HyLMoC
A Model Checker for Hybrid Logic

Alessandro Mosca, Luca Manzoni, Daniele Codecasa

Department of Computer Science, Systems and Communication (DISCo)
University of Milan - Bicocca
alessandro.mosca@disco.unimib.it

Abstract. The current technological trend depicts a scenario in which space, and more generally the environment in which the computation takes place, represents a key aspect that must be considered in order to improve systems context awareness. Reasoning about such context can be interpreted as spatial reasoning, which means not only to be able to carry out inferences about the space itself, but also inferences about spatially related information according a given background knowledge. Past works have shown that hybrid modal logics are a powerful and rich formalism to model qualitative spatial reasoning and reasoning on information spread into graph-like structures. In this paper we present the preliminary results we obtained in designing and implementing HyLMoC, a model checking system for hybrid modal logics. The functionalities of the model checker are based on backend module that accepts as inputs a list of hybrid modal formulas and the specification of a labeled graph structure that is compliant with the characteristics of a Kripke model. The current implementation of the backend allows to perform different reasoning tasks with respect to those inputs: *Local* and *Global model checking*, and *All-worlds model checking*.

1 Introduction

As shown in [1], Modal Logic, originally conceived as the logic of necessity and possibility, has developed into a powerful mathematical discipline that deals with (restricted) description languages for talking about various kinds of relational structures, as spatial and temporal localizations of entities (see [2–4]). Modal logic has been widely employed with respect to time dimension and various time interval logics have been developed (as shown in [5]). On the other hand, as far as the representation of space is concerned, within the logical approach there is a vast literature of which Davis presents a good overview in [6]. Interesting theoretical research in the area of temporalized spatial logics (with an accurate analysis of decidability and complexity issues), can be found in the works of Bennett, Cohn, Wolter, and Zakharyaschev; a result is the PSTL, a two-dimensional logic capable of describing topological relationships that change over time (see [7]).

Hybrid languages extends multimodal languages (characterized by a set of modal operators $MOD = \{\langle\pi_0\rangle, [\pi_0], ..., \langle\pi_n\rangle, [\pi_n]\}$ and a set of propositional variables $PROP = \{p_0, ..., p_n\}$) by adding: (i) a nonempty set of propositional symbols $NOM =$

$\{i_0, ..., i_n\}$, disjoint from $PROP$, that are called *nominals*, and (ii) a *satisfaction operator* of the form $@_i$ for each nominal $i \in NOM$. Informally, we just recall that a Kripke model for hybrid logic is a triple $(W, \{R_\pi | \pi \in MOD\}, V)$ where $(W, \{R_\pi | \pi \in MOD\})$ is a frame and V is a hybrid valuation. Semantics of hybrid formulas is defined as usual for modal logics, but (i) nominals are interpreted to be true at one and only one world of the model (their *denotation*), and (ii) given a model $\mathcal{M}$ and a world w in the model, formulas preceded by satisfaction operators are interpreted as follows:

$$M, w \vDash @_i\varphi \text{ iff } M, w' \vDash \varphi \text{ , where } w' \text{ is the denotation of } i$$

Hybrid logics allow to express in the language itself, by means of nominals and satisfaction operators, sentences about the satisfiability of formulas; formulas preceded by satisfaction operators allow in fact to represent statements about specific states of the model [8, 9]. Suppose we deal with a hybrid language containing the nominal 'living room' and the proposition 'alarm', the formula

$$@_{livingRoom}alarm$$

states that at the state of the model denoted as living room (think for example to a smart home environment), it is true that an alarm has been triggered. Moreover, different properties of hybrid logic revealed to be extremely useful for correlation of information that are distributed over relational structures. In particular, modal-like logics help to focus on qualitative relational aspects of networked space (e.g. containment, orientation, and reachability relations among physical entities) requiring the specification of a family of modal operators, whose meaning goes to define the underlying relational structure. With respect to some peculiar traits of hybrid logic, the combination of modal perspective and hybrid expressions enhances representation and reasoning in this context. The modal local perspective over reasoning helps in fact to easily move through the graphs following operators and their respective relations. Hybrid expressions with nominals, satisfaction operators allow not only to name specific states within these relational structures, but also to reason over equality and inequality statements about them. As an example, the formula:

$$@_i(alarm \wedge \Diamond_{NI} k) \rightarrow (@_j(alarm \wedge \Diamond_{NI} k \wedge \neg i) \rightarrow @_k alarm)$$

makes use of the negated nominals to explicitly states that the states denoted as i e j are distinct (the formula says that an alarm must be triggered only if two states an alarm has been triggered at two distinct states i and j, that are both contained in k). Furthermore, we can take advantage from the characteristics that are peculiar of hybrid logic: frame definability and modularity. In fact, the hybrid machinery allows to model quite specific conditions defining the frame of reference, and this would have not been possible within plain modal logic. We can use 'pure' formulas of the hybrid language, i.e. formulas that do not contain propositional symbols, to constraint the meaning of the accessibility relations in the Kripke model; the following formulas express irreflexivity and antisymmetry of a generic relation π:

(*irref*)	$@_i \neg \langle \pi \rangle i$
(*antisym*)	$@_i [\pi] (\langle \pi \rangle i \rightarrow i)$

