Phylogenetic Analysis Using an SMV Tool

José Ignacio Requeno, Roberto Blanco, Gregorio de Miguel Casado and José Manuel Colom

Abstract The need for general methods to verify biological properties in phylogenetics motivates research in formal frameworks so that biologists can focus their efforts exclusively in evolution modeling and property specification. Model checking is proposed to this end. Three pillars found this approach: modeling evolution dynamics as transition systems; specifying phylogenetic properties using temporal logic formulae; and verifying the former by means of automated computer tools. As prominent advantages for studying biological properties under our approach, different models of evolution can be considered, complex properties can be specified as the logical composition of others, and the refinement of unfulfilled properties as well as the discovery of new ones can be undertaken by exploiting the results of verification. Preliminary experimental results using the Cadence Symbolic Model Verifier support the feasibility of the methodology.

1 Introduction

Phylogenetic trees are useful abstractions for modeling and evaluating hypotheses about the evolution of life [5], as well as studying biological properties (e.g., [8]). However, the inherent temporal nature of phylogenetic data suggests the possibility of introducing novel formal methods capable of improving the lack of flexibility of conventional models and providing more ambitious features such as future prediction, embedding of information from the past and combination of evolutionary rules across heterogeneous levels of abstraction. In this regard, we exploit the features of model checking, a paradigm stemming from computer science based on temporal logics, which has been recently proposed for phylogenetic analysis in [3].

Department of Computer Science and Systems Engineering (DIIS)/Aragon Institute of Engineering Research (I3A), Universidad de Zaragoza, C/ María de Luna 1, 50018 Zaragoza, Spain, e-mail: {nrequeno, robertob, gmiguel, jm}@unizar.es

Model checking is an automated verification technique that, given a finite state model of a system and a formal property, systematically checks whether this property holds for (a given state in) that model. The model checking process consists of three phases: modeling with a description language (formalize both system and properties), running (check property validity with a model checking computer package) and analyzing the results (study the counterexamples if a property is not satisfied).

The aim of this paper is to illustrate the key steps for characterizing phylogenetic trees and biological properties under the scope of a Symbolic Model Verifier (SMV) tool in order to obtain performance criteria about the feasibility of our approach. The paper is arranged in five sections. After this introduction, Sect. 2 introduces the foundational roots which bridge model checking and phylogenetic analysis: phylogenies as a dynamic models of evolution, phylogenetic specifications as temporal logic formulae, and automated system verification. Next, Sect. 3 details the implementation of the phylogenetic tree in a particular model checker (Cadence SMV) and Sect. 4 shows the performance results obtained. Finally, Sect. 5 gathers the conclusions drawn from this research and outlines future work.

2 Foundations of the Approach

In principle, phylogenetics and model checking are two worlds which do not seem to have much in common. However, they can be bridged after reflecting on some considerations about the processes of modeling and specification. Following the approach presented in [3], we will now introduce the basic concepts underlying our proposal.

Generally speaking, phylogenies represent the history of the evolution of certain living organisms. In this context, we will focus our attention on phylogenetic trees, a widespread type of graph for modeling common phylogenies. Formally:

Definition 1 (Rooted Labeled Tree). Let Σ be a finite alphabet and l a natural number. A phylogenetic tree over Σ^l can be represented as 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 of the tree with its associated taxon sequence.

Using this approach, each vertex typically represents a population of genetically compatible individuals (i.e., they share a common heritage reflected in their DNA) who mate among themselves; formally, a *state*. Spontaneous transformative processes (mutations) modify the heritable information of individuals who eventually become founders of new populations; this process of speciation takes the form of oriented *transitions* between parent and child states. The graph is, in fact, a *transition system* [1, Def. 2.1].

From the point of view of the model checking theory we claim that it is possible to model and verify evolutionary systems in a natural way. To this end, an appropriate data structure for the representation of transition systems becomes necessary.

