A model checker for interval temporal logic over finite structures*

Enrico Cominato[1], Dario Della Monica[2,3],
Angelo Montanari[4], and Guido Sciavicco[1]

[1] Dept. of Mathematics and Computer Science
University of Ferrara, Italy
({enrico.cominato@student.|guido.sciavicco@}unife.it)
[2] Dept. Sistemas Informáticos y Computación
Universidad Complutense de Madrid, Spain (ddellamo@ucm.es)
[3] Dept. of Electrical Engineering and Information Technology
University "Federico II" of Naples, Italy (dario.dellamonica@unina.it)
[4] Dept. of Mathematics, Computer Science, and Physics
University of Udine, Italy (angelo.montanari@uniud.it)

Abstract. Model checking is the process of establishing whether a certain formula is satisfied by a given structure, and it is usually associated with point-based temporal logics. Recently, the question of how to correctly define and study the model checking problem for interval-based temporal logics has been raised. In this paper, we focus on a very natural finite version of the model checking problem for Halpern and Shoham's modal logic of time intervals, a.k.a. HS, for which an algorithm that behaves in a very efficient way (under certain conditions) can be designed. We present an implementation of such an algorithm and analyse its performance through a systematic series of tests.

1 Introduction

Model checking is the problem of verifying whether a given logical formula is satisfied by a certain structure. It has been extensively studied in the area of formal methods and automated verification as a technique to establish the correctness of (hardware and software) reactive systems with respect to certain properties of interest. In most cases, the property to be verified is expressed in some *point-based* temporal logic, such as LTL [24], CTL [9], and CTL* [12]. The main reason is that the behavior of a (reactive) system is usually captured by specifying how its states can possibly evolve during an execution, and thus it is naturally represented as a labeled directed graph, where vertices represent the states of the system, edges encode its transitions, and paths describe executions. Interesting properties predicate about points (states) along a timeline

* A. Montanari and G. Sciavicco acknowledge the contribution from the Italian GNCS Project *"Logics and Automata for Interval Model Checking"*, while D. Della Monica acknowledges the financial support from a Marie Curie INdAM-COFUND-2012 Outgoing Fellowship.

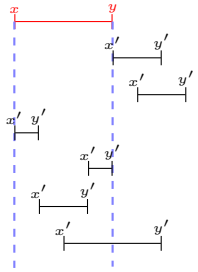
(execution), which makes point-based temporal logics a natural choice for the specification formalism.

In the last years, however, a certain interest is emerging towards model checking for temporal logics with an *interval-based* semantics, in particular for Halpern and Shoham's logic HS [13], which plays a prominent role in the literature about interval-based temporal logics, along with its fragments. HS can be viewed as the logic of Allen's relations [2]. While some of its theoretical properties, such as satisfiability and expressiveness, have been already studied in a very systematic way (see, for instance, [1, 8]), its model checking problem has entered the research agenda only recently, despite it has several potential application domains, as shown in [22].

Model checking for full HS, interpreted over finite Kripke structures, according to the so-called *state-based semantics* [6], has been studied in [18, 23]. The authors showed that, under the homogeneity assumption [25], which constrains a proposition letter to hold over an interval if and only if it holds over each component state, the problem is non-elementarily decidable (EXPSPACE-hardness has been later shown in [5]). Since then, the attention has been brought to HS fragments, whose computational behavior is often much better [5, 7, 19–21]. The model checking problem for some HS fragments extended with epistemic operators has been investigated in [15, 16]. The semantic assumptions for these epistemic HS fragments differ from those of [5, 7, 18–21, 23] in two important respects, making it difficult to compare the two families of logics: formulas are interpreted over the unwinding of the Kripke structure (*computation-tree-based semantics* [6]) and interval labeling takes into account only the endpoints of intervals. The latter assumption has been later relaxed by allowing the use of regular expressions to define the labeling of proposition letters over intervals in terms of the component states [17]. An in-depth investigation of interval temporal logic model checking with regular expressions, under the homogeneity assumption, has been recently undertaken [3, 4].

In this paper, we focus our attention on the problem of model checking a single, finite path/interval. Unlike standard model checking, where logical specifications are evaluated against temporal structures which are abstract (finite) representations of the infinite set of finite/infinite computation paths of a system, we are interested in checking the truth of a formula against a concrete, finite interval model. In [22], it has been shown that such a problem has various interesting applications, such as, for instance, *temporal query evaluation* and *temporal constraint checking* in temporal databases [10, 14, 26, 27].

We describe an implementation of a model checker for HS formulas against concrete, finite interval models, and we report the outcomes of a series of systematic performance tests. Even though the proposed implementation is inspired by the well-known model checking algorithm for CTL formulas by Emerson and Clarke [11], devising an efficient procedure is not trivial at all and it requires a deep theoretical analysis as well as suitable choices in representing and searching the finite model, e.g., a symbolic representation is used when labeling intervals with special kinds of formulas (see [22] for a more detailed discussion).

