**Michał KAŃSKI**[1],
**Artur NIEWIADOMSKI**[1],
**Magdalena KACPRZAK**[2],
**Wojciech PENCZEK**[3],
**Wojciech NABIAŁEK**[1]

[1] Siedlce University of Natural Sciences and Humanities
Faculty of Exact and Natural Sciences
Institute of Computer Science
ul. 3 Maja 54, 08-110 Siedlce, Poland

[2] Białystok University of Technology,
Faculty of Computer Science
ul. Wiejska 45A, 15-351, Białystok , Poland

[3] Institute of Computer Science,
Polish Academy of Sciences,
ul. Jana Kazimierza 5, 01-248, Warsaw, Poland

# Unbounded Model Checking for ATL

DOI: 10.34739/si.2021.25.01

**Abstract.** In this paper, we deal with verification of multi-agent systems represented as concurrent game structures. To express properties to be verified, we use Alternating-Time Temporal Logic (ATL) formulas. We provide an implementation of symbolic model checking for ATL and preliminary, but encouraging experimental results.

**Keywords.** ATL, Temporal Logics, Model Checking, SAT, SMT, QBF

## 1. Introduction

Many dangerous situations caused by software bugs have happened over the past decades. Testing is a commonly used method of detecting errors in systems. There are various types of tests applied to detect and eliminate bugs in developed systems, e.g., unit and integration tests. Testing process greatly improves the software quality, as it helps to find bugs but it does not ensure that the system is error-free. However, such a guarantee can be provided by formal methods.

Formal methods are based on solid mathematical and logical basis, and they provide techniques enabling the design of reliable and error-free systems. One of the formal methods is Model Checking (MC) [29, 13, 5] which can be applied to ensure that given model of hardware or software system behaves according to its specification. Typically, a system is modelled as a labelled transition system, and the specification is usually a set of temporal formulas expressing properties like safety (bad things never happen) or liveness (good things eventually happen). An example of the former could be: *two processes never reach a critical section simultaneously*, and of the latter: *every process reach a critical section eventually.*

One of the well-known temporal logic is Computation Tree Logic (CTL) [12] introduced by Clarke and Emmerson in 1981. Here, the formulas are interpreted over a tree-like structure where the future is not determined, because there exist a number of possible paths that may be actually realized. This is a branching-time logic allowing to quantification over the computation paths. In CTL, every temporal operator should be preceded by A (*along all paths*) or E (*there exists a path*).

However, in the case of multi-agent systems, other temporal logics are suitable to express properties involving cooperation and strategic abilities. One of them is Alternating-time Temporal Logic (ATL) [2-4, 22, 17, 18, 20]. The strategic modalities enable to formulate properties like there exists a strategy such that a goal will be achieved by an agent, or a group of agents. Thus, ATL allows for selective quantifications over paths that are outcomes of games between (groups of) agents. For example, the ATL formula $\ll A \gg X\varphi$means that the group of agents A has a strategy to enforce the property $\varphi$ in the next step, regardless of the actions performed by the other agents.

Model checking has been used for over four decades to verify various hardware and software systems [36, 35, 1, 8]. However, one of its biggest obstacles is a huge number of states in the verified systems, since it grows exponentially with the number of system components (e.g., agents). Usually, model checking can be reduced to a kind of a graph search problem, which could be solved using either *explicit* or *symbolic* methods. Symbolic model checking makes use of logical formulas or binary decision diagrams (BDDs) [9] to represent sets of states and transitions, and handle a number of them at once. In many cases, this approach is much more efficient than the explicit one, and symbolic representations of transition systems are often quite successful in alleviating the state explosion problem. A lot of model-checking tools exploit BDDs to represent the state spaces. These are, e.g., NuSMV [10] for verifying CTL and MCMAS [26, 27] for verifying of properties expressed in CTL and ATL against systems specified in Incremental System Programming Language (ISPL) [28].

Sometimes, however, the BDDs have a tendency to an exponential blow-up in the number of variables, what impedes the verification of large systems. The methods aimed at addressing this problem include, amongst others, model checking algorithms based on propositional satisfiability (SAT) checking [34]. The Unbounded Model Checking (UMC) method, introduced by McMillan [30], is based on modification of Davis-Logemann-Loveland (DLL) [14] algorithm. It eliminates universal quantifiers from Boolean expressions, enabling evaluation of arbitrary CTL formulas using fixed point characterizations of the CTL operators. The proposed method is extremely efficient in cases, when the resulting fixed points do not have a concise representation as a BDD, but can be succinctly described as CNF formula.

The authors of [21] show that UMC can be also applied to verification of ATL. The key issue in solving this problem consists in encoding the *next time* operator by a Quantified Boolean Formula (QBF) and translating it to a corresponding propositional formula. The other modal operators are computed as the greatest or least fixed points of functions defined over the basic *next time* operator.

The main contribution of this paper is a practical realization of the UMC method for multi-agent systems modelled in terms of Concurrent Game Structures (CGSs) [24] and the properties expressed as ATL formulas. We follow the theoretical results reported in [21] and provide, to our best knowledge, the first implementation of this method together with preliminary experimental results. However, we do not translate the verification problem to CNF, but we stop at the QBF level and use SMT-solver Z3 [15] to solve it. We compare the efficiency of our tool with the MCMAS model checker.

The rest of the paper is structured as follows. In the next sections we introduce CGSs, and then syntax, semantics, and recall the fixed-point characterization of ATL operators and their translation to QBF. Then, we present the most important details of implementation of our tool UMC4ATL and preliminary experimental results, followed by conclusion.

## 2. Concurrent Game Structures

In our approach, we model the systems under consideration by Concurrent Game Structures. The transitions between (global) states are determined by actions made by the system components. Each global transition represents a simultaneous step made by all the system components. Formally, following [21], CGS is defined as follows.