Definition 2 (Kripke Structure). Let AP be a set of *atomic propositions*. 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; 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 Kripke structure models a system that is capable of an infinite number of behaviors or *paths*, infinite sequences of successive states $\pi = s_0 s_1 s_2 \ldots$ such that $s_0 \in S_0$ and $(s_i, s_{i+1}) \in R, i \in \mathbb{N}$. The set of possible executions (paths) in a structure can be unfolded into its *computation tree*.

Here we are interested in the construction of Kripke structures that are equivalent to a certain phylogenetic tree, interpreted as a computation of the process of evolution. As we have discussed in [3], seamless conversion from tree to Kripke structure demands careful examination of the following points. On the one hand, the set of atomic propositions must be such that all sequences can be represented by it, and distinct populations that share a common sequence (at least as far as the tree is concerned) distinguished as separate states. On the other, representation of phylogenies (which are finite by definition) by means of the infinite computation trees that result from any Kripke structure under its standard semantics must be contemplated.

Therefore, we can define a suitable *branching-time structure*, which will form the basis for the interpretation of temporal logic formulae that express properties of the trees. The identification between phylogenetic trees (Def. 1) and transition systems (Def. 2) is thus fulfilled.

Definition 3 (Branching-time Phylogeny). A tree (per Def. 1) $P = (T, r, D)$ is univocally defined by the Kripke structure $M_P = (V, \{r\}, R_P, L_P)$, where:

- R_P is the transition relation composed of the set of tree edges (directed from r) plus self-loops on leaves: $R_P = E \cup \{(v,v) : \nexists (v,w) \in E \wedge v, w \in V\}$, and
- L_P is the standard labeling function defined by AP_P, under which a state v mapped to $D(v) = \sigma_1 \sigma_2 \ldots \sigma_l$ satisfies the family of properties $s[i] = \sigma_i, 1 \leq i \leq l$, plus any others necessary to preserve the unique logical identity of the state.

Essentially, the Kripke structure reflects the parent-child relations in the original tree and adds self-loops in terminal nodes, so that the computation tree is the same as the original one except for the infinite loops that occur at the leaves and the explicit definition of unique state identifiers.

Finally, *temporal logics* are formal systems that allow the representation and manipulation of logical propositions qualified in terms of time [6]. In the context of transition systems, they are used to define properties on sequences of transitions between states of a system through a convenient abstraction of it (in the present case, a specific type of Kripke structure). These work as specification languages for phylogenetic properties once an adequate logic is identified. Evolutionary processes are essentially branching in nature, and therefore *branching-time logics* (as opposed to linear-time logics) are especially suitable to their description. In particular, *Computational Tree Logic* (CTL) is powerful and widely used in the model checking community [4], and will be used throughout the paper. A complete grammar and

semantics of CTL formulae can be defined from a small subset of representative logical operators as follows.

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

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

Formulae are checked against a structure M considering all paths π that originate from a certain state s_0. $M, s_0 \vDash \phi$ signifies that s_0 satisfies ϕ. The semantics for verification of well-formed formulae is as follows (let $\pi = s_0 s_1 s_2 \ldots$).

- $M, s_0 \vDash p \Leftrightarrow p \in L(s_0)$,
- $M, s_0 \vDash \neg\phi \Leftrightarrow M, s_0 \nvDash \phi$,
- $M, s_0 \vDash \phi \vee \psi \Leftrightarrow M, s_0 \vDash \phi$ or $M, s_0 \vDash \psi$,
- $M, s_0 \vDash \mathbf{EX}(\phi) \Leftrightarrow \exists \pi : M, s_1 \vDash \phi$,
- $M, s_0 \vDash \mathbf{EG}(\phi) \Leftrightarrow \exists \pi : M, s_i \vDash \phi, \forall i \in \mathbb{N}$, and
- $M, s_0 \vDash \mathbf{E}[\phi \mathbf{U} \psi] \Leftrightarrow \exists \pi, i \in \mathbb{N} : M, s_i \vDash \psi$ and $M, s_j \vDash \phi, 0 \leq j < i$.

