Model checking software for phylogenetic trees using distribution and database methods

José Ignacio Requeno[1,*] and José Manuel Colom[1]

[1]Department of Computer Science and Systems Engineering (DIIS), Universidad de Zaragoza, C/ María de Luna 1, 50018 Zaragoza, Spain

Summary

Model checking, a generic and formal paradigm stemming from computer science based on temporal logics, has been proposed for the study of biological properties that emerge from the labeling of the states defined over the phylogenetic tree. This strategy allows us to use generic software tools already present in the industry. However, the performance of traditional model checking is penalized when scaling the system for large phylogenies. To this end, two strategies are presented here. The first one consists of partitioning the phylogenetic tree into a set of subgraphs each one representing a subproblem to be verified so as to speed up the computation time and distribute the memory consumption. The second strategy is based on uncoupling the information associated to each state of the phylogenetic tree (mainly, the DNA sequence) and exporting it to an external tool for the management of large information systems. The integration of all these approaches outperforms the results of monolithic model checking and helps us to execute the verification of properties in a real phylogenetic tree.

1 Introduction

A phylogenetic tree is a widely used description of the evolution process which is discovered through molecular sequencing data and morphological data matrices [1]. Both taxonomy and systematics make heavy use of empirical data, with proposition, verification and generalization of hypotheses over the reconstructed tree playing a central role in their application [2]. One of the main objectives is to extract and analyze the implicit biological messages in an inferred *true* tree of life [3]. For example, the detection of conserved regions and Single Nucleotide Polymorphisms (SNP's) in a clade, the presence of back mutations along a branch of the phylogeny or the analysis of covariation in closely-related taxa (for more examples, see [4, 5]).

This suggests the possibility of introducing a generic framework for heterogeneous hypothesis verification over a phylogenetic tree. We applied model checking techniques over dendrograms in our previous works in order to study properties that emerge from the biological labeling of the tree states [4]. Model checking is an automated generic verification technique stemming from computer science that is based on temporal logics. It has been successfully applied in industry for system modeling and verification [6]. Given a finite state model of a system and a formal property, model checking techniques systematically checks whether this property holds for (a given state in) that model. The model checking process consists of three phases: modeling both the system and properties with appropriate description languages, running the

*To whom correspondence should be addressed. Email: nrequeno@unizar.es

verification (checking the property validity with a model checking software) and analyzing the results (studying counterexamples to the property).

The system we desire to analyze is commonly abstracted by a set of discrete reachable states and the transitions that allow the movement from one state to another. In the context of phylogeny, these states and transitions can be naturally identified with the taxa and the speciation events of a phylogenetic tree. Thus, a phylogeny represents a *model* of a particular system endowing an evolutionary process in which biological properties expressed with formal logics can be *verified*. The properties mentioned above can be translated into a particular temporal logic and evaluated over the current phylogeny.

The model checking technique used in this paper should not be confused (or be identified) with the classic phylogenetic procedures focused on guaranteeing the validity of the inferred tree. In this last context, the validation of a phylogeny determines the goodness of fit of mutation models under different biological hypothesis. These biological models are based on the concept of evolutionary distance, allowing to explore tangible numerical relationships between sets of populations or individuals characterized by biological features such as DNA. This has led to tree-building through distance and character based methods [1, 7]. The validation of these models of DNA substitution can also be carried out by means of the proposed model checking techniques, but in this paper we will focus on our first approximation: the phylogenetic tree as a model over which we can claim properties.

As a prominent advantage, model checking allows us to uncouple software tools from the definition of properties and it also hides the underlying implementation technology. Besides, these properties can be exported and evaluated in other structures (i.e., trees or networks) so as to compare the results and define metrics. Nevertheless, the performance is penalized when scaling the system for large phylogenies and alignments [8]. The underlying problem of standard model checking is the great amount of phylogenetic data it has to deal with: the information associated to each node of the tree is strongly related to the DNA sequence of the specie (up to millions of nucleotides).

In order to cope with this problem, two strategies are presented here. The first one consists of partitioning the graph structure into a set of subgraphs each one representing a subproblem of verification so as to speed up the computation time and distribute the memory consumption. Two subtactics are considered here depending on the division method: the partition of the tree into subtrees, each one managed by a different model checker; or the slicing of the tree, each slice containing a copy of the original tree but only a portion of the DNA sequence. The techniques based on distributed model checking were presented in [9, 10]. In our previous work, we presented the novelty of sliced model checking as an adaptation to the context of phylogenetics [11].

The second strategy is based on uncoupling the DNA information of the phylogenetic tree and exporting the alignment to an external tool specialized in the management of large information systems, for example, a database. This leads to a light tree structure labeled with pointers to the elements of the database that can be efficiently manipulated by current model checking tools. The database strategy is firstly introduced here for the domain of phylogenetic model checking.

In this paper, we obtain the best of the two worlds. We integrate these methodologies (distribution and databases) and show that a combination of both strategies allows us to work with real phylogenies. The paper is organized as follows. After this introduction, Section 2 presents

the key concepts about phylogenetic model checking. Section 3 introduces the adaptation and implementation of distributed model checking techniques for the particular case of phylogenetic trees. Section 4 presents the notions of external databases for storing the DNA alignment. Section 5 summarizes how all these solutions can be integrated in a workflow that improves the performance of each solution in isolation. Finally, Section 6 briefs the conclusions.