In the past years, our work has been mainly focused on the exploitation of hybrid logic expressivity and inferential mechanisms in order to perform commonsense spatial reasoning in the contexts of correlation of information for pervasive and ubiquitous computing, mobile systems, context aware agents and multi-agent systems design. Here we briefly introduce some pointers to our previous papers that provide an accurate introduction of our exploitation of hybrid logic formalism to correlation and spatial reasoning. [10] introduces the approach based on commonsense spatial representation and reasoning to model context aware reasoning. This approach has been further described and discussed in [11], together with the underlying knowledge based approach to information correlation (with the related pros and cons) and the comparison with other non knowledge based approach. Moreover the last paper discuss the choice of qualitative spatial models and qualitative spatial reasoning techniques which are similar to the reasons discussed in [12]. The formal characterization of the Hybrid Commonsense Spatial Logics (HCSLs) is given in [13], together with a calculus and the discussion of deduction examples in a Smart Home context; here the relationship between our approach and other prominent logics for qualitative spatial representation and reasoning is discussed. We refer to this last work and to [11] also for the comparison with other approaches to qualitative spatial representation and reasoning (QSRR). As for the consideration of spatio-temporal events, an example of spatio-temporal correlation (but where spatial representation is simplified up to the 1D) is presented in [14]. SAMOT, the system devoted to traffic monitoring over an highway based on this approach is described in [15].

The main reason for which we started thinking to HyLMoC one year ago was the intention to develop a system for automatizing the checking of hybrid logic formulas against graph-like and network structures. The existence of an efficient model checker for hybrid logic was in our mind a way to make executable the models we defined in the above mentioned application fields, on one hand, and to improve the reasoning capabilities of the developed systems, on the other. In particular, as the introduced examples suggest, we were interested to exploit hybrid logic model checking techniques to perform the following tasks: (i) The correlation of distributed qualitative information along a local as well as a global perspective[1]; and (ii) the verification of consistency properties of the structures at hand (e.g. to verify that the specification of a given structure respects a set of formal properties - reflexivity, antisymmetry, etc. - we can define by means of pure hybrid formulas). In this paper we introduced the main characteristics of a model checker for hybrid logic we start developing with these aims in mind. To the best of our knowledge, there is only another example of a computational tool for model checking that exploits hybrid logic expressivity, that is the `MCLITE-MCFULL` model checker,

[1] With 'local perspective' we mean the possibility to check if some property is satisfied at the current state of evaluation - for example, the state at which a specific device is located - and with 'global perspective' the possibility to check if a property is satisfied at a specified state of the model - for example, the living room of a smart home. The second task was

whose algorithms have been designed by Massimo Franceschet and implemented by Luigi Dragone [16]. We briefly discuss at the end of the paper some preliminary, and still partially incomplete, results coming from a set of comparative tests we made about the execution time of these systems.

The paper is organized as follows: The next section introduces the way HyLMoC deals with the Kripke model definition, together with the formula syntax and the grammar according to which HyLMoC parses the input formulas. The main algorithmic structure of HyLMoC is introduced in Section 4, while Section 5 is dedicated to the introduction of the core backend algorithms of the system. Experimental results are introduced in Section 6, while some concluding remarks and notes on the planned future works end the paper.

2 The HyLMoC model checker

Given a formal specification of a model and a property, Model Checking can be defined as a reasoning task of checking whether the model satisfies the property [17–19]. The model is often specified as a relational structure (e.g. a labelled graph), and the property as a formula expressed in the language of some logic. Model checking thus consist in exploring the model, jumping from one node to another according to its internal structure, in order to verify the satisfiability of the formula itself. Usually this task is performed in order to automatically verify if a model, representing hardware and software systems, meets some given specifications, e.g. safety requirements such as the absence of deadlocks and similar critical states that can cause the system to crash. We refer to [20] for a comprehensive introduction of the most important model checking techniques used in the field of systems verification, and to an introduction to the automated method based on temporal logic for this tasks. The use of modal and temporal logics for performing model checking tasks is well documented in the related literature and they have conducted to very efficient symbolic model checker implementations (see, for example, [21]).

The HyLMoC system is a model checker explicitly devoted to deal with hybrid logic formulas and Kripke models. This means that the properties one can verify using HyLMoC must be expressed using the language of hybrid logic and the relational structures representing the phenomena of interest must be compliant with the formal characteristics of Kripke models. The HyLMoC system was developed to perform the following tasks:

- [**Local** model checking] Taking a model $\mathcal{M}$, a state w, and a formula φ, the system outputs the answer 'yes' if φ is satisfied at w in $\mathcal{M}$, 'no', otherwise;
- [**Global** model checking] Taking a model $\mathcal{M}$ and a formula φ, the system outputs the answer 'yes' if φ is satisfied at all the states in $\mathcal{M}$, 'no', otherwise;
- [**All-Global** model checking] Taking a model $\mathcal{M}$ and a formula φ, the system stops its computation and outputs the answer 'yes' when it finds the first state at which φ is satisfied, 'no', if there is no state satisfying φ; and,
- [**All-States** model checking] Taking a model $\mathcal{M}$ and a formula φ, the system outputs the answer 'yes' if there exist at least a state w in $\mathcal{M}$ that satisfies φ, and prints

out the set $w_1, w_2, \ldots, w_n$ of all the states in $\mathcal{M}$ that satisfy φ (where $1 \leq n \leq |W|$, and $|W|$ is the cardinality of the set of states of the model).

