ATL*AS: An Automata-Theoretic Approach and Tool for the Verification of Strategic Abilities in Multi-Agent Systems

Sofia Garcia de Blas Garcia-Alcaide
Imperial College
London, United Kingdom

Francesco Belardinelli
Imperial College
London, United Kingdom
francesco.belardinelli@imperial.ac.uk

ABSTRACT

We present two novel symbolic algorithms for model checking the Alternating-time Temporal Logic ATL*, over both the infinite-trace and the finite-trace semantics. In particular, for infinite traces we design a novel symbolic reduction to parity games. We implement both methods in the ATL*AS model checker and evaluate it using synthetic benchmarks as well as a cybersecurity scenario. Our results demonstrate that the symbolic approach significantly outperforms the explicit-state representation and we find that our parity-game-based algorithm offers a more scalable and efficient solution for infinite-trace verification, outperforming previously available tools. Our results also confirm that finite-trace model checking yields substantial performance benefits over infinite-trace verification. As such, we provide a comprehensive toolset for verifying multi-agent systems against specifications in ATL*.

KEYWORDS

Formal Verification; Model Checking; Alternating-Time Temporal Logic; Automata Theory

ACM Reference Format:
Sofia Garcia de Blas Garcia-Alcaide and Francesco Belardinelli. 2026. ATL*AS: An Automata-Theoretic Approach and Tool for the Verification of Strategic Abilities in Multi-Agent Systems. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 9 pages. https://doi.org/10.65109/LHUR4872

1 INTRODUCTION

Formal verification of multi-agent systems (MAS) is essential for ensuring correctness in strategic, high-stakes settings [14]. Alternating-time Temporal Logic was originally proposed as a specification language to express strategic abilities of agents in MAS [1]. Since then, it has been extended in multiple directions to account for specific features of MAS, including imperfect information and knowledge [19], stochastic transitions [10], quantification on strategies [9, 25], among others.

A key distinction in this domain is about modelling system executions as either infinite or finite traces. The tradition of reactive systems has usually considered infinite traces [27], suitable to model the system's readiness to accept inputs from the user. Nonetheless, many scenarios – including games and learning algorithms – naturally lend themselves to finite-trace modelling, where executions terminate after a finite number of steps. This has led to a vast literature translating results available for the infinite-trace setting to the finite-trace case [3, 12, 13].

Besides modelling purposes, finite traces also offer practical advantages regarding the corresponding verification procedures. Specifically, model checking over finite traces might avoid the complexities of ω-automata and instead use finite automata, yielding simpler algorithms [3].

Contributions. We advance the state of the art in the formal verification of multi-agent systems by developing, implementing and evaluating effcient, symbolic model checking algorithms for both finite- and infinite-trace variants of ATL*. The core contributions of this work are:

(1) *A symbolic algorithm for ATL^*_f model checking and its implementation*, built on the theoretical foundations and complexity result presented in [3]. Our symbolic algorithm includes several design choices that improve substantially over [3] and enable scalable implementation. This is accompanied by the full implementation of both the explicit and symbolic model checking algorithms, thus achieving the first tool for verifying ATL^*_f.
(2) *A symbolic reduction of ATL* model checking to parity games*, resulting in the first available implementation for full ATL* over infinite traces.
(3) *Evaluation on a strategic cybersecurity scenario:* In Sec. 6.4, a concrete attacker-defenders MAS is modelled to demonstrate the applicability of the symbolic ATL^*_f model checker to realistic verification tasks.
(4) *Comparative analysis of finite vs. infinite-trace verification:* We investigate empirically the practical performance trade-offs between finite- and infinite-trace ATL* model checking, as well as between different algorithms in both domains. The results of these evaluations are presented in Sec. 6.

These algorithms are integrated into ATL*AS, a C++ model checking tool that supports both finite and infinite-trace verification of ATL*. Users can provide a system specification and select among the provided algorithms to verify temporal and strategic properties efficiently. The code of ATL*AS is provided in the supplementary material to this paper, which can be found at ..., which can be found at https://arxiv.org/abs/2510.17306.

Related Work. The computational complexity of model checking temporal logics varies significantly depending on the language considered and its expressive power. Table 1 summarises the complexity results for model checking some key multi-agent logics,

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus. © 2026 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). https://doi.org/10.65109/LHUR4872

under the assumption of perfect recall strategies. Existing work on ATL^*_f model checking is centered around the explicit-state algorithm introduced in [3], which provides the first decidability and complexity results for model checking ATL^*_f. However, this approach suffers from limited scalability due to full state-space enumeration. ATL^* over infinite traces lacks dedicated tools, but can be indirectly verified via the SL[1G] fragment of strategy logic, supported by the symbolic MCMAS-SL[1G] checker [7], which uses generalised parity games. Parity-game-based model checking has also been developed for logics such as the modal μ-calculus [15] and game logic [21]. The reduction of ATL^* model checking to parity games was established in [11], showing that control problems over Kripke structures with ω-regular objectives can be encoded as parity games. However, no concrete algorithm for concurrent game structures was provided, nor were the symbolic aspects required for efficient practical implementation addressed.

