Unbounded Model Checking for Alternating-Time Temporal Logic

M. Kacprzak
Białystok University of Technology
Institute of Mathematics and Physics
e-mail: mdkacprzak@wp.pl

W. Penczek
Institute of Computer Science, PAS
and
Podlasie Academy,
Institute of Informatics, Siedlce, Poland
email: penczek@ipipan.waw.pl

Abstract

This paper deals with the problem of verification of game-like structures by means of symbolic model checking. Alternating-time Temporal Logic (ATL) is used for expressing properties of multi-agent systems represented by concurrent game structures. Unbounded model checking (a SAT based technique) is applied for the first time for verification of ATL. An example is given to show an application of the technique.

1. Introduction

In 1997 R. Alur, T. A. Henzinger and O. Kupferman proposed a succinct and expressive language for reasoning about game-like systems, called Alternating-Time Temporal Logic (ATL) [2]. ATL is a generalization of the branching time logic CTL, where the path quantifiers are replaced by cooperation modalities of the form $\langle A \rangle$ with $A$ being a set of agents. The intended interpretation of an ATL formula $\langle A \rangle \psi$ is that group $A$ of agents has a winning strategy for $\psi$, i.e., the agents of $A$ can cooperate to ensure that $\alpha$ holds. For example, $\langle A \rangle \models_\alpha$ specifies that a coalition $A$ of agents has a strategy that can maintain that in a next state $\alpha$ is true.

Model checking is a powerful technique, widely used in verification of hardware and software systems [4]. It consists in automatic determining whether a given formula representing a property is satisfied in a particular model representing the executions of a system. Our paper deals with the problem of model checking multi-agent systems with respect to their specifications written in ATL.

Since exhaustive state-space exploration meets intrinsic difficulties, symbolic techniques based on a canonical form for boolean expressions (binary decision diagrams or ordered binary decision diagrams, BDD or OBDD’s in short) as well as propositional decision procedures (SAT) are investigated. OBDD’s have traditionally been used as the underlying representation for symbolic model checkers.

The authors of ATL designed a model-checker MOCHA [1] based on BDDs, which supports the heterogenous modelling framework of Reactive Modules. Unlike MOCHA, we focus on propositional decision procedures. These also operate on boolean expressions, but do not use canonical forms and therefore sometimes do not suffer from a potential space explosion of OBDD’s and excessive memory requirements.

In 2002 K. McMillan introduced a SAT based technique, called Unbounded Model Checking (UMC) [13], designed for verifying CTL properties. UMC consists in translating the model checking problem of a CTL formula into the problem of satisfiability of a corresponding propositional formula. It exploits the characterization of the basic CTL modalities in terms of Quantified Boolean Formulas (QBF), and the algorithms that translate QBF and fixed point equations over QBF into propositional formulas. Model checking via UMC can be exponentially more efficient than approaches based on BDD’s in two situations: whenever the resulting fixed-points have compact representations in CNF, but not via BDD’s and whenever the SAT-based image computation step proves to be faster than the BDD-based one [13].

The contribution of this paper is to show that UMC can be applied for verification of ATL. The key issue in solving this problem consists in encoding the “next time” operator $\langle A \rangle \models_\alpha$ by a QBF formula and next 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. In order to adapt UMC for checking ATL, we use three
algorithms. The first one, implemented by the procedure \textit{forall} (based on the Davis-Putnam-Logemann-Loveland approach) eliminates the universal quantifier from a QBF formula representing an ATL formula $⟨A⟩ \overline{α}$, and returns the result in \textit{Conjunctive Normal Form (CNF)}. The remaining algorithms calculate the greatest and the least fixed points. Ultimately, the technique allows for an ATL formula $α$ to be translated into a propositional formula $[α](w)\textsuperscript{1}$, which characterizes all the states of the model, where $α$ holds.

The rest of the paper is organized as follows. In Section 2 we define concurrent game structures (CGS). Then, in Section 3 the language of ATL is given. Section 4 reviews Quantified Boolean Formulas and formulas in Conjunctive Normal Form which are used in verification. Fixed-point characterization of temporal operators is described in Section 5. Section 6 deals with Unbounded Model Checking (UMC) for ATL. Finally, in Section 7 we show an example of applications of UMC. Conclusions are given in Section 8.

### 1.1. State of the art and related literature

The recent developments in the area of model checking MAS can broadly be divided into streams: in the first category standard predicates are used to interpret the various intensional notions and these are paired with standard model checking techniques based on temporal logic. Following this line is for example [24] and related papers. In the other category we can place techniques that make a genuine attempt at extending the model checking techniques by adding other operators. Works along these lines include [19, 20, 14, 17].

In [15, 12] and [16], presented at AAMAS’03, an extension of the method of bounded model checking (one of the main SAT-based techniques) to CTLK, a language comprising both CTL and knowledge operators, was defined, implemented, and evaluated. Quite recently, unbounded model checking method for CTLK has been defined and implemented [10, 11]. Since preliminary results appear largely positive [11], we believe that application of UMC to ATL will be efficient as well.