Consider that in standard model checking, we usually have some kind of transition system, and we want to model check formulas in runs of such transition system: either over the set of all individual runs (using linear time logics like LTL), or in the tree obtained superposing all possible runs (using tree logics like CTL). This is not what HyLMoC does: HyLMoC checks whether a formula is true (globally true or locally true) in a given model, and the model has to be taken as a finite structure. Although this is not relevant in the case of hardware and software system verification, the presence of the global task becomes increasingly interesting when model checking is applied to different domains (e.g. to the query processing of semistructured data and to commonsense spatial reasoning cited above), where, it is mandatory to check if a given property is true at each state of the model.

In what follows, we report the requirements that have guided the design and implementation of the HyLMoC model checker during all the phases of its development:

- The grammar used for the specification of the formulas must support the addition of further modal operators. For example, the addition of a ternary modal operator must be possible without changing the parser module of the model checker.
- More that one method for local and global model checking should be added without changing the language used to specify the model or the formula. In particular, different application fields (e.g. spatial information correlation task) may require the use of hybrid modal languages with different expressivity; the treatment of the formulas defined on the basis of these languages and the respective Kripke models must be preserved, and the exploitation of new optimized model checking methods for them should be allowed without breaking the compatibility with other instances of the model checker.
- The model checker must scale on multiprocessor and multicore systems, especially when the computational effort mostly depends on the throughput (i.e. number of formulas that can be checked per unit of time) instead of on the checking time of a single formula.

The first two requirements are mandatory in order to make feasible the reusing of the already written code, for example, in order to support the implementation of specific procedures for new operators in the language, or to help the prototyping of new algorithms for formula model checking. As regards to the third requirement, there are at least two distinct but interrelated reasons that forced us to work in this direction: the first one obviously refers to the goal of increasing the final performances of the model checker system, while the second one arises from the consideration that parallel multiprocessing architectures are already available on the market of desktop computers and this provides the possibility to design and implement systems for final users that exploit this new computational power.

3 The Kripke models and formulas definition

Due to well known portability and compatibility issues, the data specifying the Kripke models the HyLMoC takes as inputs are stored in XML files, formatted according to a suitable DTD. In particular, the models used by HyLMoC can be defined by using the same DTD scheme of the Franceschet and Dragone [16] system (thus providing a way to exchange model specifications among these two systems), or by using an improved version of this DTD that allows the possibility to explicitly define the formal properties of the relations characterizing the model. This resolves to be extremely useful in providing the model checker with model definitions that are formally consistent, on one hand, and guarantees a significant reduction of effort for end users in the input definition phase, on the other. In Section 4, the implemented *model completion* and *consistency check* procedures, based on the relations formal properties specification, are briefly introduced. The following paragraphs furnish examples on how a model can be defined according to the implemented DTD scheme, and introduce the grammar according to which the formulas are parsed by the system.

A **state** of a Kripke model is specified by means of a identifier and by the introduction of one or more nominals denoting it.

```
<world label="w1"/>
<nominal label="i" thruth-assignment="w1"/>
```

A binary accessibility textbfrelation is identified by a name, one or more formal properties, and a list (possibly incomplete) of the pairs of worlds for which the relation holds (the specification of the properties associated to the relation will be used by the system to complete the model).

```
<modality label="pi1" property="reflexivity simmetry">
  <acc-pair to-world-label="w2" from-world-label="w1"/>
</modality>
```

The DTD scheme and the related parser provides the possibility to introduce n-ary relations in model description as follow:

```
<relation label="r">
     <relation-element vector="w1 w2 w3" />
</relation>
```

As for the valuation function of the Kripke model, a new name for each textbfpropositional symbol is introduced together with the set of worlds at which it is true.

```
<prop-sym label="p" truth-assignment="w1 w2"/>
```

Figure 1 shows a complete specification of a Kripke model that is compliant with the HyLMoC DTD.

The syntax used to express formulas in HyLMoC is a lisp-like one. It supports the definition of all the modal and hybrid operators of the language introduced above, and also the addition of new operators by using a special sequence of symbols (i.e. the

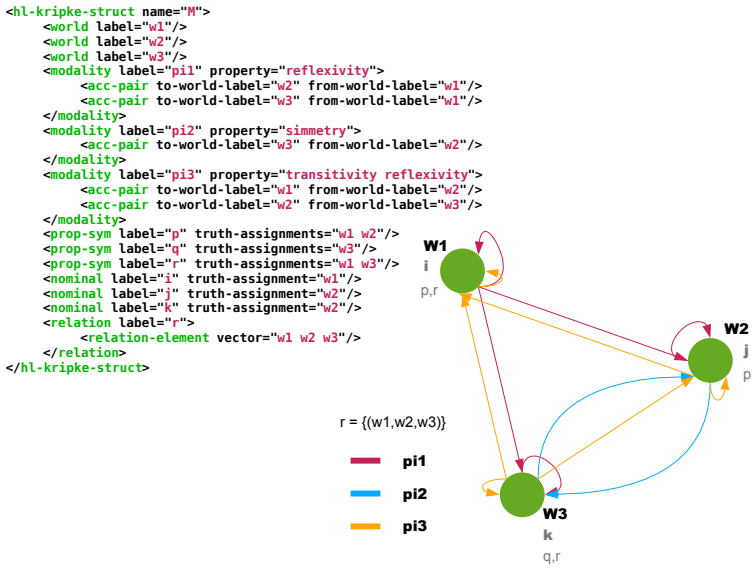

Fig. 1. An XML Kripke model specification and its diagrammatic representation.