2 Phylogenetic Model Checking

Phylogenetics and model checking can be bridged after reflecting some considerations about the processes of modeling and specification. Formally, a phylogenetic tree with root r is a rooted labeled tree where each vertex represents a population of compatible individuals described by their genome, and the vertices are arranged along the paths of the tree according to a evolutive process [4].

Definition 1 (Rooted Labeled Tree, [4]) *Let Σ be a finite alphabet and l a natural number. A phylogenetic tree over Σ^l is a tuple $P = (T, r, D)$, where:*

- *$T = (V, E)$ is a tree graph,*
- *$r \in V$ is its root, and*
- *$D : V \to \Sigma^l$ is a dictionary function that labels each vertex with its associated taxon sequence.*

At the same time, a Kripke structure stands for the behavior of a system through a graph and can be used to query properties [12]. It represents a finite transition system through a tuple $M = (S, S_0, R, L)$.

Definition 2 (Kripke Structure, [4]) *Let AP be a set of* atomic propositions*, i.e., boolean predicates that describe the observable properties of a state. A Kripke structure over AP is a finite transition system represented by a tuple $M = (S, S_0, R, L)$, where:*

- *S is a finite set of states,*
- *$S_0 \subseteq S$ is the set of initial states,*
- *$R \subseteq S \times S$ is a total transition relation between states, i.e., for every state $s \in S$, there exists $t \in S$ such that $(s, t) \in R$, and*
- *$L : S \to 2^{AP}$ is the labeling function that associates each state with the subset of atomic propositions that are true of it.*

A certain phylogenetic tree can be interpreted as a computation of the process of evolution. The set of atomic propositions represents states that stand for distinct populations of individuals sharing a common sequence.

Definition 3 (Branching-time Phylogeny, [4]) *A rooted labeled tree $P = (T, r, D)$ is univocally defined by the Kripke structure $M = (V, \{r\}, R, L)$, where:*

- *R is the transition relation composed of the set of tree edges (directed from r) plus self-loops on leaves: $R = E \cup \{(v, v) : \nexists\,(v, w) \in E \wedge v, w \in V\}$ and*
- *$AP = \bigcup_{i=1}^{l} AP_i$ is the set of atomic propositions, where $AP_i = \{seq[i] = \sigma \mid \sigma \in \Sigma\}$ with seq an array of variables with type Σ. We have a set of pairs (variable, value) such as $seq[i]$ is the variable of the string seq at the i-th position and σ is its character value, and*
- *$L : S \rightarrow AP_1 \times \ldots \times AP_l$, is the standard labeling function defined by AP, under which a state v mapped to $D(v) = seq$ with $seq = \sigma_1\sigma_2 \ldots \sigma_l$ satisfies the family of properties $seq[i] = \sigma_i, 1 \leq i \leq l$, plus a unique state identifier in case of several states share the same atomic propositions. That is, $L(s) = (seq[1] = \sigma_1, \ldots, seq[l] = \sigma_l)$.*

Definition 4 (Phylogenetic Tree Logic, [4]) *An arbitrary temporal logic formula ϕ is defined by the following minimal grammar, where $p \in AP$:*

$$\phi ::= p \mid \neg\phi \mid \phi \vee \phi \mid \mathbf{EX}(\phi) \mid \mathbf{EG}(\phi) \mid \mathbf{E}[\phi \mathbf{U} \phi] \tag{1}$$

Additionally, a *Phylogenetic Tree Logic* (PTL) is proposed as a *temporal logic* for the specification of evolutionary properties. PTL is an adaptation of *Computational Tree Logic* (CTL) [13] to the domain of bioinformatics. Formulas are checked against a model M considering all paths π that originate from a certain state s_0. For example, one of the most frequent basic phylogenetic property asks whether the sequences of a set of organisms exhibit a *back mutation* (for more examples, see [4, 5]). In terms of PTL, it looks for a node (taxon) somewhere in the tree where a mutation appeared ($\mathbf{EF}(seq[col] \neq \sigma)$) but is reverted in the future ($\mathbf{EF}(seq[col] = \sigma)$):

$$BM(col, \sigma) \equiv (seq[col] = \sigma) \wedge \mathbf{EF}[(seq[col] \neq \sigma) \wedge \mathbf{EF}(seq[col] = \sigma)] \tag{2}$$

Current model checking tools load the model and the temporal logic formulas, evaluate the truth value of them over the tree and compute counterexamples representing the states for which the formula is false [14]. Moreover, evaluation results may reveal new meaningful information that can be reused in subsequent refinements of the phylogenetic tree.

Verification of temporal formulas is formalized under the framework of set theory. We follow this convention throughout the paper, that is classic in the context of verification [13]. In fact, the traditional model checking algorithm is usually presented as a recursive function which computes the set of states in a Kripke structure which satisfy a PTL formula (Algorithm 1). The foundations of this algorithm and how it works is extensively explained in the Theorem 6.23 of [13] (page 343), where it is also possible to find detailed examples. The reader only must translate our path operators and quantifiers to the notation presented there.