Recently, ATL has received a lot of interest in MAS community. There have been several papers considering ATL extensions by an epistemic component [21, 7, 8]. This is another motivation for defining UMC for ATL.

### 2. Concurrent Game Structure

We model compositions of open systems as concurrent game structures, in which a state transition results from choices made by the system components and the environment, and represents simultaneous steps by the components and the environment. We adopt the definition given in [3], extended with the notion of an initial state.

**Definition 1** A concurrent game structure (CGS) is a seven-tuple $S = ⟨k, Q, ι, Π, π, d, δ⟩$ where:

- $k$ is a natural number defining the amount of agents;
- $Q$ is a finite set of (global) states,
- $ι ∈ Q$ is the initial state,
- $Π$ is a finite set of atomic propositions (also called observables),
- $π : Q \rightarrow 2^Π$ is a function that specifies which propositions are true in which states; this function is called labelling (or observation) function,
- $d$ move available at a state $q ∈ Q$ to an agent $a ∈ \{1, \ldots, k\}$ are identified with numbers $1, \ldots, d_a(q)$; so given a state $q$, a move vector at $q$ is a tuple $⟨j_1, \ldots, j_k⟩$ such that $j_a ≤ d_a(q)$ for every agent $a$; then $d$ is a mapping that assigns for every state $q$ the set $\{1, \ldots, d_1(q)\} × \cdots × \{1, \ldots, d_k(q)\}$ of move vectors,
- $δ$ is a transition function which assigns to each state $q ∈ Q$ and each move vector $⟨j_1, \ldots, j_k⟩ \in d(q)$ a state $δ(q, j_1, \ldots, j_k) ∈ Q$ that results from state $q$ if every agent $a ∈ \{1, \ldots, k\}$ chooses move $j_a$.

**Computations.** We say that a state $q'$ is a successor of a state $q$ if there is a move vector $⟨j_1, \ldots, j_k⟩ ∈ d(q)$ such that $q' = δ(q, j_1, \ldots, j_k)$. 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 \textit{computation} of $S$ is an infinite sequence $λ = q_0, q_1, q_2, \ldots$ of states such that for all positions $i ≥ 0$, the state $q_{i+1}$ is a successor of the state $q_i$. We refer to a computation starting at state $q$ as a q-computation. For a computation $λ$ and a position $i ≥ 0$, we use $λ[i], λ[0, i]$ to denote the $i$-th state of $λ$ and the finite prefix $q_0, q_1, \ldots, q_i$ of $λ$ respectively.

### 3. Alternating-time Temporal Logic

Now, we present a language, called \textit{Alternating-Time Temporal Logic (ATL)} [3], to represent and reason about concurrent game structures. Before defining syntax and semantics, we give an intuition behind its key constructs. ATL takes its inspiration from CTL. Thus, it contains all the conventional connectives and tense modalities. However, the path quantifiers used in CTL are replaced by cooperation operators parameterized with sets of agents. The ATL formula $⟨A⟩⟨A⟩ \overline{α}$, where $A$ is a group of agents, means that the
group $A$ has a collective strategy to force that $\alpha$ is true in a
next state. Similarly, $\langle A \rangle \Box \alpha$ means that the members of $A$
can work together to ensure that $\alpha$ is always true. The formula $\langle A \rangle \alpha \Box \beta$ means that $A$ can cooperate to ensure that $\beta$
will become true at some time in the future and at least un-
til this time $\alpha$ remains true.

**Definition 2 (Syntax of ATL) The set of ATL formulas
$\mathcal{FOM}$ is defined inductively as follows:**

- Every member $p$ of $\Pi$ is a formula,
- If $\alpha$ and $\beta$ are formulas, then so are $\neg \alpha$ and $\alpha \lor \beta$,
- If $A \subseteq \{1, \ldots, k\}$ is a set of agents, and $\alpha$ and $\beta$
are formulas, then so are $\langle A \rangle \Box \alpha$, $\langle A \rangle \Diamond \alpha$, and
$\langle A \rangle \alpha \Box \beta$.

Additional Boolean connectives $\land, \Rightarrow, \Leftrightarrow$ are defined
from $\neg, \lor$ in the usual manner. Moreover true $= p \lor \neg p$
for some $p \in \Pi$ and false $= \neg \text{true}$.

We interpret ATL formulas over the states of a concurrent
game structure $S = \langle k, Q, I, \Pi, \pi, d, \delta \rangle$ that has the
same propositions and agents. In order to define the semantics
of ATL 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
$\text{out}(q, F_A)$ of $q$-computations that the agents in $A$
enforce when they follow the strategies in $F_A$; that is a computation
$\lambda = q_0, q_1, q_2, \ldots$ is in $\text{out}(q, F_A)$ if $q_0 = q$ and for all
positions $i \geq 0$, there is a move vector $(j_1, j_2, \ldots) \in d(q_i)$
such that (1) $j_i = f_a(\lambda[0, i])$ for all agents $a \in A$, and (2)
$\delta(q_i, j_1, j_2, \ldots) = q_{i+1}$.