operator name is between two question marks: `?operator_name?`). As an example, the formula $\langle \mathrm{r} \rangle (p \rightarrow q)$ is translated as `(<r>(imply p q))`. The following is the grammar for the formula language the model checker is able to interpret:

```
formula ::= prop | nominal | variable
          | T | F
          | (not formula) | (imply formula formula)
          | (and formula formula) | (or formula formula)
          | (at id formula)
          | (bind variable formula)
          | (exists variable formula)
          | (forall variable formula)
          | (<rel> formula) | ([rel] formula)
          | (<rel>- formula) | ([rel]- formula)
          | (?rel? formula+)
id       ::= var | nominal
variable::= [A-Z][a-zA-Z0-9_]*
prop     ::= [a-z0-9_][a-zA-Z0-9_]*
nominal  ::= [a-z0-9_][a-zA-Z0-9_]*
```

4 The HyLMoC modules

The algorithmic architecture of HyLMoC can be reduced to four different modules: (i) The model parser; (ii) the formula parser; (iii) the backend manager; and (iv) the work unit dispatcher. In what follows we briefly introduce the characteristics of each module.

The model parser. The model parser handles two different tasks. The first one is related to the parsing of the XML specificatoion of the model according to the introduced DTD scheme; the second one concerns the application of the structural transformations on the resulting model in order to verify that the model specification is consistent with respect to the formal properties of the contained accessibility relations. As regards to the second task, different properties can be defined for each of the relations of the Kripke model (actually, symmetry, transitivity, reflexivity and anti-reflexivity are implemented). Therefore, for each property there exists an *ad-hoc* module, we called *filter*, that performs the completion of the model with respect to its arcs (and generates warnings when inconsistencies are detected). Filters are used to manipulate the model and new filters can be added in a easily way. The presence of filters gives greater flexibility to the model checker and simplify the backends' implementation.

The formula parser. The formula parser creates a tree representation of the formula where every subtree is the representation of a well-formed formula of the implemented hybrid language. Since the semantic interpretation of possibily new operators is left to the backend, a specific node type, bringing information about the 'name and the 'arity' of the new operator, has been introduced. In order to check if the formula is well formed it is necessary to verify some properties that are contained in the model definition (for example, the first argument of an "`at`" must be a variable or a nominal, but not a propositional symbol). Every formula is parsed independently from the model where it will be checked; this choice minimizes the number of formulas' instances that must be present to HyLMoC when the same formula must be checked against different models.

The backend manager. The main function of the backend manager is to simplify and minimize the complexity of the backend implementation. To accomplish these goals the interface of the backend manager allows to specify the model checking task, and the underlying backends only have to check a formula at a single world of the model one at the time. Moreover, the backend manager may implement specific strategies to reduce the complexity of checking a formula; in the current implementation, the manager exploit a strategy to save resources in the case where the evaluation of a formula does not depend on the current world of evaluation. This strategy is done by checking if all the world-dependent sub-formulas have a parent that is an `at` operator.

The work unit dispatcher. The model checker can work on more than one formula or one model at the time, thus giving the possibility to exploit a certain degree of parallelism. The idea here is that it is possible to create self-contained units of work that can be executed concurrently without making the backend implementation difficult. We defined the work unit in order to contain: (i) A Kripke model; (ii) a formula; (iii) an instance of the selected backend; and (iv) a list of optional flags to specify what kind of task the model checker should be done (i.e. local, global, etc.). This kind of work unit can be executed concurrently if the backend instances do not make use of shared structures (in this case, the backend must have proper locking on these structures). Therefore, the model checker creates a queue of work units which are consumed by a pool of threads (one thread per processor available to the Java Virtual Machine).

5 Backend implementation

The formula evaluation process implemented by the HyLMoC backends follows the usual semantic conditions of the involved operators. This means that HyLMoC assumes a top-down approach in evaluating formulas, instead of the usual bottom-up. Since to check atomic formulas is the easiest task the model checker can do, the bottom-up approach is clearly the most efficient when modal propositional formulas are concerned (first check the leaves of the parse tree and identify the worlds at which the atomic subformulas are true, then reconstruct the entire modal formula by navigating the inverse of the respective accessibility relations). On the other hand, the same approach becomes unfeasible when binders are present, and the evaluation process should start from a variable x, as for example in $[r]{\downarrow}\, x.\langle r\rangle x$ (i.e. there no way to evaluate the variable x without any further information on the formula structure).