In order to evaluate the temporal operators $\mathbf{EG}(\psi)$ and $\mathbf{E}[\psi_1 \mathbf{U} \psi_2]$, the greatest and least fixpoints are computed, respectively. Both fixpoint sets can be obtained as the result of a breadth-first search (Algorithm 2). In particular, the call $\mathit{fixpoint}(M, Sat(M, \psi), \emptyset, S)$ produces the greatest fixpoint, and $\mathit{fixpoint}(M, Sat(M, \psi_2), Sat(M, \psi_1), \emptyset)$ returns the least fixpoint.

Algorithm 1 Algorithm $Sat\,(M, \phi)$

Require: $M = (S, S_0, R, L)$ is a Kripke structure and ϕ is a PTL formula
Ensure: A subset of states of S that satisfies ϕ
  if $\phi \equiv \top$ **return** S {Set of states from the Kripke structure M}
  else if $\phi \equiv p \in AP$ **return** $\{s : p \in L\,(s)\}$
  else if $\phi \equiv \neg\psi$ **return** $S \setminus Sat\,(M, \psi)$
  else if $\phi \equiv \psi_1 \vee \psi_2$ **return** $Sat\,(M, \psi_1) \cup Sat\,(M, \psi_2)$
  else if $\phi \equiv \mathbf{EX}\,(\psi)$ **return** $\{s : (s, s') \in R, s' \in Sat\,(M, \psi)\}$
  else if $\phi \equiv \mathbf{EG}\,(\psi)$ **return** $fixpoint\,(M, Sat\,(M, \psi)\,, \emptyset, S)$
  else if $\phi \equiv \mathbf{E}\,[\psi_1 \mathbf{U} \psi_2]$ **return** $fixpoint\,(M, Sat\,(M, \psi_2)\,, Sat\,(M, \psi_1)\,, \emptyset)$
  end if

Algorithm 2 Algorithm $fixpoint\,(M, Sat\,(M, \phi)\,, Sat\,(M, \psi)\,, Init)$

Require: $M = (S, S_0, R, L)$ is a Kripke structure, ϕ and ψ are PTL formulas
Require: $Sat\,(M, \psi)$ and $Init$ are the sets of initial states (returned by calls to Sat algorithm) and $Sat\,(M, \phi)$ is the set of final states
Ensure: A set of states that represents paths going from $Init$ (or $Sat\,(M, \psi)$) to $Sat\,(M, \phi)$
  $New \leftarrow Init$
  repeat
    $Old \leftarrow New$
    $New \leftarrow Sat\,(M, \psi) \cup (Sat\,(M, \phi) \cap \{s : (s, s') \in R, s' \in Old\})$
  until $New = Old$
  return New

3 Partitioned Model checking

The division of the Kripke structure in subtrees enables the parallelism in the computation of paths, while sliced model checking helps to distribute the computation of states satisfying a propositional formula. In this sense, sliced model checking and the division of trees in subtrees are two complementary techniques that can be applied together for distributing the computation and improving the performance and scalability of our system for verifying PTL formulas.

3.1 Division in Subtrees

Memory usage is a major limiting factor in the verification of complex systems. For a long time it has been possible to perform model checking on systems with states in excess of 10^{120} [13]. Symbolic model checking is perhaps one of the most common methods to achieve this [15], together with general-purpose memory saving techniques such as abstractions, partial reductions or symmetries [10, 12].

One of the first attempts to infer global properties of the system through an incremental bottom-up strategy was compositional reasoning [16]. It attempts to structure the system in components and progressively verify local properties for each part (e.g., in a genomic sequence, genes could be considered its components). For example, in the assume-guarantee paradigm [17] the method establishes assumptions about the environment of each component and uses these to

aid in the verification of each property. Alternative approaches focused on the decomposition of temporal logic formulas are exemplified by [18].

Recently, model checking methods based on distributed Kripke structures attempt to improve the performance by partitioning the graph into smaller subgraphs and distributing the chunks among available computing units, both the storage of the partial Kripke structure and computation of satisfiability of logic formulas [9, 10, 19]. These methods attack the size of the structure (number of states) and not the complexity of each state, which is the other limiting factor in phylogenetic model checking.

In the particular case of trees, the verification and communication process between chunks is simplified due to their inherent acyclicness. Given the tree root, $S_0 = \{s_0\}$, we verify the property for each of the direct subtrees and operate with the boolean results according to the logical quantifiers. In the case of a generic formula ϕ written in PTL, the equivalence is supported by the following recursive expansion law of the path operators ($\mathbf{EX}$, $\mathbf{EG}$, $\mathbf{E}[\_\mathbf{U}\_]$):

$$M, s_0 \models \mathbf{EX}(\phi) \equiv \bigvee_{s_i \in successors(s_0)} M_i, s_i \models \phi$$

$$M, s_0 \models \mathbf{EG}(\phi) \equiv M_0, s_0 \models \phi \wedge (\bigvee_{s_i \in successors(s_0)} M_i, s_i \models \mathbf{EG}(\phi))$$

$$\begin{aligned} M, s_0 \models \mathbf{E}(\phi_1 \mathbf{U} \phi_2) \equiv M_0, s_0 &\models \phi_2 \vee \\ (M_0, s_0 &\models \phi_1 \wedge (\bigvee_{s_i \in successors(s_0)} M_i, s_i \models \mathbf{E}[\phi_1 \mathbf{U} \phi_2])) \end{aligned}$$