A CTL formula ϕ represents a property that may be verified at certain states in the computation tree. In this context, a system M satisfies ϕ iff every one of its initial states does: $\bigwedge_{s_0 \in S_0} M, s_0 \vDash \phi$. A logic thus defined allows the formal expression of properties on evolving biological sequences and their eventual automated verification. Examples of non-trivial properties on sequences and trees were presented in [3].

3 Characterization of Branching-time Phylogenies with SMV

The use of model checking techniques is domain-independent. That is, the verification of a specific system is completely transparent to the end-user: given a model of the system (e.g., a phylogenetic tree) and a specification of its requirements (i.e., biological properties), the verification software automatically checks the correctness of the system. In the event of failure to comply with the specification, the software outputs the scenarios which infringe the property as counterexamples.

This section develops the implementation of the branching-time phylogenetic tree (Def. 1) into a model checking verification tool. To this effect, we have used Cadence SMV, a well-known model checking software tool [7]. As the input for the model checker, a description of the Kripke structure and the atomic propositions in Cadence SMV syntax must be provided by the user. To this end, we preprocess the sequence alignment and the phylogenetic tree. Translation from the phylogenetic tree to the SMV Kripke structure syntax has been performed automatically by a general conversion script. The inclusion in Cadence SMV syntax of loop constructs, macros and a rich assortment of vector operations facilitates a compact characterization of DNA and protein sequences as strings of characters.

Algorithm 1 Mapping of a phylogenetic tree in SMV.

```
MODULE main
VAR
 node: {N1,N2,N3,N4,N5};     /* States represent taxa in the phylogenetic tree */
 /* Function that asynchronously returns the DNA value that labels the current node */
 dna: process dna_sequence(node);
ASSIGN
 init(node) := N1;                /* Tree root */
 next(node) :=                   /* Definition of the successors of each node */
    case
      node=N1: {N2, N3};     /* Nodes N2 and N3 are the successors of N1 */
      node=N2: {N4, N5};     /* Nodes N4 and N5 are the successors of N2 */
      1: node;                    /* Self-loops in leaf nodes */
    esac;
MODULE dna_sequence(n){
 INPUT  n: {N1,N2,N3,N4,N5};
 /* Definition of the array of characters and the DNA alphabet */
 VAR    sequence: array 1..3 of {A,C,G,T};
 sequence:= switch(n){
      N1: [A,A,G];
      N2: [G,A,G];
      N3: [A,A,T];
      N4: [A,T,T];
      N5: [A,T,G];
      default: [A,A,A];
 };
}
```

Algorithm 1 shows the implementation of a branching-time phylogenetic tree in SMV code. The main module describes the topology of the evolutionary tree, where the names of the tree nodes (taxa) are defined as $N1,\ldots,N5$, and variables label the states (in this case, only the DNA string is defined). The *init* and *next* clauses are used to mark the root of the tree and the successors of a given state. The second part of the above description consists of the function returning the particular DNA string associated to each node. In the next section, we will verify phylogenetic properties using this description as input data.

4 Verification of Phylogenetic Properties with SMV

The performance evaluation of our system has been measured with human protein alignments that we have retrieved from GenBank [2]. In particular, we selected genes of respiratory complex I encoded in mitochondrial DNA (mtDNA). We have chosen them because they are biologically interesting and varied in length, which makes them suitable for a complete performance analysis. This data set includes ND5, one of the biggest genes in mtDNA. Thus, experimental results will draw out approximate upper bounds for work with mtDNA genes and help estimate costs

elsewhere. All tests have been run on a scientific workstation (Intel Core 2 Duo E6750 @ 2.66 GHz, 8 GB RAM, Debian Linux). Note that the public version of Cadence SMV is sequential and hence it uses only a single core.