**Definition 1.** A Concurrent Game Structure is a tuple $S =< k, Q, \tau, \Pi, \pi, d, \delta >$ , where:

- k is a natural number defining the amount of agents.
- We identify the agents with numbers $0, \ldots, k$ so the set of agents is $\{0, \ldots, k-1\}$,
- $Q$ is a finite set of global states, and $\tau \in Q$ is the initial state,
- $\Pi$ is a finite set of atomic propositions (also called observables),
- $\pi: Q \rightarrow 2^{\Pi}$ is a labeling (or observation) function,
- moves (actions) available at a state $q \in Q$ to an agent $a \in \{0, \ldots, k-1\}$ are identified with numbers $0, \ldots, d_a(q)$; so given a state q, a move vector at q is a tuple $< j_0, \ldots, j_{k-1} >$ such that $j_a \leq d_a(q)$ for every agent a; then d is the mapping that assigns for every state q the set $\{0, \ldots, d_0(q)\} \times \ldots \times \{0, \ldots, d_{k-1}(q)\}$ of move vectors,
- $\delta$ is a transition function which assigns to each state $q \in Q$ and each move vector $< j_0, \ldots, j_{k-1} > \in d(q)$ a state $\delta(q, j_0, \ldots, j_{k-1}) \in Q$ that results from state q if every agent $a \in \{0, \ldots, k-1\}$ chooses move $j_a$.

We say that a state $q$ is a successor of a state $q'$ if there is a move vector $< j_0, \ldots, j_{k-1} > \in d(q)$ such that $q' = \delta(q, j_0, \ldots, j_{k-1})$. Thus $q'$ is a successor of $q$ iff whenever the game is in state $q$, the agents can choose moves so that $q'$ is a next state. A computation of $S$ is an infinite sequence $\lambda = q_0, q_1, q_2, \ldots$ of states such that for all positions $i \geq 0$, the state$q_{i+1}$. is a successor of the state $q_i$. We refer to a computation $\lambda$ and a position $i \geq 0$, we use $\lambda [i], \lambda [0, i]$ to denote $i$-th state of $\lambda$ and the finite prefix $q_0, q_1, \ldots, q_i$ of $\lambda$ respectively.

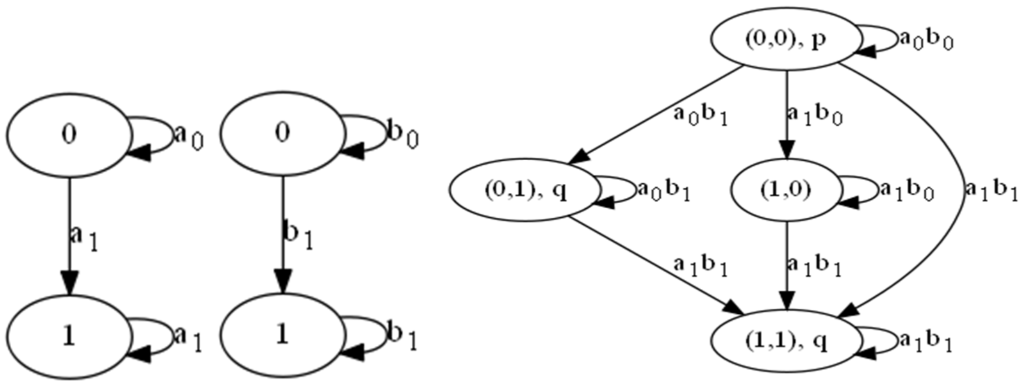


**Figure 1.** An example system: two agents (left) and the corresponding CGS (right). Source: own study

***Example 1.*** In Fig. 1 (left) an example system consisting of two agents is depicted. Every agent has two (local) states $0, 1$ and three transitions labeled by two actions $a0, a1$ for agent 0, and $b0, b1$ for agent 1. The CGS is shown in Fig. 1 (right). Thus, the set of global states is $Q = \{(0,0), (0,1), (1,0), (1,1)\}$. Let the initial state be $\tau = (0,0)$, and we have two propositional variables $\Pi = \{p, q\}$. Let $\pi(p) = \{(0,0)\}, \pi(q) = \{(0,1), (1,1)\}$, thus $p$ is true in state (0,0)and $q$ is true in states (0,1), and (1,1). The available moves and transitions are depicted as arrows. For example, $d_0((0,0)) = \{0,1\}, d((0,1)) = \{(0,1), (1,1)\}$, and $\delta((0,0), 1, 1) = (1,1)$. Note that besides numbers we additionally label actions with letters assigned to the consecutive agents.

## 3. Alternating-time Temporal Logic

Before we give formal definitions, we proceed with intuitions behind ATL. A strategy of an agent is a plan describing what agent can do in each situation (state). Every agent can base its decisions only on the current state. A strategy for a group of agents is a tuple of individual strategies. A formula $\ll A \gg X\, \alpha$ means that the group $A$ has a strategy to make $\alpha$ true in the next step. $\ll A \gg G\, \alpha$ means that the group $A$ can cooperate in a way that $\alpha$ is always true. $\ll A \gg \alpha\, U\, \beta$ means that the group $A$ can enforce $\beta$, but until that happens $\alpha$ is true.

**Definition 2 (ATL syntax).** The set of ATL formulas is defined as follows:

- $p \in \Pi$ is a formula,
- if $\alpha, \beta$ are formulas, then $\neg \alpha, \alpha \vee \beta$ are also formulas,
- if $A \subseteq \{1, \dots, k\}$ is a set of agents and $\alpha, \beta$ are formulas, then $\ll A \gg X\, \alpha, \ll A \gg \alpha\, U\, \beta$, and $\ll A \gg G\, \alpha$ are also formulas.