**Definition 3 (Interpretation of ATL) Let $S$ be a concurrent
game structure, $q \in Q$ a state, and $\alpha, \beta$ formulas of
ATL. $S, q \models \alpha$ denotes that $\alpha$ is true at the state $q$
in the structure $S$. $S$ is omitted, if it is implicitly understood.
The relation $\models$ is defined inductively as follows:**

- $q \models p$ iff $p \in \pi(q)$, for $p \in \Pi$,
- $q \models \neg \alpha$ iff $q \not\models \alpha$,
- $q \models \alpha \lor \beta$ iff $q \models \alpha$ or $q \models \beta$,
- $q \models \langle A \rangle \Box \alpha$ iff there exists a set $F_A$ of strategies,
one for each agent in $A$, such that for all computations
$\lambda \in \text{out}(q, F_A)$, we have $\lambda[1] \models \alpha$,
- $q \models \langle A \rangle \Box \beta$ iff there exists a set $F_A$ of strategies,
one for each agent in $A$, such that for all computations $\lambda \in
\text{out}(q, F_A)$, and all positions $i \geq 0$, we have $\lambda[i] \models \alpha$,
- $q \models \langle A \rangle \alpha \Box \beta$ iff there exists a set $F_A$ of strategies,
for each agent in $A$, such that for all computations $\lambda \in
\text{out}(q, F_A)$, and for all positions $0 \leq j < i$, we have
$\lambda[j] \models \alpha$.

**Definition 4 (Validity) An ATL formula $\varphi$ is valid in $S$
(denoted $S \models \varphi$) iff $S, i \models \varphi$, i.e., $\varphi$ is true at the initial
state of the model $S$.**

## 4. Quantified Boolean Formulas and CNF Formulas

In order to have a more succinct notation for complex op-
erations on Boolean formulas, we use Quantified Boolean Formulas (QBF), an extension of propositional logic by
means of quantifiers ranging over propositions. In BNF:

- $\alpha :::= p \mid \neg \alpha \mid \alpha \land \beta \mid \exists p.\alpha \mid \forall p.\alpha$.
- The semantics of the quantifiers is defined as follows:

- $\exists p.\alpha$ iff $\alpha(p \leftarrow \text{true}) \lor \alpha(p \leftarrow \text{false})$,
- $\forall p.\alpha$ iff $\alpha(p \leftarrow \text{true}) \land \alpha(p \leftarrow \text{false})$,

where $\alpha \in \text{QBF}$, $p \in \mathcal{PV}$ (a set of propositional variables)
and $\alpha(p \leftarrow \psi)$ denotes substitution with the formula $\psi$ of
every occurrence of the variable $p$ in formula $\alpha$.

We use the notation $\forall v.\alpha$, where $v = (v[1], \ldots, v[m])$
is a vector of propositional variables, to denote
$\forall v[1].\forall v[2].\ldots.\forall v[m].\alpha$.

We usually deal with formulas in conjunctive normal forms. A formula is in
Conjunctive Normal Form (CNF) if it is a conjunction of zero or more clauses where by a clause
we mean a disjunction of zero or more literals, i.e., propositional variables as well as negations of these.

In the following we show a standard polynomial algo-
rithm, which we use later, that given a propositional formula $\alpha$,
constructs a CNF formula which is unsatisfiable exactly when $\alpha$ is valid. The procedure works as follows. First of
all, for every subformula $\beta$ of formula $\alpha$, including $\alpha$, we introduce a distinct variable $l_\beta$. Furthermore, if $\beta$ is a variable,
then $l_\beta = \beta$. Next we assign a formula $CNF(\beta)$ to every
subformula $\beta$ according to the following rules:

- If $\beta$ is a variable then $CNF(\beta) = \beta$,
- If $\beta = \neg \phi$ then $CNF(\beta) = CNF(\phi) \land (l_\beta \lor l_\phi)$,
- If $\beta = \phi \lor \phi$ then $CNF(\beta) = CNF(\phi) \land (l_\beta \lor \neg l_\phi)$,
- If $\beta = \phi \land \phi$ then $CNF(\beta) = CNF(\phi) \land CNF(\phi) \land
(l_\beta \lor \neg l_\phi) \land (l_\phi \lor \neg l_\phi)$,
- If $\beta = \phi \rightarrow \phi$ then $CNF(\beta) = CNF(\phi) \land CNF(\phi)$,
- If $\beta = \phi \leftarrow \phi$ then $CNF(\beta) = CNF(\phi) \land CNF(\phi)$,
- If $\beta = \phi \leftarrow \phi$ then $CNF(\beta) = CNF(\phi) \land CNF(\phi)$,
- If $\beta = \phi \lor \phi$ then $CNF(\beta) = CNF(\phi) \lor CNF(\phi) \land
(l_\beta \lor \neg l_\phi) \land (l_\phi \lor \neg l_\phi)$.
It can be easily shown that the formula $\alpha$ is valid exactly when the CNF formula $CNF(\alpha) \land \neg \alpha$ is unsatisfiable.

What is important to us, is that for a given QBF formula $\forall v. \alpha$, we can construct a CNF formula equivalent to it by using the algorithm $forall$ [13].