where M is the original tree with root s_0. $M_i = (S^i, \{s_i\}, R^i, L)$ is the subtree with root $s_i \in successors(s_0)$. The set of states S and the reachability relation R are covered by S^i and R^i respectively. The degenerated tree $M_0 = (S^0, \{s_0\}, R^0, L)$ only needs the node s_0 and the direct successors for the synchronization. That is, $\bigcap_{i=1}^{n} S^i = \emptyset$, $\bigcap_{i=1}^{n} R^i = \emptyset$ with $S^0 = \{s_0\} \cup succesors(s_0)$ and $R^0 = \{(s_0, s_i) : s_i \in successors\,(s_0)\}$.

The formula can be unfolded indefinitely by means of $\mathbf{EX}$. In this case, the set of successors s_i defines a border at a certain depth of the original tree: we must ensure that ϕ holds in s_0, in at least a subtree rooted by s_i and in a path between s_0 and s_i. The root s_i acts as an interface node for the communication process during the composition of the partial results. The Figure 1 illustrates this technique.

Our verification system requires a scheduler that maps the verification task of each subtree over the set of available CPU's. The depth of the recursion in the formula expansion (number and size of the subtrees), as well as the appearance of short circuits during the composition of results will determine the performance of this approach. The division in subtrees is a recursive process that must be optimized for every particular case. The detection of clades with conserved regions or characteristic SNP's are the kind of properties that benefit of this approach.

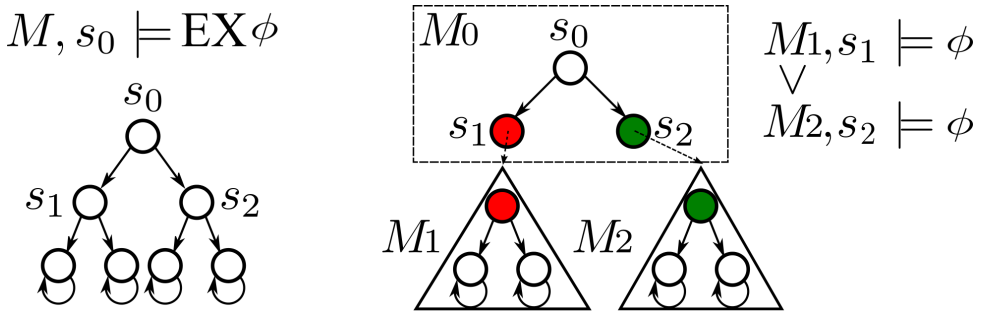

Figure 1: Distributed verification of EXϕ through the parallel execution of ϕ in the direct subtrees.

3.2 Slicing the State

Phylogenetic model checking usually associates a complete DNA sequence per node of the tree. Sometimes the storage in local memory of the phylogenetic tree together with the atomic propositions leads to a high memory consumption in the model checker tool. In order to solve it, the state slicing (also called sliced model checking) focuses on the state complexity of the Kripke structure by creating several copies of the original phylogenetic tree and verifying the subproperties in parallel, each slice labeled with a partial substring of the DNA [11].

The state slicing presented in this section can be efficiently applied to the verification of the back mutation property (Eq. 2). Each slice verifies a subformula referred to a part of the DNA sequence that is the only information stored in each state of the slice. These verifications can be done independently and so a computational speed up proportional to the number of slices can be obtained. That is, given an alignment of length l, each sequence is distributed in up to l independent slices where $\bigvee_{col \in \{1 \ldots l\}} M, s_0 \models BM\,(col, \sigma)$ are verified asynchronously. The detection of back mutations is a degenerated example of this technique because it only needs a tree labeled with a single nucleotide.

By now, we will focus on the set of atomic propositions AP. In Def. 3, the labeling function L tags each state of the Kripke structure with an unique tuple composed of several atomic propositions. Note that the Cartesian product in the labeling function involves an implicit *and* operator; that is, the label states that a temporal formula is valid on the current state iff it asserts $[(seq[1] = \sigma_1) \wedge \ldots \wedge (seq[l] = \sigma_l)]$. Multi-labels would be or-like though.

In opposition to traditional distributed model checking algorithms, which divide the system into disjoint Kripke substructures, we present a new slicing criterion. The notion of AP *normalization* in a Kripke structure motivates the definition of a projection (reading) function that takes (cuts) a subset of the variables from the AP structure of state labels.

Definition 5 (Projection Function) *A projection function with subindex $i \in \{1, \ldots, l\}$ over AP selects a subset of elements such that:*

- $proj_i : AP \rightarrow AP_i$ *with* $i \geq 1$ *such that* $proj_i(seq) = (seq[i] = \sigma)$.
- $proj_I : AP \rightarrow AP_{i_1} \times \ldots \times AP_{i_m}$ *such that* $proj_I(seq) = (seq[i_1] = \sigma_{i_1}, \ldots, seq[i_m] = \sigma_{i_m})$ *is a projection function with a set of indices* $I = \{i_1, i_2, \ldots, i_m\}$, $m \leq l$

The projection operator allows to slice the nodes of the Kripke structure instead of splitting the graph into regions. We obtain several lighter copies of the original tree, where each slice of the AP set represents some characteristics of the initial system.