Additionally, the Boolean operators $\wedge, \Rightarrow, \Leftrightarrow$ are defined by the operators $\neg, \vee$. The temporal operator eventually is defined as $F\, \alpha = true\, U\, \alpha$. The ATL formulas are interpreted over the states of CGS. In order to define the semantics formally, we first define the notion of strategies. A strategy for an agent $a$ is a function $f_a$ that maps every nonempty finite state sequence $\lambda \in Q^+$ to a natural number such that if the last state of $\lambda$ is $q$, then $f_a(\lambda) \leq d_a(q)$. Thus, the strategy $f_a$ determines for every finite prefix $\lambda$ of a computation a move $f_a\lambda$ for agent $a$. Each strategy $f_a$ for agent a induces a set of computations that agent $a$ can enforce. Given a state $q \in Q$, a set $A$ of agents, and a set $F_A = \{f_a | a \in A\}$ of strategies, one for each agent in $A$, we define the outcomes of $F_A$ from $q$ to be the set $out(q, F_A)$ of $q$-computations that the agents in $A$ enforce when they follow the strategies in $F_A$; that is a computation $\lambda = q_0, q_1, q_2, \dots$ is in $out(q, F_A)$ if $q_0 = q$ and for all positions $i \geq 0$, there is a move vector

$h_{j_1}, \ldots, j_{k_i} \in d(q_i)$ such that $j_a = f_a(\lambda[0,i])$ for all agents $a \in A$, and $\delta(q_i, j_1, \ldots, j_k) = q_i + 1$.

**Definition 3 (Interpretation of ATL).** Let S be a CGS, q ∈ Qa state, and α , β formulas of ATL. S, q ⊨ α denotes that α is true at the state q in the structure S. S is omitted, if it is implicitly understood. The relation ⊨ is defined inductively as follows:

- $q \vDash p$ iff $p \in \pi\,(q)$, for $p \in \Pi$ ,
- $q \vDash \neg\,\alpha$ iff $q \neg \vDash \alpha$,
- $q \vDash \alpha \vee \beta$ iff $q \vDash \alpha$ or $q \vDash \beta$,
- $q \vDash \ll A \gg X\,\alpha$ iff there is a set $F_A$ of strategies for each agent in $A$, such that for all computations $\lambda \in out(q, F_A)$, we have $\lambda[1] \vDash \alpha$,
- $q \vDash p \ll A \gg G\,\alpha$ iff there is a set $F_A$ of strategies for each agent in $A$, such that for all computations $\lambda \in out(q, F_A)$,, and all positions $i \geq 0, \lambda[i] \vDash \alpha$,
- $q \vDash \ll A \gg \alpha\, U\, \beta$ iff there is a set $F_A$ of strategies for each agent in $A$, such that for each computation $\lambda \in out(q, F_A)$,, there is a position $i \geq 0$ such that $\lambda[i] \vDash \beta$ and for all positions $\lambda[i] \vDash \beta$, we have $\lambda[j] \vDash \alpha$.

**Definition 4 (Validity)**. An ATL formula ϕ is valid in S iff S, τ ⊨ ϕ.

***Example 2.*** Consider the CGS of Example 1 and the formula $\phi = \ll 0{,}1 \gg Xp$ , which means that the group of agents $A = \{0{,}1\}$ has a strategy that $p$ will be true in the next step. The initial state of the system is $\tau = (0{,}0)$ and there exists a transition changing state (0,0) to state (0,0). Thus, there exists a strategy for the agents to enforce p in the next step: both should perform action 0 in the initial state, so the given formula is valid, whereas the formula $\psi = \ll 0{,}1 \gg X\,(p \wedge q)$ is not valid in this CGS.

## 4. Fixed point representation of ATL and QBF encoding

In this section we briefly recall from [21] how the UMC can be used for ATL verification. The crucial point is encoding the *next* operator as a Quantified Boolean Formula. QBF is an extension of propositional logic by means of quantifiers ranging over propositions. The semantics is as follows:

- $\exists\, p, \alpha\; iff\; \alpha(p \leftarrow true\,) \vee \alpha\,(p \leftarrow false)$,
- $\forall\, p, \alpha\;\; iff\; \alpha(p \leftarrow true\,) \wedge \alpha\,(p \leftarrow false)$,

where $\alpha$ is a QBF formula, $p$ is a propositional variable, and $\alpha(\,p \leftarrow \Psi\,)$ stands for a substitution of every occurrence of the variable $p$ with $\Psi$ in formula $\alpha$.

The other modal operators are computed as the greatest or least fixed points of functions defined over the basic *next* operator. In order to obtain fixed-point characterizations of operators, we identify each ATL formula $\alpha$ with the set $\langle\alpha\rangle_S$ of states in $S$ at which this formula is true that is $\{q \in Q : S, q \models \alpha\}$. If $S$ is known from the context we omit the subscript $S$. Furthermore, we define functions $\ll A \gg X(Z)$ for every $A \subseteq \{0, \ldots, k-1\}$ as follows:

- $\ll A \gg X(Z) = \{q \in Q :$ for every $a \in A$ there exists a natural number $\$ \leq d_a(q)$ such that for every state $q' \in Q$, every agent $b \in \{0, \ldots, k-1\} \setminus A$ and every natural number $j_b \leq d_b(q)$ if $q' = \delta(q, j_0, \ldots, j_{k-1})$then $q' \in Z$.

We assume a set of agents $\{0, \cdots, k-1\}$, a set of global states $Q$, sets of possible actions $Act_a$for each agent $a$, and a set of protocols $P_a: Q \mapsto 2^{Act_a}$ that indicate which actions can be executed in which states. All actions are defined by means of $pre$ and $post$ conditions, i.e., for action $c$, $pre(c)$ is a set of all states from which action $c$ can be executed and $post(c)$ is a set of all states which can be reached after the execution of action $c$. Furthermore, we assume that for every state $q$ and $c_0 \in P_0(q), \cdots, c_{k-1} \in P_{k-1}(q)$ there exists exactly one state $q'$ such that $q' \in post(c_0) \cap \cdots \cap post(c_{k-1})$. Next, we define the function $\delta$ that assigns state $q' \in post(c_0) \cap \cdots \cap post(c_{k-1})$ to every tuple $(q, c_0, \cdots, c_{k-1})$ such that $q \in Q$ and $c_a \in P_a(q)$ for $a = 0, \cdots, k-1$. Given such a description of a system it is easy to build the corresponding concurrent game structure by taking $|P_a(q)| = d_a(q)$and numbering actions belonging to the set $P_a(q)$for every state $q$ and agent $a$. Next, in order to symbolically represent the (sets of) states, we assume $Q \subseteq \{0,1\}^m$, where $m = \lceil log_2(|Q|) \rceil$. Let $PV$ be a set of fresh propositional variables such that $PV \cap \Pi = \emptyset$ . Then, each state $q \in Q$ is represented by a global state variable $w = (w[0], \cdots, w[m-1])$, a vector of propositions, where $w[i] \in PV$ for each$i = 0, \cdots, m-1$. Let $FPV$ be a set of propositional formulas over $PV$, and let $lit : \{0,1\} \times PV \mapsto FPV$ be a function defined as follows: $lit(0,p) = \neg p$and$lit(1,p) = p$. Furthermore, let $w$ be a global state variable. We define the following propositional formulas:

