LoTREC: Logical Tableaux Research Engineering Companion

Olivier Gasquet, Andreas Herzig, Dominique Longin, and Mohamad Sahade

IRIT - Université Paul Sabatier — Toulouse (France)
{gasquet|herzig|longin|sahade}@irit.fr
www.irit.fr/recherches/LILaC/Lotrec/

Abstract. In this paper we describe a generic tableaux system for building models or counter-models and testing satisfiability of formulas in modal and description logics. This system is called LoTREC. It is characterized by a high-level language for tableau rules and strategies. It aims at covering all Kripke-semantic based logics. It is implemented in Java and characterized by a user-friendly graphical interface. It can be used as a learning system for possible worlds semantics and tableaux based proof methods.

1 Introduction

In general, most tableaux-based theorem provers [7, 6] and DPLL-based implementations [10] use many different optimization techniques to speed up proof search in a particular logic. But they are limited in the sense that they treat a fixed set of logics: we cannot use them to treat other logics directly.

If the user is not the programmer of the prover, he needs (1) high-level languages for tableau rules and strategy definition, (2) user-friendly interfaces and (3) flexibility and portability of the implementation. LoTREC is a such generic tableau prover. It is designed for researchers specialized in modal logic and for students concerned by learning modal logic. It aims at covering all logics having possible worlds semantics, in particular modal and description logics. LoTREC not only allows to test satisfiability, but it also builds models. In that perspective LoTREC traverses the whole search space and keeps all the models in memory. This is also the price to pay for a generic tool, whose efficiency cannot be compared with other more specialized tableaux provers. such as FaCT [14], MSPASS [11] or LWB [8].

In its genericity LoTREC is similar to the higher-order provers Isabelle [12] and PVS [13]. In its aims LoTREC is similar to the Tableaux Workbench TWB [9]. With TWB, we can go beyond traditional tableaux systems, too, and handle for example modal tableaux with back/forth rules [3]. The differences are as follows: while LoTREC is semantics-driven and tries to build models, TWB is rather syntax-driven and closer to sequent calculi. With TWB, it is neither possible to implement calculi that require two or more passages of the tableaux algorithms (as required for example for Linear Temporal Logic LTL), nor is it possible to test loops or to handle relational properties like weak-directness. Moreover, TWB does not offer proof editing capabilities.

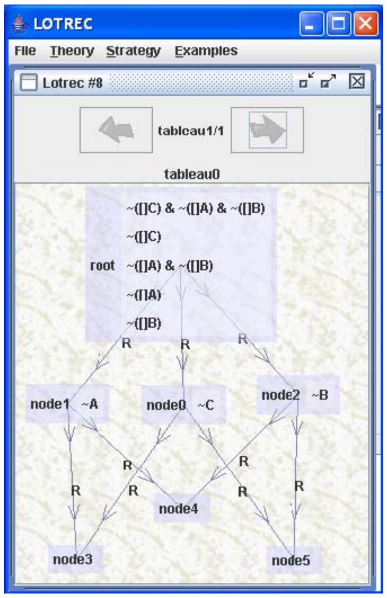
LoTREC is implemented in JAVA, a flexible and portable object oriented programming language. A first version was implemented by D. Fauthoux [1]. This version neither allowed to implement semantic properties like linearity, confluence, density, nor to test for example if a formula $\Diamond A$ is realized in a node. Recently, a new version of LoTREC has been implemented by M. Sahade [2], with a lot of modifications in its high-level language and its capacities. This motivated us to write this paper.

The theoretical bases of our system LoTREC are presented in [15] that is submitted as research paper at TABLEAUX'05 and that is available from our website.

In the next section, we will present the general architecture of LoTREC, and in section 3 we present the main extensions we have made w.r.t. the preceding version.

2 LoTREC in a nutshell

Tableaux are usually presented in tree form. In LoTREC, they are generalized to RDAGs (Rooted Directed Acyclic Graphs) in order to enable complex logics such as the modal logic of confluence, or modal logics with complex interaction between knowledge and action. Graphs also allow to visualize possible worlds models. Graph nodes are labelled by sets of formulae, and edges by terms.