Given a propositional formula $\alpha$ and a set of variables $v[1], ..., v[m]$ the algorithm $forall$ constructs a CNF formula $\chi$ equivalent to $\alpha$ and eliminates quantified variables on the fly. A description of this procedure is given below.

**procedure** $forall(v, \alpha)$, where $v = (v[1], ..., v[m])$ and $\alpha$ is a propositional formula

let $\phi = CNF(\alpha) \land \neg \alpha$, $\chi = true$, and $A = \emptyset$

repeat

if $\phi$ contains $false$, return $\chi$
else if conflict

analyse conflict and backtrack
else if current assignment satisfies $\phi$

build a blocking clause $\ell'$

remove variables of form $v[i]$ or $\neg v[i]$ from $\ell'$

add $\ell'$ to $\phi$ and $\chi$
else

choose a literal $l$ such that $l \notin A$ and $\neg l \notin A$ and add $l$ to $A$

The procedure works as follows. Initially the algorithm assumes an empty assignment $A$, a formula $\chi$ to be $true$ and $\phi$ to be a CNF formula $CNF(\alpha) \land \neg \alpha$. First, the procedure finds a satisfying assignment for $\phi$. The search of an appropriate assignment is based on the Davis-Putnam-Logemann-Loveland approach [5] which makes use of two techniques: Boolean constraint propagation (BCP) and conflict-based learning (CBL). The first builds an assignment $A_0$ which is an extension of the assignment $A$ and is implied by $A$ and $\alpha$. Next BCP determines the consequence of $A_0$. The following three cases may happen:

1. A conflict exists, i.e., there exists a clause in $\phi$ such that all of its literals are false in $A_0$. So, the assignment $A$ can not be extended to a satisfying one. If a conflict is detected the CBL finds the reason for the conflict and tries to resolve it. Information about the current conflict may be recorded as clauses, which are then added to the formula $\phi$ without changing its satisfiability. The algorithm then backtracks, i.e., it changes assignment $A$ by withdrawing one of the previous decisions.

2. A conflict does not exist and $A_\phi$ is total, i.e., the satisfying assignment is obtained. In this case we generate a new clause which is false in the current assignment $A_\phi$ (i.e., rules out the satisfying assignment) and whose complement characterizes a set of assignments falsifying the formula $\alpha$. This clause is called a blocking clause. The construction of this clause is given in [13]. Next the blocking clause is deprived of the variables either of the form $v[i]$ or the negation of these and then what remains is added to the formulas $\phi$ and $\chi$ and the algorithm again tries to find a satisfying assignment for $\phi$.

3. The first two cases do not apply. Then, the procedure makes a new assignment $A$ by giving a value to a selected variable.

On termination, when $\phi$ becomes unsatisfiable, $\chi$ is a conjunction of the blocking clauses and precisely characterizes $\forall v. \alpha$. For more details see [13] or [9, 10].

**Theorem 1** Let $\alpha$ be a propositional formula and $v = (v[1], ..., v[m])$ be a vector of propositions, then QBF formula $\forall v. \alpha$ is logically equivalent to CNF formula $forall(v, \alpha)$.

The proof of the above theorem follows from the correctness of $forall$ algorithm (see [13]).

5. Fixed-point representation of ATL

In this section we show how the set of states satisfying an ATL formula can be characterized as a fixed point of an appropriate function. We adapt definitions given in [4].

Let $S = (k, Q, \iota, \pi, \forall, \exists, \delta)$ be a concurrent game structure. Notice that the set $2^Q$ of all subsets of $Q$ forms a lattice under the set inclusion ordering. Each element $Q'$ of the lattice can also be thought of as a predicate on $Q$, where the predicate is viewed as being true for exactly the states in $Q'$. The least element in the lattice is the empty set, which we also refer to as false, and the greatest element in the lattice is the set $Q$, which we sometimes write as true. A function $\tau$ mapping $2^Q$ to $2^Q$ is called a predicate transformer. A set $Q' \subseteq Q$ is a fixed point of a function $\tau : 2^Q \rightarrow 2^Q$ if $\tau(Q') = Q'$.

Whenever $\tau$ is monotonic, i.e., $P_1 \subseteq P_2$ implies $\tau(P_1) \subseteq \tau(P_2)$, it has the least fixed point denoted $\mu Z.\tau(Z)$ and the greatest fixed point denoted $\nu Z.\tau(Z)$. When $\tau(Z)$ is also $\forall$-continuous, i.e., $P_1 \subseteq P_2 \subseteq \ldots \implies \tau(\bigcup_i P_i) = \bigcup_i \tau(P_i)$ then $\mu Z.\tau(Z) = \bigcup_{i>0} \tau^i(\text{false})$. When $\tau(Z)$ is also $\exists$-continuous, i.e., $P_1 \supseteq P_2 \supseteq \ldots \implies \tau(\bigcap_i P_i) = \bigcap_i \tau(P_i)$ then $\mu Z.\tau(Z) = \bigcap_{i\geq 0} \tau^i(\text{true})$ (see [18]).