- $I_q(w) := \bigwedge_{i=0}^{m-1} lit(q[i], w[i])$; this formula encodes the state $q$ over the vector $w$. In fact, a state is represented as a binary number.
- $pre_c(w)$and $post_c(w)$ for every $c \in Act_1 \cup \cdots \cup Act_k$; $pre_c(w)$ is a formula which is true for valuation $q = (q[0], \cdots, q[m-1])$of $w = (w[0], \cdots, w[m-1])$ iff and $post_c(w)$ is a formula which is true for valuation $q$ of $w$ iff $q \in post(c)$.

Next, we translate ATL formulas into QBF formulas. Specifically, for a given ATL formula $\phi$we compute a corresponding propositional formula $[\phi](w)$which is satisfied by a valuation $q$ of $w$ iff $q \in \langle\phi\rangle$ . In so doing we obtain a formula $[\phi](w)$ such that $\phi$ is valid in the structure $S$ iff the conjunction $[\phi](w) \wedge I_\tau(w)$is satisfiable. Operationally, we work outwards from the most nested subformulas, i.e., to compute $[O\alpha](w)$, where $O$ is a modality, we work under the assumption of already having computed $[\alpha](w)$. The translation is as follows:

- $[p](w) := \bigvee_{q \in <p>} I_q(w)$, for $p \in \Pi$,
- $[\neg \alpha](w) := \neg [\alpha](w)$,
- $[\alpha \vee \beta](w) := [\alpha](w) \vee [\beta](w)$,
- let $A = \{a_1, \dots, a_t\} \subseteq \{1, \dots, k\}$ and let $B = \{b_1, \dots, b_s\} \subseteq \{1, \dots, k\} \setminus \{a_1, \dots, a_t\}$,
  $[\ll A \gg X\alpha](w) := \bigvee_{c_{a1} \in Act_{a1}, \dots, c_{at} \in Act_{at}} \quad (\bigwedge_{i=1}^{t} pre_{cb_j} \quad (w) \vee forall(v, \bigwedge c_{b1}$

  $\in Act_{b1}, \dots, c_{bs} \in Act_{bs}$ (

  $\bigwedge_{j=1}^{s} pre_{cb_j}(w) \wedge \bigwedge_{j=1}^{s} post_{cb_j}(v) \quad \bigwedge_{i=1}^{t} post_{ca_i}(v) \Rightarrow [\alpha](v)))$

- $[\ll A \gg G\alpha](w) := gfp_A([\alpha](w))$,
- $[\ll A \gg \alpha U\beta](w) := lfp_A([\alpha](w), [\beta](w))$.

where $gfp$ and $lfp$ are based on the standard procedures computing fixed points. See [21], Sec. 6 for more details.

***Example 3.*** Consider the CGS from Example 1 and the formula $\ll 0 \gg X\, p_0$. We have:

- $pre(a_0) = \{ (0,0), (0,1)\}, pre_{a_0}(w) = (\neg w[0] \wedge \neg w[1]) \vee (\neg w[0] \wedge w[1])$,
- $pre(a_1) = \{ (0,0), (0,1), (1,0), (1,1)\}, pre_{a_1}(w) = (\neg w[0] \wedge \neg w[1]) \vee (\neg w[0] \wedge w[1]) \vee (w[0] \wedge \neg w[1]) \vee (w[0] \wedge w[1])$
- $post(a_1) = \{(1,0), (1,1)\}, post_{a_1}(w) = (w[0] \wedge \neg w[1]) \vee (w[0] \wedge w[1])$.

## 5. Implementation

The model-checking method described above has been implemented in C# language as a tool UMC4ATL. It performs a translation of the verification problem to a QBF formula, and it uses Z3 [15] to check for its satisfiability. Z3 theory prover is a software that supports Boolean logic, arithmetic, data types, quantifiers and more. It is often applied to solve hard problems from various domains, like model checking and planning.

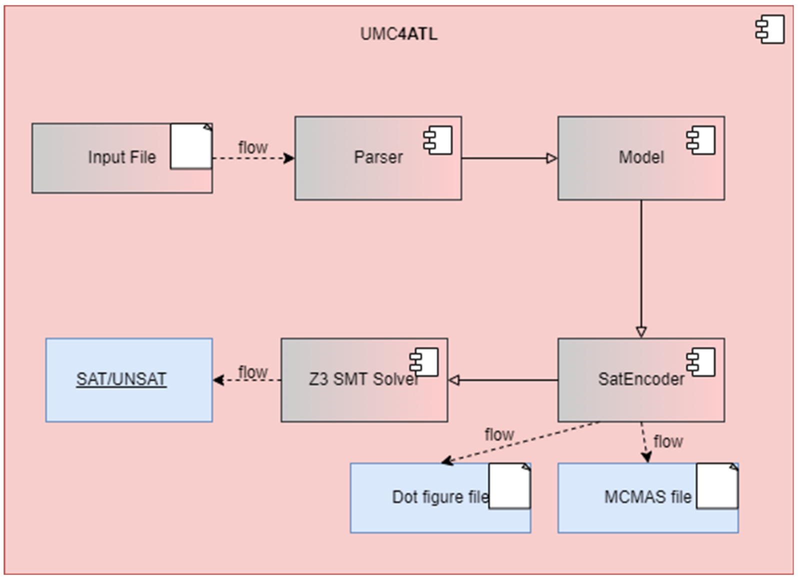


**Figure 2.** UMC4ATL architecture. Source: own study

Fig. 2 shows the overall architecture of UMC4ATL application. The very first element needed for execution of our software is an input file with agent specifications and an ATL formula to be verified. Its structure is discussed in Example 4. The *Parser* module reads the content of the input file, checking its correctness and consistency, and then transforms it into a set of corresponding objects stored in system memory and the formula into a binary tree. Then, the *Model* module builds the graph product and computes the *pre* and *post* sets. Such prepared model objects are ready to be consumed by the *Encoder* module. In this step, if the appropriate options are set, we are translating the model into an MCMAS input file, and into DOT format to visualize the agents and CGS. Finally, we are translating the verification problem into a QBF formula to check it for satisfiability using Z3 solver.

