Lotrec: a generic tableau prover for modal and description logics
Luis Fariñas del Cerro, David Fauthoux, Olivier Gasquet, Andreas Herzig, Dominique Longin, Fabio Massacci

To cite this version:

HAL Id: hal-03523405
https://hal.science/hal-03523405
Submitted on 12 Jan 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers. L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
A Manifesto for a Generic Tableau Prover

The last years have seen a renewed interest in modal and description logics (MDLs). Better algorithms, coding, and technology have led to effective systems based on tableau and constraint systems [6, 7] to DPLL-based implementations [5], first order provers [8] and the inverse method [13]. PSPACE problems such as satisfiability are within reach for realistic instances [10] and potentially EXP-TIME problems stemming from real applications can also be solved [3, 7].

However, the comparisons now held at the Description Logic workshops and at the TABLEAUX conferences have also shown a major problem: the emphasis on performance is so strong that most implementors have restricted their prover to few fixed logics, hacking logics and strategies in their systems.

Yet, there are infinitely many MDLs and the choice of one logic over another is driven by modeling needs and computational constraints of one’s applications. A logic about actions and plans is likely to have different semantical and computational properties from a logic about database schemata. Even with the same logic, different search strategies may be needed for different applications.

If a user wants to use logics or even search strategies slightly different from those of the current systems, he must hack his own prover. “What if I use this constructor”, “What if I change the order of rules” experiments are almost impossible for somebody who is not the implementor of the system.

To answer the needs of users wishing to experiment and model with different logics or strategies there is a need of a generic theorem prover for MDLs. A prover playing the same role as Isabelle [12] or PVS [11] for higher order logics, while being less complex. If the user is not the same person as the programmer of the prover, one needs (a) flexibility and portability of the implementation, (b) high-level languages for tableau rules and strategy definition, and (c) user-friendly interfaces.

Lotrec is such a generic tableau prover. It aims at covering all logics having possible worlds semantics, in particular MDLs1.

1 Behind Lotrec is the work on modal tableaux with back/forth rules [9], graphs [1, 2], and its DL counterpart in [7]. Lotrec has been implemented by D. Fauthoux [4].
2 Architecture

The aims of flexibility, portability, and nice interfaces had motivated the choice of Java as the implementation language. Within such an object-based programming language, Lotrec raises Java’s event-based architecture to a declarative approach.

Tableaux are usually presented in tree form. In Lotrec, they are generalized to graphs in order to enable complex MDLs such as the ML of confluence, or MLs with complex interactions between knowledge and action. Graphs also allow to visualize possible worlds models (e.g., after transitive or symmetric closure of accessibility relations). Graph nodes are labelled by formulae, and edges by any term (possibly containing variables). Lotrec graphically presents the tableaux it has generated (Fig. 1), and allows for “drag-and-drop” restructuring of its shape by the user. (It remains to implement “drag-and-drop” interfaces for defining rules, strategies…As in Isabelle and PVS this is currently done via textual files.)

3 Defining the language of your own pet logic…

Before starting to define the rules, the user must define the logical connectives he wants to use. Let us take the definition for a logic of actions:

<table>
<thead>
<tr>
<th>connector</th>
<th>num</th>
<th>assoc</th>
<th>formula</th>
<th>priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>falsum</td>
<td>0</td>
<td>true</td>
<td>&quot;FALSUM&quot;</td>
<td>4</td>
</tr>
<tr>
<td>and</td>
<td>2</td>
<td>true</td>
<td>&quot;.&amp;.&quot;</td>
<td>3</td>
</tr>
<tr>
<td>not</td>
<td>1</td>
<td>true</td>
<td>&quot;.&quot;</td>
<td>5</td>
</tr>
<tr>
<td>feasible</td>
<td>2</td>
<td>true</td>
<td>&quot;&lt;<em>&gt;</em>&lt;&quot;</td>
<td>4</td>
</tr>
<tr>
<td>after</td>
<td>2</td>
<td>true</td>
<td>&quot;[&lt;]_&lt;&quot;</td>
<td>4</td>
</tr>
</tbody>
</table>

Consider e.g. the last definition: after is the internal name of the connective, and 2 is the number of its arguments. The rest of the parameters defines the graphical presentation: true means that the connective is associative, "." stipulates that the internal (after hit (feasible smash broken)) will be written [hit]<smash>broken on the screen. 4 is the priority of the connective wrt the others.
A rule consists of a descriptor and an action part. The former contains the applicability conditions, while the latter contains operations on tableaux.