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, formally $\langle \alpha \rangle_S = \{ q \in Q \mid S,q \models \alpha \}$. If $S$ is known from the context we omit the subscript $S$. Furthermore, we define functions $\langle A \rangle \circ (Z)$ for every $A \subseteq \{1, \ldots, k\}$ as follows:

- $\langle A \rangle \circ (Z) = \{ q \in Q \mid \text{for every } a \in A \text{ there exists a natural number } j_a \leq d_a(q) \text{ such that for every state } q' \in Q, \text{ every agent } b \in \{1, \ldots, k\} \setminus A \text{ and every } \text{natural number } j_b \leq d_b(q) \text{ if } q' = \delta(q, j_1, \ldots, j_k) \text{ then } q' \in Z \}$. 

Lemma 1 \( (\langle A \rangle \circ \alpha) = (\langle A \rangle \circ (\alpha)) \).

Proof: \( (\langle A \rangle \circ (\alpha)) = \{ q \in Q \mid \text{for every } a \in A \text{ there exists a natural number } j_a \leq d_a(q) \text{ such that for every state } q' \in Q, \text{ every agent } b \in \{1, \ldots, k\} \setminus A \text{ and every natural number } j_b \leq d_b(q) \text{ if } q' = \delta(q, j_1, \ldots, j_k) \text{ then } q' \in (\alpha) \} = \{ q \in Q \mid \text{for every } a \in A \text{ there exists a function } f_a : Q^+ \rightarrow N \text{ such that } f_a(q) = j_a \leq d_a(q) \text{ and for every state } q' \in Q \text{ if } q' = \delta(q, j_1, \ldots, j_k) \text{ then } q' \in (\alpha) \} = \{ q \in Q \mid \text{there exists a set } F_A \text{ of strategies, one for each agent in } A \text{ such that for every } \lambda \in \text{out}(q, F_A) \text{ and for every state } q' \in Q \text{ if } q' = \lambda[1] \text{ then } q' \in (\alpha) \} = \{ q \in Q \mid \text{there exists a set } F_A \text{ of strategies, one for each agent in } A \text{ such that for every } \lambda \in \text{out}(q, F_A), \lambda[1] \in (\alpha) = \{ q \in Q \mid \langle A \rangle \circ \alpha) = (\langle A \rangle \circ (\alpha) \} \).  □

Then, each of the following operators may be characterized as the least or the greatest fixed point of an appropriate monotonic (\cap-continuous or \cup-continuous) predicate transformer.

- \((\langle A \rangle \sqcap \alpha) = \nu Z. (\alpha) \cap (\langle A \rangle \circ (Z))\)
- \((\langle A \rangle \sqcup \beta) = \mu Z. (\beta) \cup (\langle A \rangle \circ (\iota))\)

6. Symbolic Unbounded Model Checking on ATL

The UMC technique is based on forall procedure that allows us to eliminate quantifiers in QBF formulas as well as standard fixed-point algorithms, both used for translation of ATL formulas into propositional formulas. Then, it is combined with SAT-solvers to perform satisfiability checking.

We assume a set of agents \(\{1, \ldots, k\}\), a set of global states \(Q\), sets of possible actions \(\text{Act}_a\) for each agent \(a\), and a set of protocols \(P_a : Q \rightarrow 2^{\text{Act}_a}\) that indicate which actions can be executed in which states. All actions are defined by means of \(\text{pre}\) and \(\text{post}\) conditions, i.e., for action \(c\), \(\text{pre}(c)\) is a set of all states from which action \(c\) can be executed and \(\text{post}(c)\) is a set of all states which can be reached after the execution of action \(c\). This means that action \(c\) can be executed at any state in \(\text{pre}(c)\) and takes any state in \(\text{post}(c)\). Furthermore, we assume that for every state \(q\) and \(c_1 \in P_1(q), \ldots, c_k \in P_k(q)\) there exists exactly one state \(q'\) such that \(q' \in \text{post}(c_1) \cap \cdots \cap \text{post}(c_k)\). Next, we define the function \(\delta\) that assigns state \(q' \in \text{post}(c_1) \cap \cdots \cap \text{post}(c_k)\) to every tuple \((q, c_1, \ldots, c_k)\) such that \(q \in Q\) and \(c_a \in P_a(q)\) for \(a = 1, \ldots, k\). Given such a description of a system it is easy to build the corresponding concurrent game structure \(S = (k, Q, \iota, \Pi, \pi, d, \delta)\) by taking \(P_a(q) = d_a(q)\) and numbering actions belonging to the set \(P_a(q)\) for every state \(q\) and agent \(a\). However, the same does not go automatically the other way round, i.e., not for every CGS such a description exists, although an equivalent CGS can be always constructed for which it does [6].

Next, we assume \(Q \subseteq \{0, 1\}^m\), where \(m = \lceil \log_2(|Q|) \rceil\). Let \(PV\) be a set of new propositional variables such that \(PV \cap \Pi = \emptyset\). Then, each state \(q \in Q\) is represented by a global state variable \(w = (w[1], \ldots, w[m])\), where \(w[i] \in PV\) for each \(i = 1, \ldots, m\).