Firstly, we analyze time and memory usage for the construction of the phylogenetic Kripke structure and for the storage of protein sequences. Table 1 shows time and memory consumption with respect to sequence length and set size.

		Set Size							
		500		1000		1500		2000	
Seq. size	Gene	Time	Memory	Time	Memory	Time	Memory	Time	Memory
98	NDL4	6.59	116	13.64	228	20.07	340	29.23	451
115	ND3	9.53	135	18.72	266	27.20	397	39.53	528
174	ND6	20.24	202	41.02	401	59.61	599	86.04	797
318	ND1	69.99	366	129.29	728	191.82	1089	261.54	1451
347	ND2	76.88	399	152.90	794	227.13	1188	326.11	1582
459	ND4	133.32	527	271.05	1048	393.49	1570	543.87	2091
603	ND5	224.58	691	450.07	1376	673.50	2061	970.39	2746

Table 1 Resources needed for the creation of the Kripke structure and the storage of protein sequences: time (in seconds) and memory (in megabytes).

Time increases linearly with the number of sequences and quadratically with gene length (Fig. 1). The regression graphics correspond to individual series in rows and columns, though the trends extend to the rest of the table. It is possible that these moderate trends are partly due to the use of highly conserved genes and closely related sequences (an interesting and common situation nonetheless).

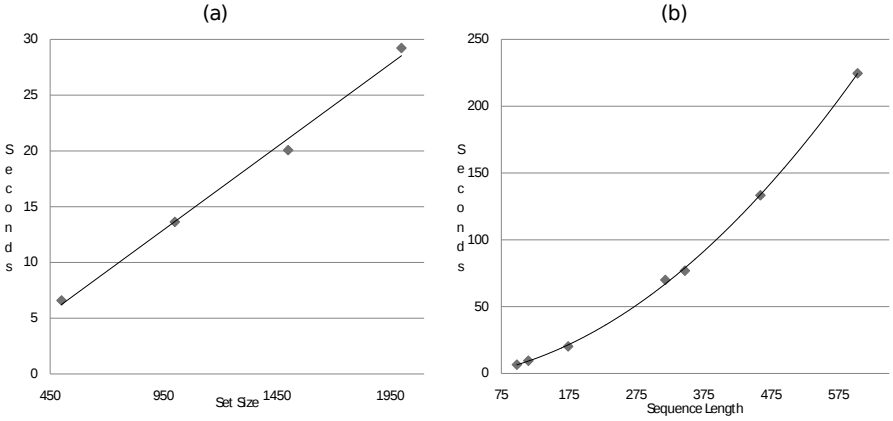

Fig. 1 **(a)** Time is linear with respect to set size (ND4, $y = 0.015x - 1.208, R^2 = 0.994$). **(b)** It is quadratic with respect to sequence length (size 500, $y = 0.005x^2 + 0.047x - 3.431, R^2 = 1.000$).

On the other hand, memory increases linearly in both dimensions, which is very encouraging from a computational point of view. Nevertheless, the huge amount of

memory required for data storage (more than 2.5 GB in the worst case), hints that the system can benefit from and may eventually be in need of some optimization.

In the second part of the benchmarks, we analyzed the impact of the verification process in overall execution times. The time required for verification of a single temporal logic formula is extremely variable, as it depends on formula complexity, search strategy of the model checker (e.g., depth or breadth first search) and the occurrence of verification interruptions caused by counterexamples. Nevertheless, we have measured the joint verification of 190 basic sequence properties and found them to add less than a minute to the results in Table 1, which seems very reasonable.