Definition 6 (Sliced Kripke Structure) *Let $\{I_i \mid 1 \leq i \leq n, I_i \subseteq I\}$ be a partition of the set of indices $I = \{1, \ldots, l\}$. We say that $M = (S, S_0, R, L)$ can be obtained from the composition of n Kripke structures, named slices of M, each one defined as*

- *$M_i = (S, S_0, R, L_i)$, $\forall i \in \{1, \ldots, n\}$, and*
- *$L_i : S \to proj_{I_i}(AP)$, where $L(s) = L_1(s) \cdot L_2(s) \cdot \ldots \cdot L_n(s)$, $\forall s \in S$, with " $\cdot$ " the operator function that concatenates tuples.*

Take as example a generic phylogenetic property $\phi(p_1, p_2, \ldots, p_l)$ represented by a temporal logic, with $p_i \in AP_i = \{seq[i] = \sigma | \sigma \in \Sigma\}$ an atomic proposition that asks for the tree nodes having the nucleotide σ in the *i*-th position of the DNA sequence (Def. 3, [11]). The classic model checking algorithm parses the formula and starts verifying the inner subformulas first. In the derivation tree from the PTL grammar for the given formula, we reach a propositional operator at some point of the recursion (boxed lines in Algorithm 3).

The verification of formulas with propositional operators, such as $\psi_1 \vee \psi_2$, begins with the computation of the satisfiability sets $Sat(M, \psi_j)$, $j = \{1, 2\}$. These sets allow us to distribute the computation in parallel, as long as we compose the partial results with a synchronized union. The support of ψ_j is $\|\psi_j\| = \{p_i \in AP_i | p_i$ or $\neg p_i$ appears in $\psi_j\}$. The verification of ψ_j is mapped to a remote model checking tool that has a copy of the phylogenetic tree labeled with the slice $R_I = \bigcup_{i \in I} AP_i$, with $I \subseteq \{1 \ldots l\}$ a set of index, $l = length(DNA)$ and $AP = \bigcup_{i=1}^{l} AP_i$. The remote model checker also executes the Algorithm 3 but with the set of atomic propositions belonging to $\|\psi_j\| \subseteq R_{I_j}$. In order to obtain a perfect distribution, $\bigcap_{j=\{1,2\}} \|\psi_j\| = \emptyset$. The notation $par(R_{I_j}, Sat(M, \psi_j))$ means that $Sat(M, \psi_j)$ is computed in parallel in the remote model checker R_{I_j} associated to the slice containing $\|\psi_j\|$. L_{R_I} is the labeling function of the states of that slice.

Algorithm 3 Algorithm $Sat\,(M, \phi)$

Require: $M = (S, S_0, R, L)$ is a Kripke structure; $\phi(p_1, p_2, \ldots, p_l)$ is a PTL formula
Ensure: A subset of states of S that satisfies $\phi(p_1, p_2, \ldots, p_l)$
if $\phi \equiv \top$ **return** S {Set of states from the Kripke structure M}
else if $\boxed{\phi \equiv p_i \in AP}$ **return** $par(R_I, \{s : p_i \in L_{R_I}(s)\})$ with $i \in I$
else if $\boxed{\phi \equiv \neg\psi}$ **return** $S \setminus par(R_I, Sat(M, \psi))$
else if $\boxed{\phi \equiv \psi_1 \vee \psi_2}$ **return** $par(R_{I_1}, Sat(M, \psi_1)) \cup par(R_{I_2}, Sat(M, \psi_2))$
else if $\phi \equiv \mathbf{EX}\,(\psi)$ **return** $\{s : (s, s') \in R, s' \in Sat\,(M, \psi)\}$
else if $\phi \equiv \mathbf{EG}\,(\psi)$ **return** $fixpoint\,(M, Sat\,(M, \psi), \emptyset, S)$
else if $\phi \equiv \mathbf{E}\,[\psi_1 \mathbf{U} \psi_2]$
return $fixpoint(M, \boxed{par(R_{I_2}, Sat(M, \psi_2))}, \boxed{par(R_{I_1}, Sat(M, \psi_1))}, \emptyset)$
end if

We consider the access to the atomic propositions transparent to the underlying technology. However, we must advance that this is usually the most time-consuming part during the experimentations in Section 5. This fact motivates the introduction of information systems optimized for the management of huge amounts of phylogenetic data in the next section.

Finally, the size of the slices depends on the target DNA regions we desire to analyze (i.e., single nucleotides, genes or chromosomes), the kind of properties we want to verify and the hardware requirements we have. A high number of slices will provide of a better level of parallelism and low hardware requirements (CPU, memory), but it will be limited by the potential appearance of bottlenecks during the composition of results.

4 Database Model Checking

The second step of our approach consists of uncoupling the atomic propositions from the model checker. The use of external databases as a repository of biological sequences alleviates the memory explosion problem when the local storage of trees with partial DNA is not enough. Moreover, the database manager simplifies the interface to the DNA data because it usually allows concurrent queries and it hides the internal synchronization and data structures. In a general sense, trees are labeled with pointers to DNA sequences stored in an external server.