Let \(F_{PV}\) be a set of propositional formulas over \(PV\), and let \(\text{lit}: \{0, 1\} \times PV \rightarrow F_{PV}\) be a function defined as follows: \(\text{lit}(0, p) = \neg p\) and \(\text{lit}(1, p) = p\). Furthermore, let \(w\) be a global state variable. We define the following propositional formulas:

- \((\text{I}_q(w)) := \bigwedge_{i=1}^m \text{lit}(q[i], w[i])\)

this formula encodes the state \(q = (q[1], \ldots, q[m])\) of the model, i.e., \(q[i] = 1\) is encoded by \(w[i]\), and \(q[i] = 0\) is encoded by \(\neg w[i]\).

- \((\text{pre}_c(w))\) and \((\text{post}_c(w))\) for every \(c \in \text{Act}_1 \cup \cdots \cup \text{Act}_k\),

\[\text{pre}_c(w) = \text{a formula which is true for valuation } q = (q[1], \ldots, q[m]) \text{ of } w = (w[1], \ldots, w[m]) \text{ iff } q \in \text{pre}(c) \text{ and } \text{post}_c(w) = \text{a formula which is true for valuation } q \text{ iff } q \in \text{post}(c)\.]

Next, we translate ATL formulas into propositional 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 \(q \in (\phi)\). In so doing we obtain a formula \((\phi(w))\) such that \(\phi\) is valid in the structure \(S\) iff the conjunction \((\phi(w) \wedge I_s(w))\) is satisfiable. Notice that \((\phi(w) \wedge I_s(w))\) is satisfiable only if \((\phi(w))\) is valid for the valuation implied by the initial state \(s\). Operationally, we work backwards from the most nested subformulas, i.e., to compute \((Oa)(w)\), where \(O\) is a modality, we work under the assumption of already having computed \(a(w)\).

Definition 5 (Translations) Given an ATL formula \(\phi\), the propositional translation \((\phi(w))\) is inductively defined as follows:

- \((p)(w) := \bigvee_{q \in (p)} I_q(w)\), for \(p \in \Pi\).
- \((\neg a)(w) := \neg(a(w))\), \((a \lor b)(w) := [a(w)] \lor [b(w)]\).
- \(a = (a_1, \ldots, a_t) \subseteq \{1, \ldots, k\}\) and \(B = \{b_1, \ldots, b_s\} \subseteq \{1, \ldots, k\}\), \(\{a_1, \ldots, a_t\}\) \(\bigcap \bigvee_{c_{a_1} \in \text{Act}_{a_1}, \ldots, c_{a_t} \in \text{Act}_{a_t}} (\Lambda_{i=1}^t \text{pre}_{c_{a_i}}(w) \wedge \text{forall}(v, \bigwedge_{c_{a_1} \in \text{Act}_{a_1}, \ldots, c_{a_t} \in \text{Act}_{a_t}} (\Lambda_{j=1}^s \text{pre}_{c_j}(w) \wedge \bigwedge_{v=1}^t \text{post}_{c_{a_j}}(w) \Rightarrow [a](v)))\).
- \((\langle A \rangle \circ [\alpha](w)) := gfPA([\alpha](w)\).
- \((\langle A \rangle \text{U} [\beta](w)) := 1fPA([\alpha](w), [\beta](w))\).
The algorithms $gfp$ and $lfp$ are based on the standard procedures computing fixed points and are given below. In order to simplify notation, formula $\forall_{a_i \in Act_{a_1}, \ldots, a_n \in Act_{a_n}} (\Lambda_{i=1}^{s} pre_{c}a_i (w) \wedge \exists v. \Lambda_{c_i \in Act_{c_1}, \ldots, c_n \in Act_{c_n}} (\Lambda_{j=1}^{s} post_{c_j} (v) \land \Lambda_{i=1}^{s} post_{c_i} (v) \Rightarrow \psi (v)))$, for a formula $\psi (w)$ defined over variables $w[1], \ldots, w[m]$, is denoted with $\{[A]\} \circ \psi (w) (w)$.}

**Theorem 2 (UMC for ATL)** Given an ATL formula $\varphi$ and a concurrent game structure $S$, the following condition holds: $S \models \varphi$ iff $[\varphi (w)](w) \land I_{s}(w)$ is satisfiable.

**Proof:** First let us observe that a valuation $s$ of $w$ satisfies the formula $I_{s} (w)$ iff $s = q$. So, $s$ is the only valuation that satisfies the formula $I_{s} (w)$.

Next, we need to check that the translations given in Definition 5 are well defined. In order to do it, we have to prove that for every ATL formula $\varphi$ and every state $q$ of a structure $S$ holds: $S, q \models \varphi$ if and only if the valuation $q$ of $w$ satisfies $[\varphi (w)](w)$. The proof follows by induction on the complexity of $\varphi$ and directly stems from the construction of $[\varphi (w)](w)$.

The theorem follows directly for the propositional variables. Assume that the hypothesis holds for all the proper sub-formulas of $\varphi$. If $\varphi$ is equal to either $\neg \alpha$, $\alpha \land \beta$, or $\alpha \lor \beta$, then it is easy to check that the theorem holds.