HS	Allen's relations	Graphical representation
$\langle A \rangle$	$[x,y]R_A[x',y'] \Leftrightarrow y = x'$	
$\langle L \rangle$	$[x,y]R_L[x',y'] \Leftrightarrow y < x'$	
$\langle B \rangle$	$[x,y]R_B[x',y'] \Leftrightarrow x = x', y' < y$	
$\langle E \rangle$	$[x,y]R_E[x',y'] \Leftrightarrow y = y', x < x'$	
$\langle D \rangle$	$[x,y]R_D[x',y'] \Leftrightarrow x < x', y' < y$	
$\langle O \rangle$	$[x,y]R_O[x',y'] \Leftrightarrow x < x' < y < y'$	

Shortcuts	Semantics
$[U]$	$[U]\psi \equiv \psi \wedge \bigwedge_{X \in \{A,B,E,D,O,L\}}([X]\psi \wedge [\overline{X}]\psi)$
$\langle SI \rangle$	$\langle SI \rangle \psi \equiv \bigvee_{X \in \{O,D,B,E\}}(\langle X \rangle \psi \vee \langle \overline{X} \rangle \psi)$

Fig. 1. Allen's interval relations, HS modalities, and some useful shortcuts.

2 The logic HS

Halpern and Shoham's logic HS is a multi-modal logic with formulas built on a set $\mathcal{AP}$ of proposition letters, the Boolean connectives $\vee$ and $\neg$, and 6 *existential* modalities based on the following 6 Allen's relations: *meets* (modality $\langle A \rangle$), *begins* (modality $\langle B \rangle$), and *ends* (modality $\langle E \rangle$), graphically depicted in Figure 1, and their transposes *met-by* (modality $\langle \overline{A} \rangle$), *begun-by* (modality $\langle \overline{B} \rangle$), and *ended-by* (modality $\langle \overline{E} \rangle$). Formulas of HS are defined by the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \langle A \rangle \varphi \mid \langle \overline{A} \rangle \varphi \mid \langle B \rangle \varphi \mid \langle \overline{B} \rangle \varphi \mid \langle E \rangle \varphi \mid \langle \overline{E} \rangle \varphi.$$

The other Boolean connectives are defined as usual, while the *universal* version of each existential modality $\langle X \rangle$ is denoted by $[X]$ and defined as $[X]\psi \equiv \neg\langle X \rangle \neg\psi$. The language of HS allows for additional (existential) modalities corresponding to the other Allen's relations, namely, *during* (modality $\langle D \rangle$), *later* (modality $\langle L \rangle$), and *overlaps* (modality $\langle O \rangle$), graphically depicted in Figure 1, and their transposes *contains* (modality $\langle \overline{D} \rangle$), *before* (modality $\langle \overline{L} \rangle$), and *overlapped-by* (modality $\langle \overline{O} \rangle$), which can be defined as follows:

$$\begin{array}{ll} \langle D \rangle \psi \equiv \langle B \rangle \langle E \rangle \psi, & \langle \overline{D} \rangle \psi \equiv \langle \overline{B} \rangle \langle \overline{E} \rangle \psi, \\ \langle L \rangle \psi \equiv \langle A \rangle \langle A \rangle \psi, & \langle \overline{L} \rangle \psi \equiv \langle \overline{A} \rangle \langle \overline{A} \rangle \psi, \\ \langle O \rangle \psi \equiv \langle E \rangle \langle \overline{B} \rangle \psi, & \langle \overline{O} \rangle \psi \equiv \langle B \rangle \langle \overline{E} \rangle \psi. \end{array}$$

Hereafter, we denote by R_X the Allen relation corresponding to modality $\langle X \rangle$, e.g., we denote by R_A the relation corresponding to modality $\langle A \rangle$ (see Figure 1).

Let $\mathbb{D} = \langle D, < \rangle$ be a finite linearly ordered set. An *interval* over $\mathbb{D}$ is an ordered pair $[x, y]$, where $x, y \in D$ and $x < y$ (*strict semantics*). An interval of the type $[x, x+1]$ is called *unit* ($x+1$ denotes the immediate successor of x in $\mathbb{D}$).

The semantics of HS is given in terms of *interval models* $M = \langle \mathbb{I}(\mathbb{D}), V \rangle$, where $\mathbb{I}(\mathbb{D})$ is the set of all intervals over $\mathbb{D}$ and $V : \mathcal{AP} \mapsto 2^{\mathbb{I}(\mathbb{D})}$ is a *valuation function* that assigns to every $p \in \mathcal{AP}$ the set of intervals $V(p)$ over which p holds. Since every finite linear order is isomorphic to a prefix of the set $\mathbb{N}$ of the natural numbers, for every $N \in \mathbb{N}$ we denote the linear order $\mathbb{D} = \{0, 1, \ldots, N-1\}$ by $[N]$. The *truth relation* $\Vdash$ of an HS formula over a given interval $[x, y]$ in an interval model M is defined by structural induction on formulas as follows:

- $M, [x, y] \Vdash p$ iff $[x, y] \in V(p)$;
- Boolean connectives are dealt with in the standard way;
- $M, [x, y] \Vdash \langle A \rangle \varphi$ iff there exists $z > y$ such that $M, [y, z] \Vdash \varphi$;
- $M, [x, y] \Vdash \langle \overline{A} \rangle \varphi$ iff there exists $z < x$ such that $M, [z, x] \Vdash \varphi$;
- $M, [x, y] \Vdash \langle B \rangle \varphi$ iff there exists $x < z < y$ such that $M, [x, z] \Vdash \varphi$;
- $M, [x, y] \Vdash \langle \overline{B} \rangle \varphi$ iff there exists $z > y$ such that $M, [x, z] \Vdash \varphi$;
- $M, [x, y] \Vdash \langle E \rangle \varphi$ iff there exists $x < z < y$ such that $M, [z, y] \Vdash \varphi$;
- $M, [x, y] \Vdash \langle \overline{E} \rangle \varphi$ iff there exists $z < x$ such that $M, [z, y] \Vdash \varphi$.

We say that an HS formula φ is *satisfiable* if and only if there exists a model M and an interval $[x, y]$ such that $M, [x, y] \Vdash \varphi$. The *satisfiability problem for* HS is the problem of finding a model and an interval that satisfies a formula.

Given an interval model $M = \langle \mathbb{I}(\mathbb{D}), V \rangle$ over a finite linearly ordered set $\mathbb{D} = \langle D, < \rangle$, an interval $[x, y]$ in M, and an HS formula φ, the problem of checking φ against $[x, y]$ in M (*finite (concrete) model checking*, *model checking* for short) consists in establishing whether or not $M, [x, y] \Vdash \varphi$.

In the rest of the paper, we make use of the following two shortcuts: the *global* modality $[U]\psi$, that forces ψ to hold everywhere in the model, and the *strong intersect* modality $\langle SI \rangle$, that forces ψ to hold on some interval $[z, t]$, distinct from the current interval $[x, y]$, such that at least one $w \in \{z, \ldots, t\}$ falls inside $[x, y]$. Their formal definitions are given in Figure 1.

3 A model checking algorithm for HS

In this section, we present the algorithm for model checking HS formulas against (concrete) finite intervals/models [22].

Data structures for representing the input. The algorithm receives as input an instance of the model checking problem, that is, a pair consisting of an HS formula and an interval model.

An HS formula can be represented as a binary rooted decorated tree. A *rooted tree* is a triple $G = (V, E, r)$, where (V, E) is a directed acyclic graph and $r \in V$ is a distinguished vertex, called *root* of G, such that there is exactly one incoming edge for each vertex except for the root, which has none, that is, $\{(u, r) \mid u \in E\} = \emptyset$ and $|\{(u, v) \mid u \in E\}| = 1$, for each $v \in V \setminus \{r\}$. Vertices of a tree are also called *nodes*. A *rooted decorated tree* is a rooted tree equipped with a function that associates a *decoration* with each node. A decoration is either a proposition letter or an operator, which, in turn, is either a Boolean or a modal

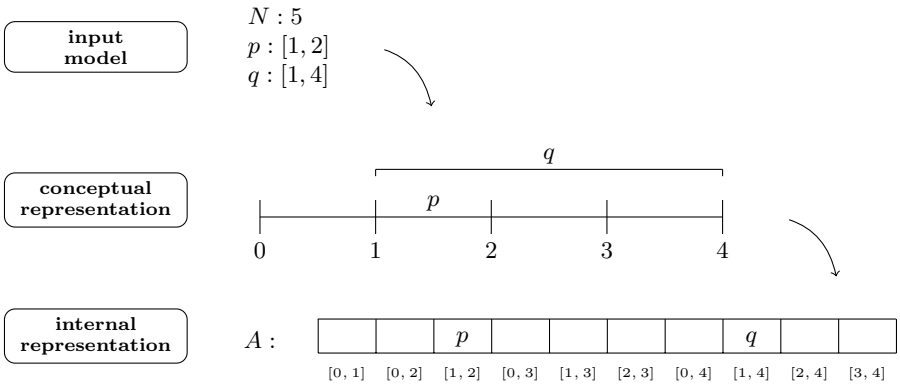


Fig. 2. An example of model representation as (i) input model (top side), (ii) conceptual model (middle), and (iii) internal data structure (bottom).

operator. For each edge $(v, u) \in E$, we say that u is a *child* of v. We focus on *binary* trees, that is, trees in which each node has at most 2 children, which we refer to as its *left* and its *right* child. If a node has only one outgoing edge, then its right child is undefined. Nodes without children are called *leaves*.