By default, we use BioSQL, a shared database schema for storing sequence data in SQL servers that is supported by several script languages [20]. The DNA alignment is stored in a single table with a row per sequence. Each row has two important fields: an identifier (i.e., the GenBank accession [21]) plus the plain string of nucleotides.

An interface script is focused on masking the access to the alignment via the atomic propositions. This script translates the phylogenetic properties expressed in terms of sequences and nucleotides into phylogenetic properties written in terms of taxon identifiers. For example, the following formula reinterprets the back mutation property (Eq. 2):

$$BM\,(id) \equiv (id \in seq_{i\sigma}) \wedge \mathbf{EF}\,[id \notin seq_{i\sigma} \wedge \mathbf{EF}\,(id \in seq_{i\sigma})] \tag{3}$$

where id is the identifier of the initial state and $seq_{i\sigma} = \{id_1, \ldots, id_n\}$ contains the identifiers of the set of states satisfying the property $seq[i] = \sigma$ in the external database. The states of the phylogenetic tree are labeled with these identifiers.

In fact, the database tables can be seen as matrices that we can slice by row (number of taxons) or by column (divide the DNA in substrings). These subtables can be stored or replicated in separated servers or cluster nodes so as to allow parallel access to the DNA data and to improve the communication bandwidth between the database and the model checker. In addition, recent versions of SQL servers support multi-core CPU's, which improves the time response when attacking the server with several queries. This approach allows scaling in memory and speeds up the system.

Finally, relational databases cannot only store phylogenetic data, but also partial verification results. Furthermore, relational databases can execute the model checking algorithms for PTL formulas [22]. The main advantage of this topic is that we avoid the exportation of DNA data

from the database and consequently the bandwidth bottleneck. An evaluation of SQL model checkers for phylogenetic data will comprise our future work.

5 Workflow

We compose our framework by integrating the different modules described in previous sections. In this part we detail the characteristics of our workflow and the performance results with large phylogenetic data. We must remark that without the implementation of all these optimizations, current model checking tools couldn't manage big phylogenies efficiently.

5.1 Description

Figure 2 represents a graphical description of our workflow. The central core of our approach is the model checker package NuSMV v2.5.4 [23], which is a well-known public software. It is surrounded by a set of modules and tools that accommodate the phylogenetic data. Our framework is divided in three important modules. First of all, the *loader* consists of a BioPython script that creates and initializes a local MySQL server v5.5.24-7 with the DNA alignment. It is executed only once during the initialization phase and the database remains constant for the rest of the verification process.

Secondly, the *property transformer* is a BioPython script that pre-processes the phylogenetic formulas. Every time the user asks for the states satisfying a pattern $seq[i] = \sigma$ in the DNA, the script retrieves the set of identifiers of those species that satisfy the formula in the database. The script is optimized for sending concurrent queries to the database in order to take advantage of current multi-core servers. As result, it outputs several temporal files with the original phylogenetic properties expressed in terms of Genbank identifiers. These files correspond to the projections of the formula in the sliced model checking methodology. Each projection computes the set of identifiers satisfying an atomic proposition.

Next, the *NuSMV transformer* is a BioPerl script that translates the phylogenetic tree into the NuSMV syntax and labels the states with the associated identifiers of the database. The NuSMV input file is equivalent to that defined for Cadence SMV in [8] except for the inclusion of DNA sequences. The script can be extended in order to divide the original tree in multiple subtrees as a part of the implementation of partitioned model checking. For each subtree, the system would execute in parallel the verification of the phylogenetic formulas.

Once the phylogenetic tree is translated into the model checker syntax and the properties are translated and projected according to the slicing criterion, the workflow starts the verification process. To this end, we launch a set of independent instances of NuSMV. Each task works with a DNA projection and a copy of the Kripke structure. At the end of the verification process, a multi-threading version of the model checker collects the partial results and reconstructs the PTL paths. Depending on the number of formulas and slices, we obtain a higher speed up from the distribution. The results of the verification process offer an important feedback for the refinement of the original tree and the biological assumptions.

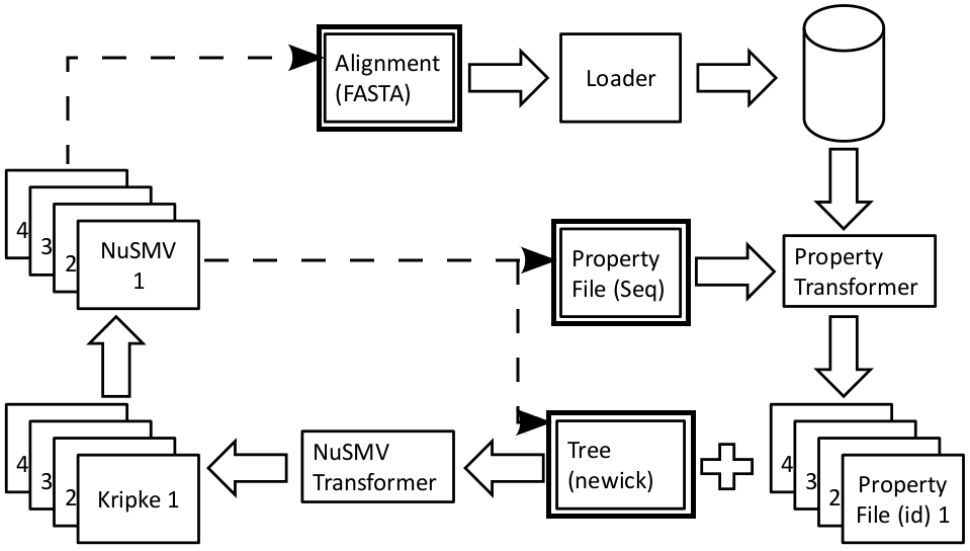

Figure 2: Workflow diagram with the alignment, phylogenetic tree and properties as input.