Consider $\varphi$ to be of the following form:

$\varphi = \{A\} \circ \alpha$. Let $A = \{a_1, \ldots, a_k\} \subseteq \{1, \ldots, k\}$ and $B = \{b_1, \ldots, b_k\} = \{1, \ldots, k\} \setminus \{a_1, \ldots, a_k\}$. Then for every $q = \{A\} \circ \alpha$ (by Lemma 1) $q \models [A](\{\alpha\})$ iff for every agent $a \in A$ there exists a natural number $j_a \leq d_{a}(q)$ such that for every state $q' \in Q$, every agent $b \in B$ and every natural number $j_b \leq d_{b}(q')$ if $q' = \delta(q, j_1, \ldots, j_k)$ then $q'' \models \alpha$ iff for every agent $a \in A$ there exists an action $c_a \in Act_a$ such that $q' \models \pre{c_a}$ and for every state $q' \in Q$, every agent $b \in B$ and every action $c_b \in Act_b$ such that $q' \models \pre{c_b}$ then $q'' \models \alpha$ (by definition of formulas $\pre{c_a}$, $\pre{c_b}$ and inductive assumption) for every $a \in A$ there exists an action $c_a \in Act_a$ such that valuation $q$ of $w$ satisfies formula $\pre{c_a}$ and for every valuation $q'$ of $v$, every agent $b \in B$ and every action $c_b \in Act_b$ if $q'$ satisfies formula $\pre{c_b}$ and $q''$ satisfies conjunction $\Lambda_{i=1}^{s} post_{c_i}(v)$ then $q''$ satisfies formula $[\alpha] (v)$ iff there exist actions $c_{a_1} \in Act_{a_1}, \ldots, c_{a_n} \in Act_{a_n}$ such that valuation $q$ of $w$ satisfies formula $\Lambda_{i=1}^{s} pre_{c_i}(w)$ and for every valuation $q'$ of $v$ and all actions $c_{b_1} \in Act_{b_1}, \ldots, c_{b_n} \in Act_{b_n}$ if $q'$ satisfies formula $\Lambda_{j=1}^{s} pre_{c_j}(w)$ and $q''$ satisfies conjunction $\Lambda_{j=1}^{s} post_{c_j}(v)$ then $q''$ satisfies formula $[\alpha] (v)$ iff valuation $q$ of $w$ satisfies QBF formula $\forall_{c_1 \in Act_{c_1}, \ldots, c_n \in Act_{c_n}} (\Lambda_{i=1}^{s} pre_{c_i}(w) \wedge \forall v. (\Lambda_{c_1 \in Act_{c_1}, \ldots, c_n \in Act_{c_n}} (\Lambda_{i=1}^{s} post_{c_i}(v) \land \Lambda_{i=1}^{s} post_{c_i}(v) \Rightarrow [\alpha] (v))))$ (by Theorem 1) valuation $q$ of $w$ satisfies propositional formula $\forall_{c_1 \in Act_{c_1}, \ldots, c_n \in Act_{c_n}} (\Lambda_{i=1}^{s} pre_{c_i}(w) \wedge \forall v. (\Lambda_{c_1 \in Act_{c_1}, \ldots, c_n \in Act_{c_n}} (\Lambda_{j=1}^{s} post_{c_j}(v) \land \Lambda_{j=1}^{s} post_{c_j}(v) \Rightarrow [\alpha] (v))))$ (by Definition 5) the valuation $q$ of $w$ satisfies $\{[\varphi]\} \circ \alpha (w)$. 

$\varphi = \{A\} \circ \alpha \in \{1, \ldots, k\}$, the proof is based on the fixed-point characterizations of the formulas and correctness of the procedures computing fixed points.

Thus, $S \models \varphi$ iff $S, \rho \models \varphi$ iff the valuation $\rho$ of $w$ satisfies the formula $[\varphi (w)](w) \land I_{s}(w)$ is satisfiable.
7. Example

Consider a system with three processes $a$, $b$, and $c$. The processes assign values to the Boolean variables respectively $x$, $y$, $z$. When $x = 0$, then $a$ can leave the value of $x$ unchanged or change it to 1. When $x = 1$, then $a$ leaves the value of $y$ unchanged. When $y = 0$, then $b$ can leave the value of $y$ unchanged or change $y$ from 0 to 1 either when $x$ is already 1 or when simultaneously $x$ is set to 1. When $y = 1$, then $b$ leaves the value of $y$ unchanged. In a similar way, process $c$ can leave the value of $z$ unchanged if $z = 0$ or change $z$ from 0 to 1 when $y$ is already 1. When $z = 1$, then $c$ leaves the value of $z$ unchanged. We model the synchronous composition of the three processes by the following concurrent game structure $S = \langle k, Q, t, \Pi, \pi, d, \delta \rangle$ where:

- $k = 3$, agent 1 represents process $a$, agent 2 represents process $b$, and agent 3 represents process $c$,
- $Q = \{ q, q_x, q_y, q_z, q_{xy}, q_{xz}, q_{yz}, q_{xyz} \}$, the state $q$ corresponds to $x = y = z = 0$, the state $q_x$ corresponds to $x = 1$ and $y = z = 0$, remaining states have similar interpretations,
- $t = q$,
- $\Pi = \{ x, y, z \}$,
- $\pi(q) = \emptyset$, $\pi(q_x) = \{ x \}$, $\pi(q_y) = \{ y \}$, $\pi(q_z) = \{ z \}$, $\pi(q_{xy}) = \{ x, y \}$, $\pi(q_{xz}) = \{ x, z \}$, $\pi(q_{yz}) = \{ y, z \}$, $\pi(q_{xyz}) = \{ x, y, z \}$.