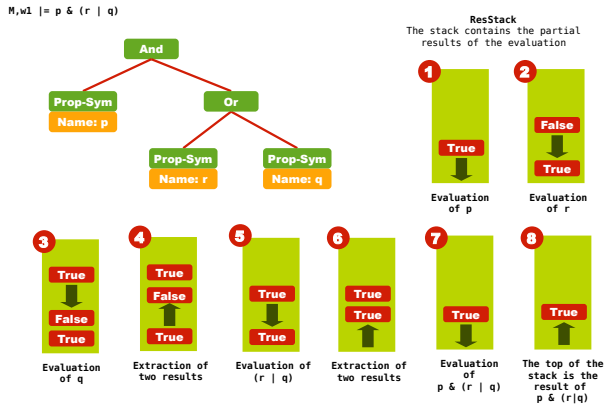

Fig. 2. An example of the use of the stack structure. The use of a pair of stacks for passing intermediate results simulate recursion on the parse tree.

Moreover, the top-down choice has supported a simple implementation of the backends, that is based on two stack data structures and a visitor pattern [22]. The first stack simply contains the intermediate results of the computation. For example, for evaluating the formula $p\wedge q$ the stack contains the thruth vales of p and of q that will be used to find the thruth value of the whole formula according to the semantic of the connective. The second stack contains the world where the evaluation takes place. For example, in checking if the formula $p\wedge @_i q$ is satisfiable at the world w, the stack contains w when evaluating the whole formula and p, but it contains i on top top when evaluating q. Moreover, the choice of following the semantic definition of the operators supported a simple implementation of the backends that is based on two stacks and a visitor pattern [22]. The first stack simply contains the intermediate results of the computation. For example, for evaluating the formula $p \wedge q$ the stack contains the thruth vales of p and of q that will be used to find the thruth value of the whole formula according to the

semantic of the connective. The second stack contains the world where the evaluation takes place. For example, in checking if the formula $p \wedge @_i q$ is satisfiable at the world w, the stack contains w when evaluating the whole formula and p, but it contains i on top top when evaluating q.

The use of the visitor pattern simplifies the mainteinance of the code, as well as the creation of new backends based on the current implementation (only new operands must be defined). This design choice makes the implementation of (simple) backend a task that can be accomplished in a very low count of lines of codes (the simplest implementation which support all the modal and hybrid operators is under 300 lines). In what follows, the pseudocode implementation of the existential quantifier is introduced:

Algorithm 5.1: EXISTS($formula f, hash\_table Variables$)

$$
\begin{cases}
variable\_name = first\_operand[f] \\
world\_list = worlds[model] \\
\textbf{while } notempty[worls\_list] \\
\quad \textbf{do} \begin{cases}
insert(variables, variable\_name, dequeue[world\_list]) \\
evaluate(second\_operand[f], variables) \\
remove(variables, variable\_name) \\
intermediate\_result = pop[results\_stack] \\
\begin{cases}
\textbf{if } intermediate\_result = \textbf{ true} \\
\quad \textbf{then } push[result_stack, \textbf{ true }] \\
\textbf{return}
\end{cases}
\end{cases} \\
push[result\_stack, \textbf{ false }]
\end{cases}
$$

The procedure changes the world where the evaluation will be done and then it starts the evaluation of the sub-formula. The procedure for the evaluation of the binder initializes the variable passed as first operand to refer to the current world.

Algorithm 5.2: BIND($formula f, hash\_table Variables$)

$$
\begin{cases}
actual\_world = top[world\_stack] \\
variable\_name = first\_operand[f] \\
insert(variables, variable\_name, actual\_world) \\
evaluate(second\_operand[f], variables) \\
remove(variables, variable\_name)
\end{cases}
$$

The pseudocode implementation of the diamond modal operator resembles the implementation of the existential quantifier because of its semantics (from a perspective close to the implementation, we consider it an existential quantification only on the neighbors of the current world):

Algorithm 5.3: POSSIBLE($formula f$)

$$
\begin{cases}
actual\_world = top[worlds\_stack] \\
relation\_name = relation[f] \\
world\_neighbours\_list = neighbours[actual\_worlds, relation] \\
\textbf{while } notempty[world_neighbours_list] \\
\quad \textbf{do} \begin{cases}
push[worlds\_stack, dequeue[world\_neighbours\_list] \\
evaluate(f) \\
pop[worlds\_stack] \\
intermediate\_result = pop[result\_stack] \\
\quad \begin{cases} \textbf{if } intermediate\_result = \textbf{ true} \\ \quad \textbf{then } push[result_stack, \textbf{ true }] \end{cases} \\
\quad \textbf{return}
\end{cases} \\
push[result\_stack, \textbf{ false }]
\end{cases}
$$

5.1 Backend improvements

Actually two backends have been implemented. The first one is a simple backend, called 'basic backend', that can be used as a baseline for comparison with other backends. The second one, called 'advanced backend' includes: (i) A cache for recently verified formulas; (ii) a module that makes statistical analyses on the model; and (iii) framework for adding operations that can reorder the structure of the formula without semantic changes (for example, to normalize all the formulas and to have an increase of the hit rate in the cache). Following the algorithmic architecture introduced above, the advanced backend provides optimization strategies that aim at decreasing the effort in time of the computation. A first kind of optimization is especially devoted to increasing the HyLMoC performances every time a set of long formulas have to be checked. It consists in using a cache that stores tuples in the form (φ, w), where φ is a formula and w is a world of the input model. The cache has been implemented in a naive way, but further work will deal explicitly with the following two improvements. The first one concerns the normalization of the formula before its insertion in the cache; this is necessary to provide matches on the basis of semantic equivalence, and not only on the basis of the syntactic one. The second improvement aims at providing a better management of the space in the cache; in order to do this we are working in adapting the model of the 'ARC cache' [23] to work on sets of pairs (φ, w) instead of pages of memory.