As a preliminary step, the algorithm transforms the input formula into an equivalent one belonging to the language generated by the grammar in Section 2, that is, a formula only containing negations, disjunctions, conjunctions, and existential modalities from the set $\{\langle A\rangle, \langle B\rangle, \langle E\rangle, \langle \overline{A}\rangle, \langle \overline{B}\rangle, \langle \overline{E}\rangle\}$, and, then, it stores the resulting formula into a tree-like structure.

An input model M is represented by a string carrying the following pieces of information: a number, that specifies the size (number of points) of the linear order, and, for each proposition letter, a list of intervals encoding the valuation function (Figure 2, top). Input models are stored in a linear array A of pointers (Figure 2, bottom), whose elements correspond to intervals, in ascending order with respect to their second coordinate and, if their values on the second coordinate is the same, with respect to their first coordinate. In Figure 2, the first (resp., second, third) element of A corresponds to the interval $[0, 1]$ (resp., $[0, 2]$, $[1, 2]$), and so on. This is a classic representation of an upper triangular matrix, that reflects the intrinsic structure of an interval model. At the beginning, each element of A contains (points to) those proposition letters that hold at the corresponding interval, that is, A represents the valuation function V of M. Then, the algorithm will associate with each element of A the set of pointers to the subtrees of the formula tree that 'hold' in the corresponding interval.

Figure 2 shows a model, based on the linear order $\mathbb{D} = \{0, 1, 2, 3, 4\}$, which features the set of proposition letters $\mathcal{AP} = \{p, q\}$. At the top, the unstructured input model is given. Such an unstructured model corresponds to the conceptual one shown in the middle of the picture. The internal representation of the model is depicted at the bottom.

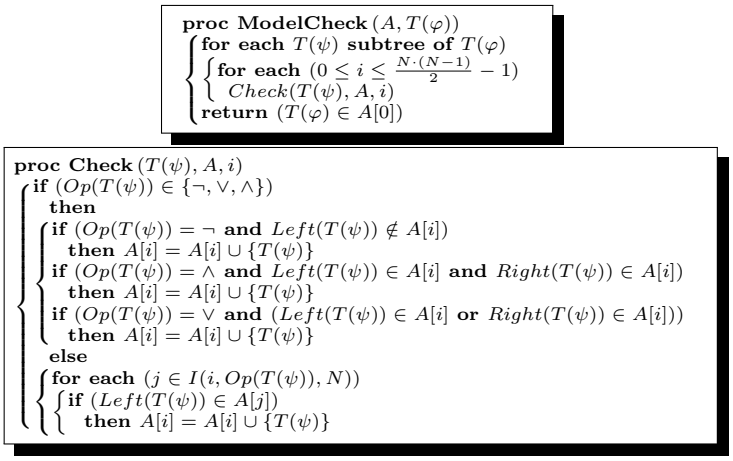


Fig. 3. Pseudo-code of the model checking procedure.

Execution. Following Clarke and Emerson's schema [9], model checking a given HS formula (represented as a binary tree) against a certain model (represented as an array of sets of pointers to trees) can be done by a simple procedure that executes a bottom-up visit of the formula tree.

Let φ be the formula to be checked and, for each subformula ψ, let $T(\psi)$ be the corresponding subtree. Moreover, let $M = \langle [N], V \rangle$ be the model against which φ is to be checked. The size of the array A used to store M is equal to the number of intervals featured by M, namely, $\frac{N \cdot (N-1)}{2}$. For each $i \in \{0, \ldots, \frac{N \cdot (N-1)}{2} - 1\}$, let us denote by $Int(i)$ the interval corresponding to the i-th position in A. As an example, we denote by $Int(2)$ the interval $[1, 2]$ (see Figure 2). For a binary tree T that represents a formula, we denote by $Op(T)$ the decoration of the root of T, and we denote by $Left(T)$ (resp., $Right(T)$) the subtree rooted in its left (resp., right) child.

We define a function I that, for any position i in A and any modality $\langle X \rangle$, returns the set of positions corresponding to intervals $[z, t]$ such that $Int(i) R_X [z, t]$. Besides the position i and the modality $\langle X \rangle$, I clearly depends on the size N of the underlying linear order:

$$I : \mathbb{N} \times \{A, B, E, \overline{A}, \overline{B}, \overline{E}\} \times \mathbb{N} \to 2^{\mathbb{N}}.$$

As an example, with reference to the model depicted in Figure 2, where $N = 5$, we have that $I(2, A, 5) = \{5, 8\}$.

In Figure 3, we provide the pseudo-code of the model checking procedure, and then, in Figure 4, we show the result of checking the formula $\varphi \equiv \langle A \rangle (\langle B \rangle p \wedge q)$ against the model in Figure 2.