The agents 1, 2, 3 have the following possible moves:

- $Act_1 = \{ a_1, a_2 \}$, $Act_2 = \{ b_1, b_2, b_3 \}$, $Act_3 = \{ c_1, c_2 \}$ such that $P_1(a_1) = \{ q, q_y, q_z, q_{yz} \}$, $P_1(a_2) = Q$, $P_2(b_1) = \{ q, q_x, q_{xz}, q_{xyz} \}$, $P_2(b_2) = \{ q_x, q_z, q_{xz}, q_{xyz} \}$, $P_3(c_1) = \{ q_x, q_y, q_{xyz} \}$, $P_3(c_2) = \{ q_x, q_y, q_{xyz} \}$.

Pre and post conditions of the actions are given in Table 1. They should be read as follows, e.g., action $a_1$ takes from any state satisfying $x = 0$ to any state satisfying $x = 0$. The resulting functions $d$ and $\delta$ are shown in Figure 1.

We now encode the states in binary form in order to use them in model checking. We need only 3 bits to encode the states. In particular we take the first bit to be equal 1 for $x = 1$ and 0 for $x = 0$, the second bit to be equal 1 for $y = 1$ and 0 for $y = 0$, and similarly the third bit describes the value of variable $z$. Thus $(0, 0, 0) = q_0, (0, 1, 0) = q_y, (0, 0, 1) = q_z, (1, 0, 0) = q_{xy}, (1, 0, 1) = q_{xz}, (0, 1, 1) = q_{xyz}.$

Let $w = (w[1], w[2], w[3])$, $v = (v[1], v[2], v[3])$ be two global state variables. We define the following propositional formulas over $w$ and $v$:

- $I_w(w) := \neg w[1] \land \neg w[2] \land \neg w[3]$,

<table>
<thead>
<tr>
<th>action</th>
<th>pre</th>
<th>post</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>$x = 0$</td>
<td>$x = 0$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$\text{true}$</td>
<td>$x = 1$</td>
</tr>
<tr>
<td>$b_1$</td>
<td>$y = 0$</td>
<td>$y = 0$</td>
</tr>
<tr>
<td>$b_2$</td>
<td>$x = 0 \land y = 0$</td>
<td>$x = 0 \land y = 0$</td>
</tr>
<tr>
<td>$b_3$</td>
<td>$x = 1 \lor y = 1$</td>
<td>$y = 1$</td>
</tr>
<tr>
<td>$c_1$</td>
<td>$z = 0$</td>
<td>$z = 0$</td>
</tr>
<tr>
<td>$c_2$</td>
<td>$y = 1 \lor z = 1$</td>
<td>$z = 1$</td>
</tr>
</tbody>
</table>

Table 1. Pre and post conditions of actions.

![Figure 1. CGS for the system of the example](image)

Consider the following ATL formula $\alpha = \langle\{1, 2\}\rangle \rhd \neg z$ which expresses that agents $a$ and $b$ can cooperate to ensure that in the next state the value of $z$ will be 0. We shall prove that this formula is valid in the structure. First we translate the formula $\neg z$:

- $[\neg z](w) := \neg w[3]$.

Then we translate $\alpha$:

- $[\langle\{1, 2\}\rangle \rhd \neg z](w) := \bigvee_{a \in Act_1, b \in Act_2} (\pre_a(w) \land \pre_b(w) \land \forall v. \bigwedge_{c \in Act_3} (\pre_c(w) \land \post_c(v) \land \pre_b(v) \land \post_b(v) \Rightarrow \neg w[3]))) := \neg w[2] \land \neg w[3]$.

Therefore, $I_w(w) \land [\alpha](w) = (\neg w[1] \land \neg w[2] \land \neg w[3]) \land (\neg w[2] \land \neg w[3]) = \neg w[1] \land \neg w[2] \land \neg w[3]$.
8. Conclusions

Alternating-time Temporal Logic is a formalism that has been proposed to specify and verify multi-agent systems. We have shown that the symbolic technique UMC can be also applied to ATL. This required to encode the ATL operators in a propositional way, which, especially for the next step ATL operator, was quite a tricky task.

Further research in this line will pursue an implementation of the method and valuation of the experimental results that can be obtained as well as comparison this results to results obtained by standard techniques using OBDD’s. On the other hand, we plan to extend this method to the case of Alternating-time Temporal Epistemic Logic (ATEL) [20, 7], formulas of which can be also verified using MOCHA [22, 23] as well as to other logics that enrich ATL with knowledge operators [8]. An extension of the technique of UMC from a purely temporal setting to a temporal-epistemic one we showed in [9]. Since experimental results for CTLK [11] are very promising, we expect to receive a similar efficiency with the current approach.

References