The way we can improve the efficiency of HyLMoC is not only limited to the semantic and syntax of formulas, but also on the structural characteristics of the model against which the formulas have to be checked. A further optimization deals in fact with the use of structural properties of the model in order to decide the *order* of evaluation of the branches of the `and`, `or` and `imply` nodes. This can be done by combining two parameters that are computed for every node of a parsed formula: (i) *cost*, and (ii) *probability*. Intuitively, the cost parameter is computed for every node of a parsed formula and, since every node is the root of a tree that is a well-formed formula, it can be interpeted as an estimate of the time that is needed to check the formula represented by the selected node. On the other hand, the second introduced parameter aims at representing

the probability that the result of the evaluation of the given formula at the current world is equal to 'true'. At the first step, the parameter is computed at each terminal node of the parsed formula by using statistical measures that come from an *ad-hoc* analysis of the model at which the formula has to be evaluated (e.g. number of outgoing relations per node, distribution of true propositions over the nodes). As a second step, the parameter is computed also in correspondence to each of the internal nodes of the tree, by combining this with the value obtained for the child nodes. The exploitation of both these parameters is essentially directed to perform syntactical transformations of a formula that have to be checked in order to obtain a semantically equivalent formula that can be checked in less time and with a lower computational effort.

6 Experimental results

In this section we present some preliminary results we obtained on the performances of HyLMoC once equipped with the 'basic' backend. The tests have been performed on a AMD Athlon 64 x2 5200+ (2.6GHz), with 4096 MB of RAM, running Ubuntu 8.04 and the JVM version 1.6.0_07. The design of an experimental campaign for a system like HyLMoC has not been immediate. On one hand, up to our knowledge there exists no available benchmark for hybrid logic model checking (consider that even if the `MCLITE-MCFULL` algorithms are described in publications, there is no accessible experimental results measuring their performances). On the other hand, we considered the choice of confining the tests to the formulas and models of a real-world project as too much restrictive and, in some sense, insufficient to provide the data variability we needed to test the system.

In order to design and partially automatize the experimental campaign, two ad-hoc algorithms generating random formulas and Kripke models have been implemented. The random formulas generator provides sets of formulas of growing complexity on the basis of the following fourth parameters: (i) formula depth; (ii) number of propositional symbols; (iii) number of modal operators; and (iv) number of nominals. A fifth parameter has been introduced just to fix the cardinality of the set of formulas one need to generate. The depth of a formula is understood here as the depth of the generated parse tree and, as such, it strictly depends on the introduced formula syntax (e.g. the formula `(not p4)` has depth 2 and 0 nested operators, while the formula

```
(and([r1](<r3>-w82))(forallY0(atY0(existsY1(atY1w11))))),
```

has depth 6 and 4 nested operators). The formula generator algorithm is flexible enough to support user in generating sets of formulas with bounded complexity according to the input parameters. Furthermore, the whole campaign has benefited from the combination of the formula random generator with an similar algorithm for the Kripke models specification (the details about the XML-based syntax for the Kripke model specification have been introduced in Section **??**). The algorithm takes as input the following parameters: (i) number of worlds; (ii) number of accessibility relations; and (iii) number of propositional symbols (this last parameter is necessary in order to set up the valuation function that is part of the model).

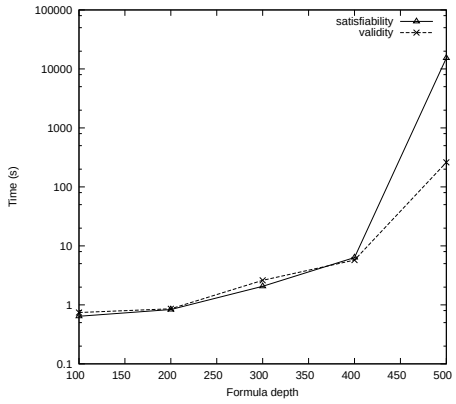

Fig. 3. ALL-GLOBAL and GLOBAL tasks with formulas of growing depth.

In particular, the curves in Figure 3 shows the performances of HyLMoC in performing the all-global and the global (blue line) tasks introduced in Section 2 with formulas of growing complexity. The structure against which the formulas have been checked is a Kripke model of 100 worlds, and 5 accessibility relations. On the other hand, the following experiments aim at verifying the performances of HyLMoC when the depth of the formulas is fixed. In particular, we used this kind of setting to compare the computational cost of different model checking tasks: Figure 5 and Figure 4 are about the results we obtained by testing HyLMoC on the all-global and global tasks respectively), while Figure 6 is about the all-worlds task.