5.2 Experimental Results

The data set of our experimentation comprises the ZARAMIT tree [24], which has been reconstructed from 7390 sequences of Human mitochondrial DNA with $16,569$ base pairs. In total, the number of internal plus terminal nodes is 14512. Table 1 shows that our architecture outperforms previous works in [8]: we spend less time and memory resources for the initialization of the model checking tool with the ZARAMIT tree than for the initialization of a protein tree with 2000 tips and sequences of 500 aminoacids.

More in detail, Table 1 represents the time needed for the initialization of the NuSMV model checker with respect to the total number of states in the Kripke structure. The experiment takes as input a phylogenetic tree labeled with GenBank identifiers. If the tree is balanced, the initialization and verification time using division in subtrees is faster than attacking the original tree because the trend is quadratic with respect to the number of nodes. Thanks to the integration with databases, the memory consumption of the model checker is reduced: the representation of the ZARAMIT tree consumes around 50 MB.

Table 1: (S)econds needed for the initialization of NuSMV w.r.t the number of tree (N)odes

N	165	435	631	785	1016	1365	1697	2136	5280	7602	13141	13444	14512
S	0.012	0.06	0.112	0.156	0.24	0.444	0.668	1.012	7.876	15.361	48.679	52.643	57.82

From the point of view of performance, the database is the most important point of our workflow because it connects the model checker with the DNA sequences. The database server is executed in a desktop workstation (AMD Opteron @3GHz, 4GB RAM, Debian Linux). Due to the alignment size (around 350MB for 7390 leafs plus ancestors) and the hardware available, we work with a single instance of MySQL server. The database loader spends less than 2 minutes at the initialization point, but the database remains constant for all the verifications. The retrieval of atomic propositions lasts 7.2s for 25 concurrent SQL queries (44.6s if we serialize them).

A second desktop workstation (Intel Core 2 Duo E6750 @ 2.66GHz, 8GB RAM, Debian Linux)

is devoted to the execution of the slices in the model checking workflow. As we remarked previously, we detected that the internal representation of the Kripke structure in NuSMV penalizes the access to the atomic propositions during the model checking process. Given a PTL formula expressed in terms of taxon identifiers, the time required for the selection of a set of states represented by an atomic proposition raises up to 1m 20s in each projected slice (around 25s if we discount the initialization costs of Table 1). Every slice is executed in parallel. Conversely, the integration of partial results and the computation of PTL paths for the verification of a back mutation property only lasts 1m 2s in total (5-10s if we exclude the initialization costs).

In sum, the manipulation of big sets represented by atomic propositions lasts around 25s, while the initialization costs of NuSMV raises up to 55s-1m in phylogenetic trees with thousands of species. Thus, we must parallel the computation of four or more slices in order to obtain a clear speed up. Longer unexplored phylogenetic properties can take advantage of multi-threading as the PTL verification cost depends of the formula length [13].

6 Conclusions

The notion of the phylogenetic tree as an evolution process is captured by the states of a transition system, which reflect a specific step of the speciation process, and the transitions themselves, which describe mutations and reproduction events. The introduction of model checking techniques for the automated verification of phylogenetic properties expressed as temporal logic formulas over phylogenetic trees, interpreted as Kripke structures, is presented in our previous works.

However, the use of monolithic model checking for the verification of huge phylogenetic data is unfeasible because of the high requirements in time and memory. In this paper we presented the adaptation and integration of two techniques that help us to scale up our framework: distributed model checking, that divides the Kripke structure in subtrees and slices the nodes in portions of DNA; and the use of external databases.

The particularities of trees with respect to more generic graph structures facilitates the implementation, synchronization and composition of results of distributed structures in phylogenetics. The division of the verification process into subtasks that check the phylogenetic properties in subtrees improved the performance of our system because the initialization time in the NuSMV model checker depends quadratically with the number of nodes.

Next, the introduction of sliced model checking has reduced the memory consumption by storing fragments of the DNA in the local memory of the model checker or the complete DNA alignment in an external persistent database. In particular, relational databases allowed us to map sequences and nucleotides with atomic propositions. Furthermore, relational databases can emulate the model checking algorithms using PL-SQL. In conjunction with a multi-threading parallel algorithm, we have provided an efficient way of verifying complex temporal logic formulas.

Finally, we have shown here that the integration of all these approaches helped us to execute the verification of properties in a real phylogenetic tree. For example, the evaluation of a back mutation property using model checking over the ZARAMIT tree has outperformed the performance results of monolithic model checking in [8]. The use of our workflow demonstrates that

we have substantially improved the power of model checking techniques for the manipulation of huge systems. Hence, it would be impossible to handle label-intensive systems efficiently in centralized model checking tools without the use of this approximation.

