calculus introduced in Section 2, and ensures termination for $H(\forall)$.  

input: init: set of clauses
var: new, clauses, inuse, inactive: set of clauses
var: given: clause
clauses := $\{ \}$
new := init
simplify(&new, inuse & inactive & clauses)
if $\{ \} \in$ new then return “unsatisfiable”
clauses := computeComplexity(new)
while clauses $\neq \{ \}$ do
  given := select(clauses)
  clauses := clauses – $\{ \text{given} \}$
  while subsumed(given, inuse) do
    if clauses $= \{ \}$
      then return “satisfiable”
    else
      given := select(clauses)
      clauses := clauses – $\{ \text{given} \}$
      simplify(&inuse, given)
      new := infer(inuse, given, &inactive)
      simplify(&new, inuse & inactive & clauses)
      if $\{ \} \in$ new then return “unsatisfiable”
      clauses := clauses & computeComplexity(new)
simplify performs subsumption deletion (& marks the modified set). computeComplexity determines length, modal depth, number of literals, etc. for each of the formulas; these values are used by select to pick the given clause. infer applies the resolution rules to the given clause and each clause in inuse; if the $\land$, $\lor$, $\Rightarrow$ or $\downarrow$-rules are applied, the given clause is added to inactive so that it is not generated again, and not to inuse as it would be redundant.

Figure 1: Given clause algorithm implemented in HyLoRes

As an example of the execution of the prover, we show how HyLoRes solves the formula in the previous 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]}
Given: (7, {[N-2, p1]})
Given: (8, {[N-2, -p1]})
RES: (7, [])
```

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

**Programming language.** HyLoRes is implemented in Haskell, and compiled with the Glasgow Haskell Compiler (GHC) Version 5.02, generating executable code which increases its usability. GHC generates fairly efficient C code which is afterward compiled into the executable. 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 [13] (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), clauses and inactive are UnbalancedSets of clauses.\(^1\) In particular, clauses

---

\(^1\)While List provides efficient sequential access to their elements, UnbalancedSet implements sets as unbalanced search trees to optimize search of single elements.
is ordered by our selection criterion, which makes for an efficient selection of the given clause. The selection orders can be chosen 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 $C$). Any combination of these 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 2 b).

The internal state of the given clause algorithm is represented as a combination of a state and an output monad (see [24]). This 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 $A$ [14]. The logic $\mathcal{H}(A, \downarrow)$ 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 ($\mathcal{H}(A)$ 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 [22]. HyLoRes uses a simple version of subsumption checking where a clause $C_1$ subsumes a clause $C_2$ if $C_1 \subseteq C_2$. Version 1.0 of the prover implemented this test very inefficiently, checking the subset relation element by element, and clause by clause. In the latest prototype, a set-at-a-time subsumption checking algorithm which uses a clause repository structured as a trie [22] was implemented with notorious improvements (see Figure 2 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 [8], 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 provers for modal and description logics like DLP [20], FaCT [16], MSPASS [17], or
RACER [15]. 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 $\mathcal{H}(\uparrow)$ for which HyLoRes implements a decision algorithm, the expressive powers are incomparable ($\mathcal{H}(\uparrow)$ permits free Boolean combinations of $\uparrow$ and nominals but lacks the limited form of universal modality available in the T-Box of DL provers [2]).

Figure 2 shows some ongoing work on basic testing with the random QBF generator [19] (graphics a) and b) and the hand-tailored set of Balsiger et al. [10] (graphics c) and d)).

In the first graphic, we investigate very simple modal problems and we compare HyLoRes with *SAT [12] (a modal prover that interleaves modal steps with calls to an efficient propositional prover), RACER [15] (one of the most developed description logic provers) and SPASS [1] (one of the best first order provers based on resolution). The unmarked line shows the relative frequency of satisfiable formulas (moving from 1 in the left to 0 in the right). The performance of two versions of HyLoRes is shown, in New-HyLoRes the set-at-a-time subsumption checking strategy is implemented. 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). This test does not involve intense modal reasoning which explains why *SAT outperforms RACER.

In graphic b) we compare RACER against HyLoRes. We show the two best and the worst order criteria for the selection of the given clause. Prioritizing the maximum modal depth $d$ or the minimum prefix $p$ in clauses, caused an important degradation of the performance. Further testing to verify if this is the standard behavior is in progress.

In graphics c) and d) we compare HyLoRes with RACER and SPASS on the hand-tailored test sets defined in [10]. We can see here clearly that DL provers like RACER have still a huge lead. RACER is able to solve almost all problems within the time out of 100 seconds per problem. The performance of HyLoRes and SPASS are comparable, but we should point out that the test were performed using the standard translation of modal formulas into FOL. Optimized translations like the one used by MSPASS [17] or the one presented in [6] would greatly improve the performance of SPASS.

To close this section, another 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, for example, providing a new translation of QBF formulas which makes use of the hybrid expressive power) 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 in Version 1.0 has been largely achieved: direct resolution can be used as an interesting, and 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 [9].

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 [7] can be adapted for our calculus. Also, more effective normal forms 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 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 $\mathcal{H}(A)$ 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


[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.