Theoretical complexity and optimization. In [22], Della Monica et al. have shown that the problem of checking an HS formula against a finite (concrete) interval model can be solved by a deterministic algorithm that runs in polyno-

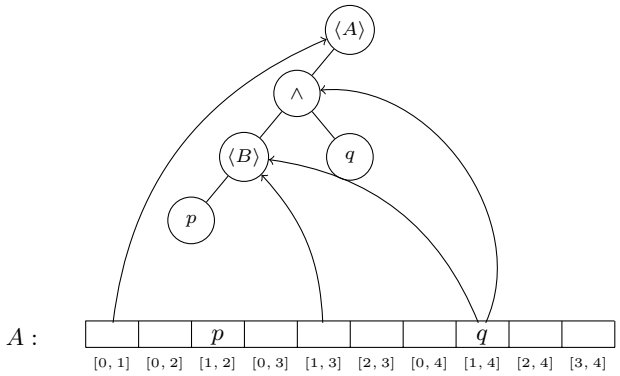


Fig. 4. A graphical account of the result of the execution of the model checking procedure on the input (φ, M), where $\varphi \equiv \langle A \rangle (\langle B \rangle p \wedge q)$ and M is the model in Figure 2.

mial time in the size of the input. The execution time of our implementation of the model checking procedure is polynomial in the size of the representation of the input formula, but it is not necessarily polynomial in the size of the representation of the model (this is the case with some particular input models).

Classic model checking, e.g., LTL and CTL model checking, where infinite paths are finitely encoded in the input structure, is infinite in nature. In our setting, as shown in Figure 2, models can be compactly represented by specifying the size (number of points) of the underlying domain and then listing, for each proposition letter, the set of intervals over which it holds. The fundamental difference between the two frameworks is that, in classic model checking, frame information, i.e., states and transitions, must be explicitly represented in the input, while, in our setting, frame information, i.e., intervals and their relations, is implicit in the size of the temporal domain, as the relations among intervals are induced by the underlying linear order. As a consequence, while the size of the representation of a Kripke structure is polynomial in the number of states and labels, the size of the representation of a finite interval model may be logarithmic in the number of intervals.

As an example, consider an interval model $M = \langle [N], V \rangle$ over $\mathcal{AP} = \{p\}$, with $V(p) = \{[N-1, N]\}$. Its representation consists of the number N, whose binary encoding takes space $O(log(N))$, and the mapping $p \mapsto \{[N-1, N]\}$, which takes space $O(log(N))$ as well, as it suffices to represent two numbers. In [22], it has been shown that such degenerate instances can be reduced in polynomial time to non-degenerate, equivalent ones, making it possible to efficiently employ the proposed algorithm to model check any instance.

To improve the performance of the algorithm, we devised a non-trivial heuristics that exploits a suitable form of 'lemma caching'. It makes use of a complex data structure, denoted here by S, which can be seen as a hash table (accessible in constant time) that, for each already-visited subtree $T(\psi)$, keeps track of the set of positions $S(T(\psi))$ in A corresponding to intervals where ψ is true.

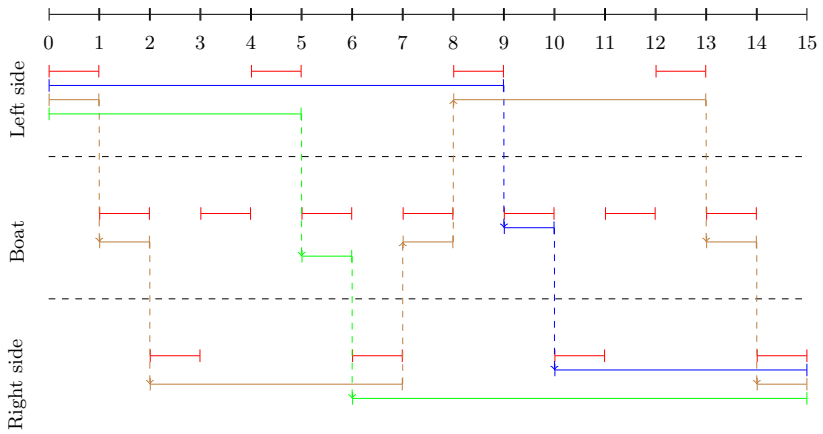


Fig. 5. A model for the classic version of wolf (blue), goat (brown), and cabbage (green) problem in HS (farmer is associated with red).

By construction, $S(T(\psi))$ is undefined if $T(\psi)$ has never been visited before. At each iteration, before checking a generic subtree $T(\xi)$, the procedure makes the following steps:

1. if $\xi = \psi$, that is, if ξ and ψ are two occurrences of the same sub-formula, for some ψ such that $S(T(\psi))$ is defined, then, for each index i, we add to $A[i]$ a pointer to $T(\xi)$ if and only if $i \in S(T(\psi))$;
2. otherwise, if $Op(T(\xi))$ is a modality and $|S(Left(T(\xi)))| = 0$, then no pointer to $T(\xi)$ is added to any cell of A.

Notice that whenever any of the above conditions applies, executing the entire check for $T(\xi)$ takes precisely one pass over A instead of two.