The properties that we have tested are related to conservation and covariation in the sequences. For example, Eq. 2 restricts the values that can appear in a substring (e.g., the presence of an amino acid in a specific position of a protein is restricted by the required physical and chemical properties of the protein) for some region of the tree. This means that there exists at least a point in the future where generally all the successors of the node verify that the characters remain equal (i.e., are conserved). In short, there exists a subtree (clade) where the conservation property holds in every state.

$$\mathbf{EF} \circ \mathbf{AG} \, (s\,[13 \ldots 17] = ACCTT) \tag{2}$$

Phylogenetic properties can be tuned according to different requirements. For example, the formula in Eq. 2 is also true for terminal leaves and we can extend the definition to detect strict subtrees. These last results are only particular examples of verification, but they primarily aim to offer insight into temporal costs for future model checking procedures. In view of the numerical results obtained for the experiments with our implementation, phylogenetic analysis over individual mtDNA genes takes approximately 15 minutes in the worst case. However, it seems that sequence length will be the main bottleneck if we straightforwardly apply our framework to bigger (e.g., nuclear) genes.

5 Conclusions

The aim of this paper has been to investigate the viability of model checking techniques as an inference framework for phylogenetic analysis. As prominent advantages stemming from the study of phylogenetic properties with this approach, different models of evolution can be considered, complex properties can be specified as the logical composition of others, and the refinement of unfulfilled properties (as well as the discovery of new ones) can be undertaken by exploiting the results of verification.

From a more technical point of view, we have shown, using Cadence SMV, how to translate phylogenetic trees into a specific model checker syntax.

We have also evaluated the performance of the model checker using phylogenetic data. We have seen that the initialization phase (creation of the associated Kripke

structure) is much costlier than the verification process. The experimental results show that initialization time increases linearly with set size and quadratically with sequence length. Additionally, memory consumption is linear in both cases. Nevertheless, the huge amount of memory required for the representation of the Kripke structure and the biological sequences points out that more efficient data structures will be needed in the future.

In particular, we have analyzed proteins coded by genes from the mtDNA genome, which are quite smaller than those from nuclear DNA. As temporal cost increases the most with respect to sequence length, phylogenetic analysis of large genes (and genomes) could become the major bottleneck in the near future. Thus, scaling the model checking verification process will be one of our main directions of future research.

We can conclude that our first approximation to phylogenetic analysis using model checking is innovative and encouraging in terms of efficiency. Although this framework is not directly applicable to bigger systems yet, it is at least a powerful and competitive approach for the analysis of mtDNA.

Acknowledgements This work was supported by the Spanish Ministry of Science and Innovation (MICINN) [TIN2008-06582-C03-02], the Spanish Ministry of Education [AP2008-03447] and the Government of Aragon [B117/10].

References

1. Baier, C., Katoen, J.-P.: Principles of model checking. The MIT Press, Cambridge, MA (2008)
2. Benson, D.A., Karsch-Mizrachi, I., Lipman, D.J., Ostell, J., Sayers, E.W.: GenBank. Nucleic Acids Res. **38**, D46–D51 (2010)
3. Blanco, R., de Miguel Casado, G., Requeno, J.I., Colom, J.M.: Temporal logics for phylogenetic analysis via model checking. In: Proceedings of the IEEE International Workshop on Mining and Management of Biological and Health Data, in press. IEEE, Los Alamitos, CA (2010)
4. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: Kozen, D. (ed.) Logics of Programs, pp. 52–71. Springer, Heidelberg (1982)
5. Felsenstein, J.: Inferring phylogenies. Sinauer, Sunderland, MA (2003)
6. Manna, Z., Pnueli, A.: The temporal logic of reactive and concurrent systems: specification. Springer, Berlin (1991)
7. McMillan, K. L.: A methodology for hardware verification using compositional model checking. Sci. Comput. Program. **37**, 279–309 (2000)
8. Montoya, J., López-Gallardo, E., Díez-Sánchez, C., López-Pérez, M.J., Ruiz-Pesini, E.: 20 years of human mtDNA pathologic point mutations: carefully reading the pathogenicity criteria. Biochim. Biophys. Acta **1787**, 476–483 (2009)