LoTREC graphically presents the tableaux it has generated (see picture), and allows the user to restructure them by "drag-and-drop". It can also output the results in a file for reuse. (In particular we plan to interact with other provers.) LoTREC allows the user to define his logic and search strategy via textual files (as in Isabelle and PVS), or via a graphical interface. The user has to define the logical connectors, the semantic tableaux rules and the research strategy.

A tableau rule is interpreted as mapping a pattern to a pattern, where patterns are connected fragments of a given RDAGs. Every rule has a *condition* part, e.g. `if node hasElement (variable A)`, `if node1 isLinked node2 R`, . . . , and an *actions* part, e.g. `do add A to node0`, `do link node0 node1 R`, . . . The condition part contains the conditions to be verified to apply the rule, and the action part contains the actions to apply if all the conditions are satisfied.

Very briefly, LoTREC strategies iteratively do the following: basically, each entity that has just been modified or created (formula, node, link,...) calls for application all rules whose condition part matches some subgraph containing this entity. These calls are managed by the Java event machine.

For more details about how to define connectors, rules and strategies see [2].

3 Improvements w.r.t. the previous version

The language of LoTREC allows for a very declarative way of programming, but this simplicity induced a lot of redundancies in the previous version (PV for simplicity).

3.1 Duplication

As we have said, at every iteration step, each entity that has just been modified or created calls for application all rules that may be concerned. In the PV, in the case of k conditions in the condition part of a rule, each rule was applied k times (one to each elementary component of the pattern under concern).

Such redundancies quickly turned out to be prohibitive for more complex logics: too much time and memory is wasted. This has been fixed in the new version, and now rules apply only once to each applicable pattern. This permits to treat logics like LTL or product logics in a more efficient manner.

3.2 Implementing Linearity and Seriality

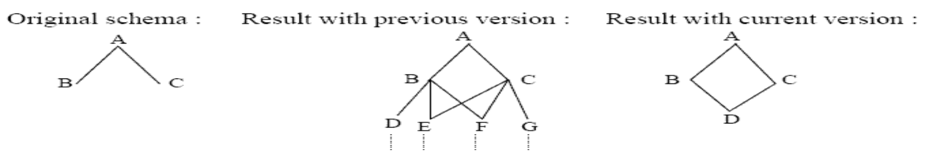
With PV we could not create only one successor of a node. For example in LTL, for a node which contains `next A` and `next B`, PV creates two different successors (while what we want is one successor only). This is due to the fact that the rule finds two instances of the conditions, and so it executes the action part for every instance.

Due to this problem we were not able to treat in a satisfactory manner logics where the accessibility relation is linear or serial, like for LTL or the logics with the axiom of seriality D because PV had to do redundant work and thus lost time and memory.

Our solution was to create a new action called `createOneSuccessor` which creates only one successor to a given node if it hasn't a successor yet, otherwise the rule continues to execute the other actions. And so now, we can build serial and linear models, and treat the corresponding logics in a satisfactory way.

3.3 Implementing Confluence

Let's apply the confluence rule to the left graph of the following schema:



Normally if we want to apply the confluence rule we will get one successor for example called `E` and we link `B` and `C` to `E`, but PV does not stop!

The problem is: in the first step it gives 4 successors because (1) we didn't specify that the two successor nodes must be different, (2) PV does not consider that two patterns like `{A,B,C}` and `{A,C,B}` are equivalent modulo commutativity and must be considered as one pattern.

For the first problem we add two new conditions which test if two given nodes are identical or not. For the second one we had to modify the internal operation of the rules to be able to express if a rule must consider two patterns equivalent modulo commutativity. In other words, we have now two kinds of rules: normal rules and commutative rules.

In the new version, the syntax of rules is changed, and a rule takes an optional boolean parameter which is by default set to `false`. To specify if the rule is commutative in case of `true`, or not in case of `false`.

3.4 Test if an Expression is Realized

In some logics where we stop by inclusion test (like in LTL and PDL), in order to know if we can build a model we have to do some postprocessing: we have to test whether a formula of the form `pos A` is realized or not. So we need to traverse the graph and to mark those `pos`-formulas that are realized. With the PV we could not perform such tasks, and thus couldn't treat LTL and PDL.