4 Experimental analysis

In this section, we describe the outcomes of a set of systematic experiments to test the effectiveness of our implementation. To start with, we consider the well-known riddle *wolf, goat, and cabbage*, which can be described as follows. A farmer wants to cross a river and take with him a wolf, a goat, and a cabbage. There is a boat that can fit himself plus either the wolf, the goat, or the cabbage, and the following constraints apply: if the wolf and the goat are alone on one shore, the wolf will eat the goat, and if the goat and the cabbage are alone on the shore, the goat will eat the cabbage. The problem asks to find a plan so that the farmer can bring the wolf, the goat, and the cabbage safely across the river. This problem can be easily described in HS, using proposition letters that represent the localization (left hand side of the river, right hand side of the river, or the boat) of the farmer, the wolf, the goat, and the cabbage at each time. A model that solves the problem is represented in Figure 5.

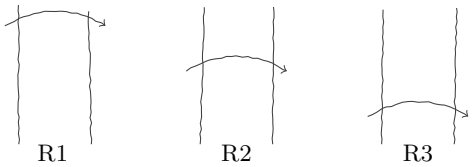


Fig. 6. Graphical representation of the generalized wolf, goat, and cabbage riddle.

To test our implementation, we designed a generalized version of the above-described riddle with n characters (or elements) and m rivers. In this version, the (single) farmer must cross every element $e_1, \ldots, e_n$ from the left side of the i-th river to its right side using a boat that may host, at most, $n-2$ elements at the same time, plus the farmer. The main constraints of the system are: *(i)* for each j, e_j "eats" e_{j+1} (if any), and *(ii)* the relation "eats" is not transitive. Clearly, to keep the meaning of the original riddle unchanged, the farmer must bring everyone across all rivers, sequentially (no river can be skipped). We give a formalization of this generalized version of the problem in HS, which makes use of the following proposition letters:

- for $i \in \{0, \ldots, m\}$, $f^{i,S}$ represents the situation where the farmer is on some river side: $f^{0,S}$ means that the farmer is on the left side of the 1st river, $f^{m,S}$ means that the farmer is on the right side of the m-th river, and $f^{i,S}$, with $1 \leq i < m$, means that the farmer is on the right side of the i-th river (and on the left side of the $(i+1)$-th one); analogously, proposition letters $e_j^{i,S}$, with $i \in \{0, \ldots, m\}$ and $j \in \{1, \ldots, n\}$, represent the situation where the j-th element is on some river side;
- for $i \in \{1, \ldots, m\}$, $f^{i,B}$ represents the situation where the farmer is on a boat crossing the i-th river; analogously, proposition letters $e_j^{i,B}$, with $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$, represent the situation where the j-th element is on a boat crossing the i-th river.

The problem can be encoded in HS as follows. First of all, we introduce a formula φ_{boat}, which states that if anyone is on the boat, then the farmer must be as well, and a formula φ_{safe}, which prevents two incompatible elements from being alone (unguarded) on the same side of a river:

$$\begin{aligned} \varphi_{boat} &\equiv [U] \textstyle\bigwedge_{i,j} (e_j^{i,B} \rightarrow f^{i,B}); \\ \varphi_{safe} &\equiv [U] \textstyle\bigwedge_{i,j} ((e_j^{i,S} \wedge e_{j+1}^{i,S} \wedge \neg f^{i,S}) \rightarrow \bot). \end{aligned}$$

Next, we need to force some obvious physical constraints:

$$\varphi_{phy} \equiv \begin{cases} [U] \bigwedge_{i>0,j} (e_j^{i,S} \rightarrow (\langle \overline{L} \rangle e_j^{i,B} \vee \langle \overline{A} \rangle e_j^{i,B})); & (i) \\ [U] \bigwedge_{i>0} (f^{i,S} \rightarrow (\langle \overline{L} \rangle f^{i,B} \vee \langle \overline{A} \rangle f^{i,B})); & (ii) \\ [U] \bigwedge_{i,j} (e_j^{i,B} \rightarrow (\langle \overline{L} \rangle e_j^{i-1,S} \vee \langle \overline{A} \rangle e_j^{i-1,S})); & (iii) \\ [U] \bigwedge_{i} (f^{i,B} \rightarrow (\langle \overline{L} \rangle f^{i-1,S} \vee \langle \overline{A} \rangle f^{i-1,S})); & (iv) \\ [U] \bigwedge_{i,j,*} (e_j^{i,*} \rightarrow \bigwedge_{i',*'} \neg \langle SI \rangle e_j^{i',*'}); & (v) \\ [U] \bigwedge_{i,*} (f^{i,*} \rightarrow \bigwedge_{i',*'} \neg \langle SI \rangle f^{i',*'}). & (vi) \end{cases}$$