Our future work aims to continue with the improvement of the performance in the model checking framework presented here. For example, one possible optimization in this direction is the implementation and evaluation of model checking algorithms in the native languages of databases. In addition, we need to classify the classic phylogenetic properties according to their structure. The algorithms for partitioning the tree into subtrees strongly depends on the path operators and quantifiers of the temporal formula to be analyzed. Similarly, the algorithm for slicing the state has influence on the final performance of the verification. The sliced algorithms also depend on the structure of the formula although in a different way to the case of the partition of the tree: the formula determines the set of labels of interest. Henceforth, we need to analyze the potential speed up obtained by the evaluation of each kind of phylogenetic property and the compatibility with the distribution and database methodologies summarized in this paper. Finally, we would like to extend this framework to the evaluation of probabilistic and quantitative properties in phylogenies using adapted logics.

Acknowledgements

This work was supported by the Spanish Ministry of Science and Innovation (MICINN) [Project TIN2011-27479-C04-01] and the Government of Aragon [B117/10].

References

[1] Z. Yang and B. Rannala. Molecular phylogenetics: principles and practice. *Nature Reviews Genetics*, 13(5):303–314, 2012.

[2] A. Ø. Mooers and S. B. Heard. Inferring evolutionary process from phylogenetic tree shape. *The Quarterly Review of Biology*, 72(1):31–54, 1997.

[3] W. M. Fitch. Uses for Evolutionary Trees. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 349(1327):93–102, 1995.

[4] R. Blanco, G. de Miguel Casado, J. I. Requeno and J. M. Colom. Temporal logics for phylogenetic analysis via model checking. In *IEEE International Conference on Bioinformatics and Biomedicine Workshops*, pages 152–157. IEEE, 2010.

[5] J. I. Requeno, G. de Miguel Casado, R. Blanco and J. M. Colom. Classification and representation of phylogenetic properties using temporal logics and model checking. Technical Report RR-0187-2012, Computer Science and Systems' Engineering Department, University of Zaragoza, 2012.

[6] O. Grumberg and H. Veith (editors). *25 years of model checking: history, achievements, perspectives*. Springer, Berlin, 2008.

[7] J. Felsenstein. *Inferring phylogenies*. Sinauer, Sunderland, MA, 2003.

[8] J. I. Requeno, R. Blanco, G. de Miguel Casado and J. M. Colom. Phylogenetic Analysis Using an SMV Tool. In *5th International Conference on Practical Applications of Computational Biology & Bioinformatics*, pages 167–174. Springer, 2011.

[9] M. C. Boukala and L. Petrucci. Distributed CTL Model-Checking and counterexample search. In *3rd International Workshop on Verification and Evaluation of Computer and Communication Systems*. 2009.

[10] D. Bošnački and S. Edelkamp. Model checking software: on some new waves and some evergreens. *International Journal on Software Tools for Technology Transfer*, 12(2):89–95, 2010.

[11] J. I. Requeno, R. Blanco, G. de Miguel Casado and J. M. Colom. Sliced Model Checking for Phylogenetic Analysis. In *6th International Conference on Practical Applications of Computational Biology & Bioinformatics*, pages 95–103. Springer, 2012.

[12] E. M. Clarke, J. O. Grumberg and D. A. Peled. *Model checking*. MIT Press, Cambridge, 2000.

[13] C. Baier and J.-P. Katoen. *Principles of model checking*. MIT Press, Cambridge, 2008.

[14] A. Donaldson and D. Parker (editors). *Proceedings of the 19th International SPIN Workshop on Model Checking of Software*, volume 7385 of *LNCS*. Springer, 2012.

[15] J. R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan and D. L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(4):401–424, 1994.

[16] E. M. Clarke, D. E. Long and K. L. McMillan. Compositional model checking. In *Logic in Computer Science, 1989. LICS'89, Proceedings., Fourth Annual Symposium on*, pages 353–362. IEEE, 1989.

[17] A. Pnueli. *In transition from global to modular temporal reasoning about programs*. Springer, 1985.

[18] D. M. Gabbay and A. Pnueli. A sound and complete deductive system for CTL* verification. *Logic Journal of IGPL*, 16(6):499–536, 2008.

[19] C. P. Inggs and H. Barringer. CTL* model checking on a shared-memory architecture. *Formal Methods in System Design*, 29(2):135–155, 2006.

[20] H. Mangalam. The Bio* toolkits - a brief overview. *Briefings in Bioinformatics*, 3(3):296–302, 2002.

[21] D. A. Benson, I. Karsch Mizrachi, K. Clark, D. J. Lipman, J. Ostell and E. W. Sayers. GenBank. *Nucleic Acids Research*, 40(D1):D48–D53, 2012.

[22] G. Shegalov. CTL Model Checking in Database Cloud. Oracle Corporation.

[23] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani and A. Tacchella. NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In E. Brinksma and K. Larsen (editors), *Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 359–364. Springer Berlin Heidelberg, 2002.

[24] R. Blanco, E. Mayordomo, J. Montoya and E. Ruiz Pesini. Rebooting the human mitochondrial phylogeny: an automated and scalable methodology with expert knowledge. *BMC Bioinformatics*, 12(1):174, 2011.