## 6. Experimental Results

In this section we report our preliminary experimental results compared with the state-of-the-art BDD-based model checker MCMAS [26, 27]. The experiments were performed using a PC equipped with AMD Ryzen 5 3600X CPU and 32 GB RAM running under Windows 10 OS. We used two scalable benchmarks known from literature: Train Gate Controller [32], and Castles game [31].

## 6.1. Train Gate Controller

The Train Gate Controller scenario [32] considers a number of trains trying to access a tunnel, whose entrance is managed by a controller. The controller allows only one train in the tunnel at any time. Fig. 3 shows the TGC system with two trains.

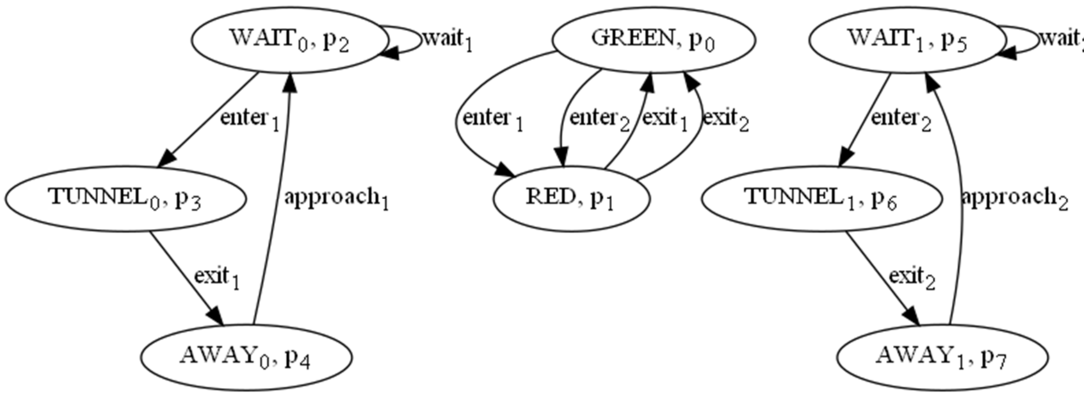


**Figure 3.** Train Gate Controller system with two trains (left and right). Source: own study

The Controller has two states, displaying a green light when the tunnel is empty, and the red light, when a train is in the tunnel. These states are marked with propositions $p_0$, and $p_1$, respectively. A train can be in one of the three states. It can be in front of the tunnel waiting for the green light, or it can be in the tunnel, or it could be away after traversing the tunnel. Due to synchronization of the appropriate actions, in Fig. 3 marked with the same labels, the Controller admits only one of the waiting trains at a time.

The meaning of propositions is as follows. $p_0$ means that controller displays a green light and $p_1$ means that controller displays a red light. Variables from $p_2$ to $p_{2+(3n-1)}$ mark states for each train, where $n$ is the number of trains. For example $p_2$ - Train1 is waiting, $p_3$ - Train1 is in tunnel, $p_4$ - Train1 lefts the tunnel. Thus, when we divide the consecutive numbers corresponding to propositions by 3, if the remainder is 2 the train is in waiting state, when the remainder is 0 the train is in tunnel, and if the remainder is 1 it means that the train lefts the tunnel.

**Table 1.** The ATL formulas checked against TGC specification. The column meaning from left to right: formula id, the formula, the number of nested strategic operators, the number of different coalitions in the formula, the number of subformulas, the total length of the formula. Source: own study

| No. | Formula | Depth | Coals. | Subf. | Length |
|---|---|---|---|---|---|
| 1 | $\ll 0,1,2 \gg Fp_0$ | 1 | 1 | 1 | 11 |
| 2 | $\ll 0,1,2 \gg Fp_3$ | 1 | 1 | 1 | 11 |
| 3 | $\ll 0,1,2 \gg F(p_0 \wedge (p_3 \wedge p_6))$ | 1 | 1 | 1 | 19 |
| 4 | $\ll 0,1,2 \gg F(p_3 \wedge \ll 0,1,2 \gg Fp_6)$ | 2 | 1 | 2 | 23 |
| 1R | $\ll 0,1 \gg F(p_0 \wedge (p_1 \wedge p_2))$ | 1 | 1 | 1 | 17 |
| 2R | $(\ll 0 \gg X(p_0 \wedge (p_1 \wedge p_2)) \wedge \ll 1 \gg X\ldots$ | 1 | 2 | 2 | 36 |
| 3R | $\ll 0,1,2 \gg X\neg p_0$ | 1 | 1 | 1 | 12 |
| 4R | $(((p_3 \vee \ll 1 \gg F(p_1 \vee \neg \ll 2 \gg G\neg \ll 1 \gg F\ldots$ | 5 | 5 | 10 | 99 |
| 1T | $\ll 0,1,2 \gg X(p_0 \wedge (p_4 \wedge p_5))$ | 1 | 1 | 1 | 19 |
| 2T | $\ll 0,1,2 \gg F(p_0 \wedge (p_4 \wedge p_5))$ | 1 | 1 | 1 | 19 |
| 3T | $\ll 0,1,2 \gg G(p_0 \wedge (p_4 \wedge p_5))$ | 1 | 1 | 1 | 19 |
| 4T | $\ll 0,1,2 \gg F(p_1 \wedge \ll 1,2 \gg F(p_2 \wedge p_6))\ldots$ | 3 | 3 | 3 | 44 |
| 5T | $((p_1 \wedge (\ll 0,2 \gg F\neg p_3 \wedge p_5)) \wedge \ldots$ | 2 | 2 | 3 | 41 |