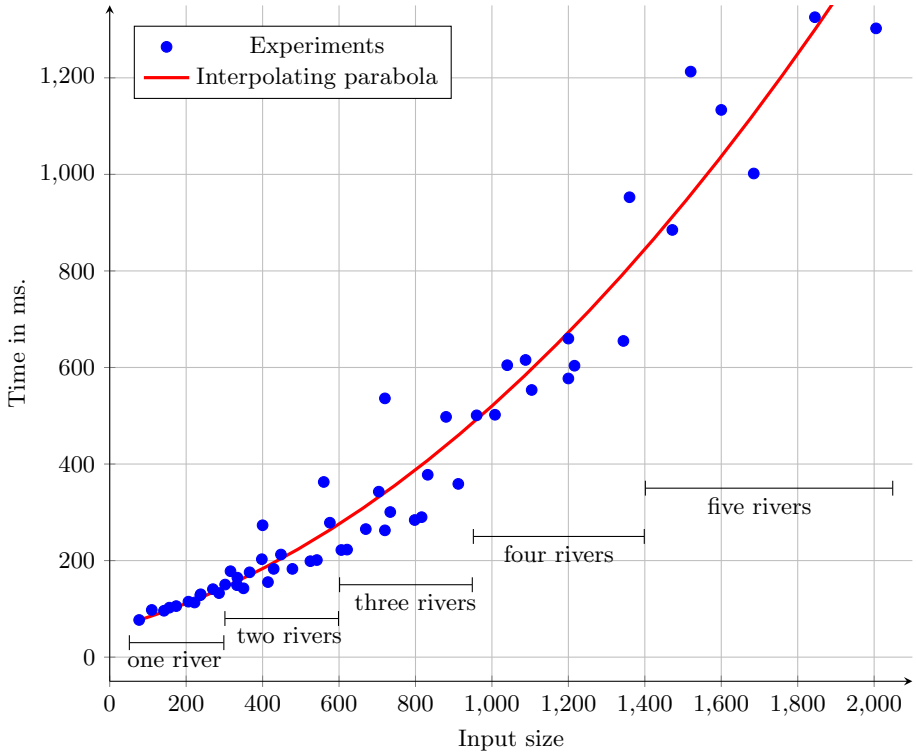


Fig. 7. Elapsed time.

Formulas (i) and (ii) impose that the farmer, as well as any element, has been on the boat on the i-th river before being on the right hand side of that river. Similarly, formulas (iii) and (iv) state that the farmer, as well as any element, must have been on the left hand side of the i-th river before being on the boat across it. Furthermore, formulas (v) and (vi) constrain the model to be coherent: whenever any element or the farmer are somewhere, they are not anywhere else.

Finally, we add a formula φ_{riddle} (the riddle constraint) which states that the capacity of the boat is limited (as in the original formulation of the riddle, it states that the boat cannot host all the elements together) and that everyone starts at the left hand side of the first river and, eventually, ends up at the right hand side of the last one:

$$\varphi_{riddle} \equiv \begin{cases} f^{0,S} \wedge e_1^{0,S} \wedge \ldots \wedge e_n^{0,S}, \\ [U] \bigwedge_j (\bigwedge_i (e_1^{i,B} \wedge \ldots \wedge e_{j-1}^{i,B} \wedge e_{j+1}^{i,B} \wedge \ldots \wedge e_n^{i,B}) \rightarrow \bot), \\ \langle L \rangle (f^{m,S} \wedge e_1^{m,S} \wedge \ldots \wedge e_n^{m,S}). \end{cases}$$

In Figure 7, we have reported the elapsed time for model checking the conjunction of the above formulas against a generalized version of the model in Figure 5. The size of the problem is expressed in bytes, and time in milliseconds. All experiments have been run on a CPU Intel(R) Core(TM) i7, with a clock

of 2.60GHz and 8GB RAM. For each number of rivers, ranging from 1 to 5, we have tried to model check the conjunction of all specifications for a number of elements ranging from 8 to 48 (with 4-unit steps).

As one may observe from Figure 7, the elapsed time grows quadratically, i.e., as a parabola, with the size of the input (size of the model plus size of the formula), as expected in a non-degenerate experiment.

5 Conclusions

Model checking is a fundamental problem in computational logic. In the recent past, it has been applied to a number of different contexts, ranging from program and protocol checking to system verification. Point-based temporal logics have been traditionally used for property specification in model checking. The interest for interval temporal logic model checking is much more recent.

We focused on the problem of model checking formulas of the interval temporal logic HS against (concrete) finite interval models, which comes into play in a number of applications different from automated system verification. As an example, constraint checking and query answering in temporal databases can be viewed as special instances of such a problem. The same holds for temporal rule extraction in data mining. A formalization of the problem, an analysis of its complexity, and a discussion of some possible applications can be found in [22].

Here, we proposed a first implementation of the model checking procedure outlined in [22], and we reported the outcomes of its systematic testing against benchmarks obtained from the generalization of a known riddle and its solution.