A tableau rule is interpreted as mapping a pattern to a pattern, where patterns are connected fragments of a given tableau: if the descriptor part matches the pattern then that pattern is replaced by the result of the action part.

Consider the standard multi-modal logic $K4_n$, with modal operator $\text{neq}$. A handful of rules is in Fig. 2. The rule for handling formulae of the form $\text{neq } A$ is the rule $\text{K}$. It says if some node $\text{node0}$ of a Kripke structure is linked to a node $\text{node1}$ via the relation $R$ and contains a formula of the form $\text{neq } R A$, then $A$ is added to $\text{node1}$. $R$ and $A$ must be variables in order to make the rule work as a schema. Constants are useful for specific formulae or relations.

Lotrec also allows for manipulating expressions on links. Thus one may easily have logics like dynamic logics where links are labelled by complex programs.

Sometimes the ordered application of rules via a strategy is not enough to ensure termination or completeness; or maybe a user just wants to test various strategies. Thus, nodes, links, and formulae in nodes can be marked. For instance, for logics with transitive accessibility relations such as $K4$, before creating a new node we may wish to check whether the current node is not included in some ancestor. The rules $\text{diamond}$ and $\text{InclusionTest}$ in Fig. 2 do that.

We may want to do more than just simple propositional reasoning and we may have what are called concrete domains or quantitative domains. Then we allow for oracle calls to programs exterior to Lotrec. These programs typically are rewriting procedures, constraint solvers, SAT provers, etc.
5 Defining your pet search strategies...

After you have the rules defining the semantics of your logics you may want to say how to combine and apply them. Search Strategies do exactly that by mapping tableaux to tableaux (or sets thereof for disjunction-like rules) by repeatedly applying rules in some suitable ways.

If a user has a set of tableau rules \{rule1, rule2, ..., ruleN\} that has been proven to be complete for the logic under concern, then he can immediately implement a complete theorem prover for this logic via a fair strategy, which repeats applying all rules sequentially. Such a naive strategy is written:

\[
\text{repeat allRules rule1; rule2; ... ; ruleN end end}
\]

Here, to apply a rule means to apply the rule simultaneously to every possible pattern in the tableau. For our rule "K" this means simultaneous application to every formula of every node.

Your pet logic may require more sophisticated strategies for termination, or completeness, soundness etc. or, again, you may just want to experiment. Thus we allow for search strategy programming with the following constructs:

\[
\text{strategy ::= rule |}
\text{repeat strategy end |}
\text{allRules strategy1; strategy2; ... ; strategyN end |}
\text{firstRule strategy1; strategy2; ... ; strategyN end}
\]

We use firstRule rule1; rule2; rule3 when we want to apply the first applicable rule, and we use allRule rule1; rule2; rule3 to apply all applicable rules among rule1, rule2, ... in that order. For instance, if rule1 and rule3 are the applicable rules then firstRule will only apply rule1, whereas allRule will apply first rule1 and then rule3 to the result of the first rule. There is a close similarity with Isabelle “tacticals” FIRST and EVERY for combining tactics.

In Fig. 3 we show an example of a correct but inefficient strategy for K4. With Lotrec it is easy to experiment and see what happens and what we save if we move the not and rule outside the innermost repeat (which is one of the improvements to make the strategy more efficient). More efficient versions are available at the Lotrec webpage.

At present, strategies are applied globally, to all nodes and formulae. We plan future refinements where users may wish to define orderings among nodes or formulae and strategies applying rules only to the first element in the order.

6 Great, where can I find Lotrec?

Lotrec is available at http://www.irit.fr/ACTIVITES/LILaC/Lotrec

One can also find there a library containing the standard modal logics such as K, KD, KT, K4, S4, KB, PDL, the modal logic of density, and several logics of knowledge and action, as well as intuitionistic logic.
repeat firstRule
   "stop";
   // propositional rules
   repeat allRules
      "not not"; "and"; "not and"
   end
end

// generate and check successors
allRules
   "diamond"; "K"; "4"; "InclusionTest"
end

end

Fig. 3. A possible strategy for the logic $K4$

7 Acknowledgements

Thanks to the LILaC members who tested Lotrec and helped to improve it by their comments, in particular L. Aszalos, J.-F. Condotta, Th. Polacsek, and S. Suwanmanee. F. Massacci acknowledges the CNR Fellowship CNR-203-07-27.

References