Table 1 presents formulas that have been checked against TGC specification. They are divided into three groups. The formulas 1-4 was tested in the presence of only 3 propositions. For example, Formula 2 expresses that all agents have a common strategy to ensure that eventually Train1 will be in tunnel, while Formula 4 means that for agents 0,1,2 there is a strategy that eventually Train1 will be in the tunnel and then a strategy that eventually Train2 will be in the tunnel. The most interesting, however, is Formula 3, which we use to test whether there may be a situation that there is a green light and both trains are in the tunnel. The next group, the formulas 1R-4R, constitute random generated ATL formulas. Due to the lack of space, in the case of long formulas, we only show their beginning. The third group, the formulas 1T-5T, have been tested in presence of all propositions shown in Fig. 3. For example, formula 1T expresses that there is a strategy for agents 1,2,3, allowing them to achieve, in one step, the state in which the first train is away, while the second train is waiting in front of tunnel. Formula 2T expresses a similar property, but does not require reaching such a state in one step, only at some time in the future. As will be shown next, Formula 2T is valid, while Formula 1T does not hold in the TGC model.

Table 2 presents the results of Experiment 1 for TGC with 2,3,4 and 5 trains. The column meaning is as follows. Column "No." represents the formula id, column "Sat" informs if the formula is satisfiable ( Y-yes, N-no), "UMC" shows time consumed by UMC4ATL (in seconds), while the column "MCMAS" shows run-time of MCMAS. The numbers in bold show the cases where the UMC4ATL has an advantage over the MCMAS. In most cases, the performance of both tools was similar, but as the state space increased, in many cases MCMAS outperformed our tool.

**Table 2.** Results of Experiment 1, for TGC with 2-5 trains. Source: own study

| | | TGC2 | | TGC3 | | TGC4 | | TGC5 | |
|---|---|---|---|---|---|---|---|---|---|
| **No.** | **SAT** | **UMC** | **MCMAS** | **UMC** | **MCMAS** | **UMC** | **MCMAS** | **UMC** | **MCMAS** |
| 1 | Y | 0,0174 | 0,013 | 0,0201 | 0,020 | 0,1271 | 0,039 | 0,3711 | 0,045 |
| 2 | Y | 0,0174 | 0,013 | 0,0201 | 0,018 | 0,1022 | 0,029 | 0,3704 | 0,045 |
| 3 | N | **0,0141** | 0,016 | **0,0149** | 0,023 | **0,0167** | 0,037 | **0,0223** | 0,035 |
| 4 | Y | 0,0248 | 0,012 | 0,2345 | 0,023 | 0,8891 | 0,030 | 5,154 | 0,043 |
| 1R | N | 0,0139 | 0,013 | **0,0146** | 0,023 | **0,0165** | 0,034 | **0,0224** | 0,150 |
| 2R | N | 0,0134 | 0,012 | **0,0148** | 0,020 | 1,3210 | 0,022 | 6,072 | 0,042 |
| 3R | N | **0,0122** | 0,017 | **0,0136** | 0,024 | 0,0751 | 0,025 | 0,2397 | 0,047 |
| 4R | N | 0,0130 | 0,013 | 0,0632 | 0,019 | 1,5316 | 0,020 | 5,1930 | 0,036 |
| 1T | N | 0,0130 | 0,013 | **0,0131** | 0,016 | 0,1040 | 0,026 | 0,3512 | 0,036 |
| 2T | Y | 0,0158 | 0,012 | 0,0188 | 0,018 | 0,1334 | 0,084 | 0,2830 | 0,047 |
| 3T | N | 0,0181 | 0,012 | 0,0191 | 0,018 | **0,022** | 0,026 | 0,0670 | 0,043 |
| 4T | Y | 0,0245 | 0,017 | 0,0255 | 0,023 | 1,3923 | 0,025 | 6,4358 | 0,04 |
| 5T | N | 0,0301 | 0,012 | 0,0332 | 0,019 | 0,6061 | 0,0232 | 2,4695 | 0,045 |

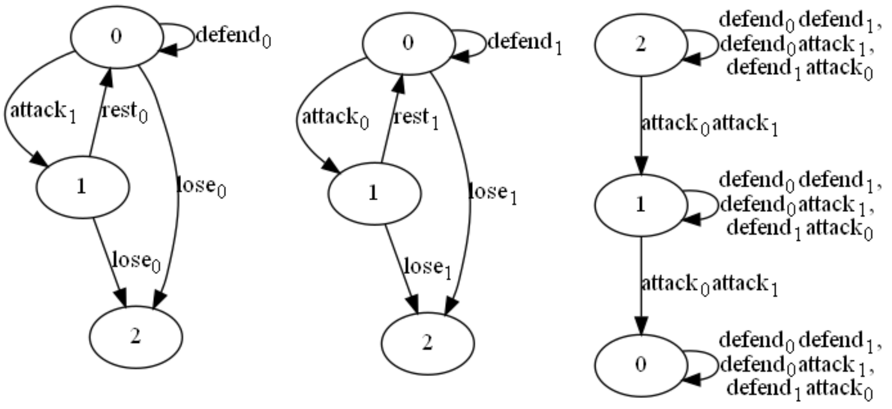
## 6.2. Castles game

The second benchmark for testing the performance of the UMC4ATL application is a version of Castles game [31]. In this example we have two parameters: N - the number of castles, and HP - the number of hit points per each castle. Every castle has a knight, which can either defend own castle, or attack another castle. However, after every attack, the knight has to return to the castle for rest. While resting, the knight defends the castle as well. Every attack of a single knight decreases the castle's HPs by 1, unless a knight is defending the attacked castle, what reduces the taken damage by 1. Thus, for example, if two knights attack the same undefended castle, it takes 2 HPs down. If the castle is defended, the same double attack reduces castle's HPs by 1 point. In order to keep track of the castles' hit points, we introduced another agent, called Counter. Its states correspond to the possible combinations of HPs of every castle.