References

1. L. Aceto, D. Della Monica, V. Goranko, A. Ingólfsdóttir, A. Montanari, and G. Sciavicco. A complete classification of the expressiveness of interval logics of Allen's relations: The general and the dense cases. *Acta Informatica*, 53(3):207–246, 2016.
2. J. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.
3. L. Bozzelli, A. Molinari, A. Montanari, and A. Peron. An in-Depth Investigation of Interval Temporal Logic Model Checking with Regular Expressions. In *Proc. of the 15th SEFM*, volume 10469 of *LNCS*, 2017.
4. L. Bozzelli, A. Molinari, A. Montanari, and A. Peron. On the Complexity of Model Checking for Syntactically Maximal Fragments of the Interval Temporal Logic HS with Regular Expressions. In *Proc. of the 8th GandALF*, 2017.
5. L. Bozzelli, A. Molinari, A. Montanari, A. Peron, and P. Sala. Interval temporal logic model checking: The border between good and bad HS fragments. In *Proc. of the 8th IJCAR*, volume 9706 of *LNCS*, pages 389–405. Springer, 2016.
6. L. Bozzelli, A. Molinari, A. Montanari, A. Peron, and P. Sala. Interval vs. point temporal logic model checking: an expressiveness comparison. In *Proc. of the 36th FSTTCS*, volume 65 of *LIPIcs*, pages 26:1–26:14, 2016.
7. L. Bozzelli, A. Molinari, A. Montanari, A. Peron, and P. Sala. Model checking the logic of Allen's relations meets and started-by is P^{NP}-complete. In *Proc. of the 7th GandALF*, volume 226 of *EPTCS*, pages 76–90, 2016.

8. D. Bresolin, D. Della Monica, A. Montanari, P. Sala, and G. Sciavicco. Interval temporal logics over strongly discrete linear orders: Expressiveness and complexity. *Theoretical Computer Science*, 560:269–291, 2014.
9. E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logics of Programs*, volume 131 of *LNCS*, pages 52–71. Springer, 1981.
10. C. Combi and A. Montanari. Data models with multiple temporal dimensions: Completing the picture. In *Proc. of the 13th CAiSE*, volume 2068 of *LNCS*, pages 187–202. Springer, 2001.
11. E.A. Emerson and J.Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *Journal of Computer and System Sciences*, 30(1):1–24, 1985.
12. E.A. Emerson and J.Y. Halpern. "Sometimes" and "Not Never" revisited: On branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.
13. J. Halpern and Y. Shoham. A propositional modal logic of time intervals. *J. of the ACM*, 38(4):935–962, 1991.
14. K. Kulkarni and J.E. Michels. Temporal features in SQL:2011. *ACM SIGMOD Record*, 41(3):34–43, 2012.
15. A. Lomuscio and J. Michaliszyn. An epistemic Halpern-Shoham logic. In *Proc. of the 23rd IJCAI*, pages 1010–1016, 2013.
16. A. Lomuscio and J. Michaliszyn. Decidability of model checking multi-agent systems against a class of EHS specifications. In *Proc. of the 21st ECAI*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 543–548, 2014.
17. A. Lomuscio and J. Michaliszyn. Model checking multi-agent systems against epistemic HS specifications with regular expressions. In *Proc. of the 15th KR*, pages 298–308, 2016.
18. A. Molinari, A. Montanari, A. Murano, G. Perelli, and A. Peron. Checking interval properties of computations. *Acta Informatica*, 53(6-8):587–619, 2016.
19. A. Molinari, A. Montanari, and A. Peron. Complexity of ITL model checking: Some well-behaved fragments of the interval logic HS. In *Proc. of the 22nd TIME*, pages 90–100, 2015.
20. A. Molinari, A. Montanari, and A. Peron. A model checking procedure for interval temporal logics based on track representatives. In *Proc. of the 24th CSL*, volume 41 of *LIPIcs*, pages 193–210, 2015.
21. A. Molinari, A. Montanari, A. Peron, and P. Sala. Model checking well-behaved fragments of HS: the (almost) final picture. In *Proc. of the 15th KR*, pages 473–483, 2016.
22. D. Della Monica, D. de Frutos-Escrig, A. Montanari, A. Murano, and G. Sciavicco. Evaluation of temporal datasets via interval temporal logic model checking. In *Proc. of the 24th TIME*. LIPIcs, 2017.
23. A. Montanari, A. Murano, G. Perelli, and A. Peron. Checking interval properties of computations. In *Proc. of the 21st TIME*, pages 59–68, 2014.
24. A. Pnueli. The temporal logic of programs. In *Proc. of the 18th FOCS*, pages 46–57, 1977.
25. P. Roper. Intervals and tenses. *Journal of Philosophical Logic*, 9(4):451–469, 1980.
26. R.T. Snodgrass, editor. *The TSQL2 Temporal Query Language*. Kluwer, 1995.
27. R.T. Snodgrass, I. Ahn, G. Ariav, D.S. Batory, J. Clifford, C.E. Dyreson, R. Elmasri, F. Grandi, C.S. Jensen, W. Käfer, N. Kline, K. Kulkarni, T.Y.C. Leung, N. Lorentzos, J.F. Roddick, A. Segev, M.D. Soo, and S.M. Sripada. TSQL2 language specification. *ACM Sigmod Record*, 23(1):65–86, 1994.