All the present results have been produced by using batches of 50 formulas for each test. Moreover, due to the characteristic of Java libraries used in the implementation of HyLMoC, the process of scanning of graph nodes is not deterministic, and it may produce some unexpected result in terms of final observed performances of the system (e.g. the amount of time that is needed for checking if a formula containing existential quantifiers is satisfied at a state in model significantly depends on the order the graph nodes are explored). In order to reduce the undesirable effects of this indeterministic feature of the system, each test has been repeated 10 times and only the resulting average value has been recorded. The last set of results we present here is about a comparison between our basic backend and the `MCLITE` algorithm by Franceschet. At the present moment, `MCLITE` (and a still unstable version of `MCFULL`) has been implemented in HyLMoC and it is available as a further backend of the system. We already planned to produce a systematically comparison among these algorithms, and the results we introduce here must be considered as the first step in this direction. They concern the modal fragment of the full hybrid language and the `MCLITE` system, that is the algorithm devoted to deal with this fragment. With modal fragment we mean the language made by the hybrid apparatus of nominals and satisfaction operators, plus the diamond and box modal operators. The graphics show that the bottom-up approach of `MCLITE` work better than our top-down approach, especially when large models are considered. As one would

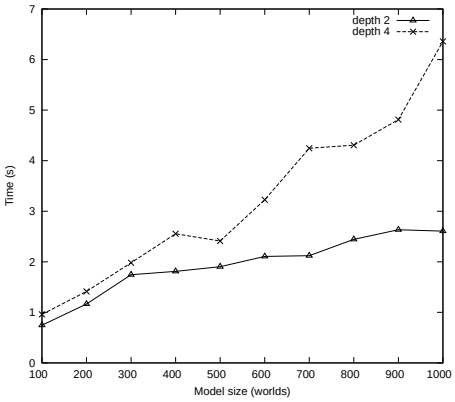

Fig. 4. GLOBAL

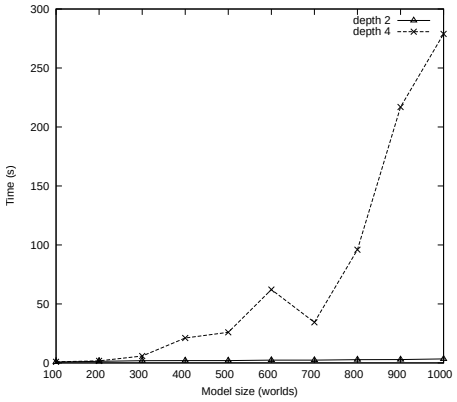

Fig. 5. ALL-GLOBAL

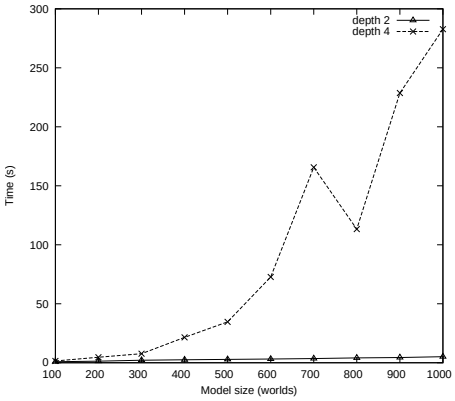

Fig. 6. ALL-WORLDS

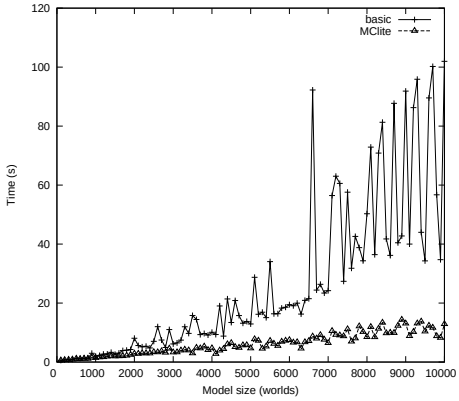

Fig. 7. ALL-GLOBAL, Formula Depth = 6, Language $\mathcal{H}_1(@, \langle\pi\rangle)$

expect, the performances of the top-down approach of HyLMoC are much more influenced by the order of visiting of the states than the bottom-up approach. In fact, starting from the leave nodes of the formula parse trees (i.e. from the contained propositional symbols), and following backward the semantics conditions of the modal operators, the bottom-up approach is able to identify a model for a formula, if it exists, in a time span that is significantly minor than that needed by by the top-down approach. Intuitively, the top-down approach is blind with respect to the possibility of finding a model for a formula (in a top-down approach every state of the model could be a good candidate for the final satisfiability of a formula containing modal operators), while the bottom-up approach resolves to be a "goal directed" process, a fact that produces an appreciable saving of computational resources.