Fig. 4 shows the system for 2 castles and 2 HPs per each. Both players start with 2 HPs, so the initial state of the system is $(0, 0, 2)$. For better readability we show counter's actions labelled by synchronized pairs of the knights' actions. For two knights, there are only four possible outcomes of the first round of game:

- the knights defend their castles, and the state of the HPs does not change; it corresponds to the move $(defend_0, defend_1)$,

- the first knight attacks the second one which defends itself; this is the move $(attack_1, defend_1)$,
- the second knight attacks the other which defends; $(defend_0, attack_0)$,
- both knights attack each other simultaneously and both lose one hit point. This is the only case in this setup to change the game result. It corresponds to the move $(attack_0, attack_1)$.



0
defend$_0$
attack$_1$
rest$_0$
1
lose$_0$
lose$_0$
2
0
defend$_1$
attack$_0$
rest$_1$
1
lose$_1$
lose$_1$
2
2
defend$_0$ defend$_1$, defend$_0$ attack$_1$, defend$_1$ attack$_0$
attack$_0$ attack$_1$
1
defend$_0$ defend$_1$, defend$_0$ attack$_1$, defend$_1$ attack$_0$
attack$_0$ attack$_1$
0
defend$_0$ defend$_1$, defend$_0$ attack$_1$, defend$_1$ attack$_0$

**Figure 4.** Castles game with two knights (left, center), and the counter (right). Each castle has initially 2 HPs. Source: own study

For this example, the formula $\ll 0 \gg G \neg\, lose_0$ was tested, which means that the first knight has a strategy to never lose. Table 3 shows the results of Experiment 2. The first column shows the number of hit points of every castle, while the next columns present the execution time of UMC4ATL and MCMAS (in seconds). This benchmark shows that the UMC4ATL is not coping well with the increase in the number of castles and their hit points, while MCMAS is more efficient here.

**Table 3.** Results of Experiment 2. Castles game for 2 and 3 knights with 1,2,3 HPs

| | N=2 | | N=3 | |
|---|---|---|---|---|
| **HP** | **UMC** | **MCMAS** | **UMC** | **MCMAS** |
| 1 | 0,036 | 0,031 | 0,146 | 0,025 |
| 2 | 0,0553 | 0,031 | 1,295 | 0,124 |
| 3 | 0,063 | 0,048 | 13,951 | 0,254 |

## 7. Comparison of local and global valuation

In our tool we implemented two types of evaluation: global, which assigns propositions to global states, and local - assigning propositions to local states. The comparison of both versions is presented in Table 4. The column meaning of Table 4 is as follows. Column "No." represents the formula id that are shown in Table 1 and TGC2-TGC5 represent the number of trains in a TGC game. Labels *glob* and *loc* stands respectively for global and local evaluation.

The numbers in bold show the few cases where the local evaluation has an advantage over global evaluation.

**Table 4.** Comparison of global and local valuation, for TGC with 2-5 trains. Source: own study

| | **TCG2** | | **TCG3** | | **TCG4** | | **TCG5** | |
|---|---|---|---|---|---|---|---|---|
| **No.** | **glob** | **loc** | **glob** | **loc** | **glob** | **loc** | **glob** | **loc** |
| 1 | 0,017 | 0,061 | 0,020 | 0,081 | 0,127 | 0,213 | 0,371 | 0,548 |
| 2 | 0,017 | 0,074 | 0,020 | 0,078 | 0,102 | 0,231 | 0,370 | 0,557 |
| 3 | 0,014 | 0,075 | 0,015 | 0,083 | 0,017 | 0,161 | 0,022 | 0,444 |
| 4 | 0,025 | 0,081 | 0,235 | **0,088** | 0,889 | **0,818** | 3,917 | 3,917 |
| 1R | 0,014 | 0,068 | 0,015 | 0,0767 | 0,017 | 0,102 | 0,022 | 0,199 |
| 2R | 0,013 | 0,068 | 0,015 | 0,075 | 1,321 | 1,324 | 6,072 | 6,609 |
| 3R | 0,012 | 0,073 | 0,014 | 0,791 | 0,075 | 0,162 | 0,239 | 0,414 |
| 1T | 0,013 | 0,069 | 0,013 | 0,074 | 0,104 | 0,191 | 0,351 | 0,511 |
| 2T | 0,016 | 0,075 | 0,019 | 0,081 | 0,133 | 0,168 | 0,208 | 0,454 |
| 3T | 0,018 | 0,073 | 0,019 | 0,081 | 0,022 | 0,115 | 0,067 | 0,244 |
| 4T | 0,025 | 0,082 | 0,026 | 0,090 | 1,392 | 1,479 | 6,436 | **6,335** |

## 8. Conclusion

In this paper we presented a prototype implementation of the Unbounded Model Checking for ATL introduced in [21]. We have translated the verification problem into a Quantified Boolean Formula and used the SMT-solver Z3 [15] to check its satisfiability. We have also compared the efficiency of our tool with the state-of-the-art MCMAS [26, 27] model checker. Our preliminary experimental results show that the UMC method works in a satisfactory way. However, we still need to introduce several optimizations and reductions - similar to those implemented in MCMAS - to improve the UMC4ATL efficiency. For example, MCMAS early prunes the unreachable fragments of the (local) state space in order to decrease the model size. Another improvement could be an on-the-fly symbolic encoding of the CGS without explicitly computing the product space. As a future work, we plan to introduce these optimizations along with checking whether the CNF translation originally proposed in [21] and the use of SAT-solvers would be more efficient than QBF solving.

## References

1. Alur R., Courcoubetis C.; Dill D.: Model-checking for real-time systems. In: [1990] Proceedings. Fifth Annual IEEE Symposium on Logic in Computer Science. IEEE, 1990. p. 414-425.

2. Alur R., Henzinger T. A., and Kupferman O.: Alternating-time temporal logic. In Proc. of the 38th IEEE Symp. on Foundations of Computer Science (FOCS'97), pages 100-109. IEEE Computer Society, 1997.