Our solution for this problem was to create two new actions: `markExpression` and `unMarkExpression`, which mark or unmark expressions in a specified node and two new conditions: `isMarkedExpression` and `isNotMarkedExpression`, to test if an expression is marked or not.

3.5 Extending the Strategy Language

The strategy language of the PV had only two operators: `repeat` and `firstRule`. The first one repeats the application of a list of rules in the order of their appearance in the list. `firstRule` applies only the first applicable rule among a list of rules.

So with this restrictive language we couldn't build strategies where we want to apply a sequence of rules only once. For example, in model checking one may have a rule which builds the model and other rules to do the remaining work. The model building rule must be applied once and after it works we apply the other rules.

In the new version, we have extended the strategy language by a new operator `allRules` which applies the applicable rules among a sequence of rules only once.

3.6 Graphical User Interface and Execution via Internet

In the new version, the programming interface is far more user-friendly: The user can define connectors, rules, strategy and the formula to test in the corresponding space. He can choose his connectors and rules by selecting them from a list. He can save his logics and strategies in files and the reuse them in a user-friendlier way, and choose between printing the result of the computation on the screen, or saving it in a text file. Finally, one can run LoTREC now via Internet from the LoTREC home page.

4 Conclusion

We believe LoTREC is a nice tool to learn and play with modal and description logics and tableaux systems for them, through an improved graphic interface which allows the user to easily define his logic and strategy.

We have implemented with LoTREC several theorem provers for the logics: K, KB, KD, KT, K4, K5, S4, S5, KD45, PLTL, PDL, K4+confluence, K4+density and several logics of knowledge and action as well as intuitionistic logics. One can find a library of all defined logics at the LoTREC web site.

Acknowledgements

Thanks to all members of LILaC team at IRIT for their comments and remarks which helped us to improve LoTREC.

References

1. F.Del Cerro, D.Fauthoux, O.Gasquet, A.Herzig and D. Longin, Lotrec: the generic tableau prover for modal and description logics. In International Joint Conference on Automated Reasoning, LNCS, 2001, 453-458.
2. M.Sahade. LoTREC: User Manuel available at:LoTREC home page.
3. F. Massacci. Single step tableaux for modal logics: methodology, computations, algorithms. *JAR*, 24(3):319–364, 2000.
4. M. Castilho, L. Fariñas del Cerro, O. Gasquet, and A. Herzig. Modal tableaux with propagation rules and structural rules. *Fund. Inf.*, 32(3):281–297, 1997.
5. L. Fariñas del Cerro, O. Gasquet. Tableaux Based Decision Procedures for Modal Logics of Confluence and Density. *Fund. Inf.*, 40(4): 317-333 (1999).
6. I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. *Log. J. of the IGPL*, 8(3):239–263, 2000.
7. I. Horrocks and P. F. Patel-Schneider. Optimizing description logic subsumption. *JLC*, 9(3):267–293, 199
8. A Heuerding. LWB theory http://www.lwb.unibe.ch
9. P. Abate and R. Gore. System Description: The Tableaux Work Bench
10. E. Giunchiglia, F. Giunchiglia, R. Sebastiani, and A. Tacchella. Sat vs. translation based decision procedures for modal logics: a comparative evaluation. *JANCL*, 10(2):145–173, 2000.
11. U. Hustadt and R. A. Schmidt (2000) In Dyckhoff, R. (eds), Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2000). Lecture Notes in Artificial Intelligence, Vol. 1847, Springer, 67-71
12. L. C. Paulson. *Isabelle: A Generic Theorem Prover*, LNCS. Springer-Verlag, 1994.
13. S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In *Proc. of CADE'92*, LNAI, pp. 748–752, 1992.
14. I.Horrocks and P.K. Patel-Schnieder. Optimising propositional modal satisfiability for discription logic subsumption. In LNCS 1476,1998.
15. O. Gasquet, A. Herzig and M. Sahade: Programming Modal Tableaux Systems. Submited to Tableaux05.