7 Concluding remarks

Even if the present literature on model checking is truly vast, and a number of excellent computational tools and algorithms have been proposed [24], the implementation of model checkers for hybrid logic still remains a quite unexplored field of research. In this paper we introduced the main characteristics of a new model checker, called HyLMoC, that is able to deal with the expressivity of hybrid multimodal logic. The reasons why we started designing and implementing HyLMoC can be traced back to previous works on spatial reasoning and correlation of information we briefly introduced in the introduction of the paper. On the other hand, the first experimental results we obtained on HyLMoC, and the relevance the networks and graph-like structures have in the present scientific research, suggest to proceed further we its development and to try to improve its performances. In particular, the results confirm that the performances of HyLMoC are negatively affected much more by the increasing of the formula complexity than by the increasing of model dimension, and this justify the attention we are paying in developing an advanced version of the backend. At the present moment, we are working on performing a set of tests explicitly devoted to compare the performances of the basic and the advanced backends. As for the future developments, we plan to extend our work along the following lines: (i) To provide a significant number of translations of XPath and XQuery queries into the hybrid language in order to obtain a measure of the efficiency of HyLMoC with respect to dedicated query/answering engines; (ii) to apply HyLMoC to RDF-XML graphs and compare of our model checking technique with SPARQL query/answering (we believe that this study could be of some interest for the Semantic Web community); and (iii) to define a new (not yet provided in the hybrid logic literature) suite of benchmarks for hybrid logic model checking that could be useful to compare existing algorithms, as well as to develop new ones.

References

1. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge University Press (2000)
2. Prior, A.: Past. Present and Future. Oxford: Clarendon Press (1967)
3. Rescher, N., Urquhart, A.: Temporal Logic. Springer-Verlag, New York (1971)
4. Benthem, J.V.: The logic of time. Reidel, Dordrecht (1983)
5. Goranko, V., Montanari, A., Sciavicco, G.: A road map of interval temporal logics and duration calculi. Journal of Applied Non-Classical Logics **14**(1-2) (2004) 9–54
6. Davis, E.: Representations of commonsense knowledge. Morgan Kaufman, Los Altos CA, July, 1990, 515 pages (1990)
7. Bennett, B., Cohn, A.G., Wolter, F., Zakharyaschev, M.: Multi-dimensional modal logic as a framework for spatio-temporal reasoning. Applied Intelligence **17** (2002) 2002
8. Blackburn, P., Seligman, J.: What are hybrid languages? CLAUS-Report 83, Universitt des Saarlandes, Saarbrcken (November 1996)
9. Blackburn, P., Seligman, J.: Hybrid languages. Journal of Logic, Language and Information **4** (1995) 251–272
10. Bandini, S., Mosca, A., Palmonari, M.: Commonsense spatial reasoning for context–aware pervasive systems. In: Location- and Context-Awareness, First International Workshop, LoCA 2005. Volume 3479 of LNCS., Springer-Verlag (2005) 180–188

11. Palmonari, M., Bandini, S.: Context-Aware Applications Enhanced with Commonsense Spatial Reasoning. Lecture Notes in Geoinformation and Cartography. In: Map-based Mobile Services. Springer (2008) 105–124
12. Cohn, A.G., Hazarika, S.M.: Qualitative spatial representation and reasoning: An overview. Fundamenta Informaticae **46**(1-2) (2001) 1–29
13. Bandini, S., Mosca, A., Palmonari, M.: Commonsense spatial reasoning for information correlation in pervasive computing. Applied Artificial Intelligence **21**(4&5) (2007) 405–425
14. Bandini, S., Mosca, A., Palmonari, M.: Intelligent alarm correlation and abductive reasoning. Logic Journal of the IGPL **14**(2) (2006) 347–362
15. Bandini, S., Bogni, D., Manzoni, S., Mosca, A.: St-modal logic to correlate traffic alarms on italian highways: project overview and example installations. In: IEA/AIE'2005: Proceedings of the 18th international conference on Innovations in Applied Artificial Intelligence, London, UK, Springer-Verlag (2005) 819–828
16. Franceschet, M., de Rijke, M.: Model checking hybrid logics (with an application to semistructured data). J. Applied Logic **4**(3) (2006) 279–304
17. Clarke, E.M., Grumberg, O., Peled, D., eds.: Model Checking. MIT Press (2000)
18. Clarke, E.M., Schlingloff, B.H.: Model checking. In Robinson, J.A., Voronkov, A., eds.: Handbook of Automated Reasoning. Elsevier and MIT Press (2001) 1635–1790
19. Clarke, E.M.: The birth of model checking. [24] 1–26
20. Emerson, E.: The beginning of model checking: A personal perspective. (2008) 27–45
21. Cimatti, A., Clarke, E.M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: Nusmv 2: An opensource tool for symbolic model checking. In: CAV '02: Proceedings of the 14th International Conference on Computer Aided Verification, London, UK, Springer-Verlag (2002) 359–364
22. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns. Addison-Wesley Professional (January 1995)
23. Megiddo, N., Modha, D.S.: Arc: A self-tuning, low overhead replacement cache. In: FAST '03: Proceedings of the 2nd USENIX Conference on File and Storage Technologies, Berkeley, CA, USA, USENIX Association (2003) 115–130
24. Grumberg, O., Veith, H., eds.: 25 Years of Model Checking - History, Achievements, Perspectives. In Grumberg, O., Veith, H., eds.: 25 Years of Model Checking. Volume 5000 of Lecture Notes in Computer Science., Springer (2008)