3. Alur R., Henzinger T. A., and Kupferman O.: Alternating-time temporal logic. LNCS, 1536:23-60, 1998.

4. Alur R., Henzinger T. A., and Kupferman O.: Alternating-time temporal logic. Journal of the ACM, 49(5):672-713, 2002.

5. Baier C. and Katoen J.: Principles of model checking. MIT Press, 2008.

6. Biere A., Cimatti A., Clarke E., and Zhu Y.: Symbolic model checking without BDDs. In International conference on tools and algorithms for the construction and analysis of systems, pages 193-207. Springer, 1999.

7. Biere A., Cimatti A., Clarke E., Strichman O., and Zhu Y.: Bounded model checking. 2003.

8. Browne M. C., Clarke E. M., Dill D. L., and Mishra B.: Automatic verification of sequential circuits using temporal logic. IEEE Transactions on Computers, C-35(12):1035-1044, 1986.

9. Bryant R.: Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers, C-35(8):677-691, 1986.

10. Cimatti A., Clarke E., Giunchiglia F., and Roveri M.: NuSMV: a new symbolic model verifier. In International conference on computer aided verification, pages 495-499. Springer, 1999.

11. Clarke E., Biere A., Raimi R., and Zhu Y.: Bounded model checking using satisfiability solving. Formal methods in system design, 19(1):7-34,2001.

12. Clarke E., and Emerson E.: Design and synthesis of synchronization skeletons using branching-time temporal logic. In Logic of Programs, pages 52-71, 1981.

13. Clarke Jr E. M., Grumberg O., Kroening D., Peled D., and Veith H.: Model checking. MIT press, 2018.

14. Davis M., Logemann G., and Loveland D.: A machine program for theorem-proving. Communications of the ACM, 5():394-397, 1962.

15. De Moura L. and Bjørner N.: Z3: An efficient SMT solver. In International conference on Tools and Algorithms for the Construction and Analysis of Systems pages 337-340. Springer, 2008.

16. Doijade M. M. and Kulkarni D. B.: Overview of sequential and parallel sat solvers. In International Conference of Information Communication adn Embedded Systems (ICICES2014), pages 1-4, 2014.

17. Jamroga W., Penczek W., Dembiński P., and Mazurkiewicz A.: Towards partial order reductions for strategic ability. In Proceedings of the 17th International Conference on Autonomous Agents and MultiAgents Systems, AAMAS'18, pages 156-165, 2018.

18. Jamroga W., Penczek W., Sidoruk T., Dembiński P., and Mazurkiewicz A.: Towards partial order reductions for strategic ability. Journal of Artificial Intelligence Research, 68:817-850, 2020.

19. Kacprzak M., Nabiałek W., Niewiadomski A., Penczek W., Półrola A., Szreter M., Woźna B., and Zbrzezny A.: Verics 2007 - a model checker for knowledge and real-time. Fundamenta Informaticae, 85(1-4):313-328, 2008.

20. Kacprzak M., Niewiadomski A., and Penczek W.: SAT-based ATL satisfiability checking. In D. Calvanese, E. Erdem, and M. Thielscher, editors, Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR 2020, pages 539-549, 2020.

21. Kacprzak M., and Penczek W.: Unbounded model checking for Alternating-Time Temporal Logic. In Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagents Systems, AAMAS 2004, pages 646-653, 2004.

22. Kacprzak M., and Penczek W.: Fully symolic unbounded model checking for alternating-time temporal logic. Auton. Agents Multi Agent Syst., 11(1):69-89,2005.

23. Knapik M., Niewiadomski A., Penczek W., Półrola A., Szreter M., and Zbrzezny A.: Parametric model checking with VerICS. In Transactions on Petri nets and other models of concurrency IV, pages 98-120. Springer, 2010.

24. Laroussine F., Markey N., and Oreiby G.: Model-checking timed ATL for durationalconcurrent games structures. In International Conference on Formal Modeling and Analysis of Timed Systems, pages 245-259. Springer, 2006.

25. Lions J.-L., Luebeck L., Fauquembergue J.-L., Kahn G., Kubbat W., Levedag S., Mazzini L., Merle D., and O'Halloran C.: Ariane 5 flight 501 failure report by the inquiry board, 1996.

26. Lomuscio A., Qu H., and Raimondi F.: Mcmas: A model checker for the verification of multi-agent systems. In International conference on computer aided verification, pages 682-688. Springer, 2009.

27. Lomuscio A., Qu H., and Raimondi F.: MCMAS: an open-source model checker for the verification of multi-agent systems. International Journal on Software Tools for Technology Transfer, 19(1):9-30, 2017.

28. Lomuscio A., and Raimondi F.: MCMAS: a model checker for multi-agent systems. In International Conference on Tools and Algorithms for the Construction and Analysis of Systems, pages 450-454. Springer, 2006.

29. McMillan K. L.. Symbolic model checking. Kluwer, 1993.

30. McMillan K. L.. Applying SAT methods in unbounded symbolic model checking. In International Conference on Computer Aided Verification, pages 250-264. Springer, 2002.

31. Pilecki J., Bednarczyk M. A., and Jamroga W.: SMC: synthesis of uniform strategies and verification of strategic ability for multi-agent systems. Journal of Logic and Computation, 27(7):1871-1895, 2017

32. Van der Hoek W. and Wooldridge W.: Cooperation, knowledge, and time: Alternating-time temporal epistemic logic and its applications. Studia logica, 75(1):125-157, 2003.

33. Vardi M. Y.: Automata-theoretic model checking revisited. In International Workshop on Verification, Model Checking, and Abstract Interpretation, pages 137-150. Springer, 2007.

34. Vizel Y., Weissenbacher G., and Malik S.: Boolean satisfiability solvers and their applications in model checking. Proceedings of the IEEE, 103(11):2021-2035, 2015.

35. Wang J., Wang M., Zheng K., and Huang X.: Model checking nrf24101-based internet of things systems. In 9th International Conference of Information Technology in Medicine and Education (ITME), pages 867-871, 2018.

36. Zhang X. and van Breugel F.: Model checking randomized allgorithms with Java PathFinder. In Seventh International Conference on the Quantitative Evaluation of Systems, pages 157-158, 2010.