Table 1: Model Checking Complexity of Relevant Logics

Logic	Complexity	Implementation
ATL	PTIME-complete [1]	MCMAS [22]
ATL^*	2EXPTIME-complete [1]	**ATL^*AS (here)**
ATL^*_f	2EXPTIME-complete [3]	**ATL^*AS (here)**
SL	Non-elementary [24]	MCMAS-SLK [6]
SL[1G]	2EXPTIME-complete [24]	MCMAS-SL[1G] [7]

2 BACKGROUND

In this section we introduce the background on Alternating-time Temporal Logic, including its syntax and semantics (Sec. 2.1) and the basics on automata theory to solve the corresponding model checking problem (Sec. 2.2). In what follows, give a (possibly infinite) sequence $\tau = t_0, \dots, t_n, \dots$ of elements from set T, we denote its length as $|\tau| \in \mathbb{N} \cup \{\infty\}$, the i-th element as $\tau_i = t_i$, and the subsequence $t_i, \dots, t_{n+i}, \dots$ starting at position i as $\tau_{\geq i}$.

2.1 Alternating-time Temporal Logic

Syntax. We state the syntax for the language ATL^* as introduced in [1], which is also the syntax for its variant ATL^*_f interpreted over finite traces. Let AP be the set of *atomic propositions* (or atoms) and Ag the set of *agents*. Formulas in ATL^* are the state formulas φ built using path formulas ψ with Boolean operators and strategic quantifiers:

$$\varphi \quad ::= \quad p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle\langle A \rangle\rangle\psi$$
$$\psi \quad ::= \quad \varphi \mid \neg\psi \mid \psi \wedge \psi \mid \mathsf{X}\psi \mid \psi \mathsf{U} \psi$$

where $p \in AP$ and $A \subseteq Ag$.

Operator *next* X and *until* U are the standard modalities of Linear-time Temporal Logic (LTL) [26]. Additional temporal operators are also derivable from X and U, including finally $\mathsf{F}\varphi \equiv \mathtt{true}\mathsf{U}\varphi$ and globally $\mathsf{G}\varphi \equiv \neg\mathsf{F}\neg\varphi$. For simplicity, we include these derived operators in the syntax.

Semantics. We present the semantics of ATL^* by means of Concurrent Game Structures.

Definition 2.1 (CGS). A *Concurrent Game Structure* is a tuple $G = \langle Ag, AP, S, s_0, Act_{a \in Ag}, \delta, \lambda \rangle$ where:

- Ag is the finite, non-empty set of *agents*;
- AP is the finite, non-empty set of *atoms*;
- S is the finite, non-empty set of *states*, with *initial state* $s_0 \in S$;
- for each agent $a \in Ag$, Act_a is a finite, non-empty set of *actions*;
- $\delta : S \times Jact \rightarrow S$ is the *transition function*, where $Jact = \prod_{a \in Ag} Act_a$ is the set of joint actions;
- $\lambda : S \rightarrow 2^{AP}$ is the *labelling function* that assigns to each state s the set $\lambda(s)$ of atoms that are true in s.

To interpret ATL^* on finite traces, we extend Def. 2.1 with a set $F \subseteq S$ of final states to identify terminating executions, as originally proposed in [3].

Further, we define the following concepts related to CGSs. A *path* is a sequence $\pi = s_0, s_1, \dots, s_n, \dots$ of states, either infinite, or finite terminating in a final state in F, where for every $i \leq |\pi|$, there exists a joint action $J \in Jact$ such that $\delta(\pi_i, J) = \pi_{i+1}$. In the finite-trace setting, we additionally define a *history* as a finite path not necessarily ending in a final state. The sets of all histories and paths are denoted as $Hist$ and $Path$ respectively.

A *strategy* for an agent $a \in Ag$ is a function $\sigma_a : Hist \rightarrow Act_a$, mapping histories to actions. Notice that here we consider *perfect recall* strategies, as action selection depends on the entire history, not just on the current state. The set of all (perfect recall) strategies is denoted as $\sum_R(G)$. A *joint strategy* $\sigma_A : A \rightarrow \sum_R(G)$ is a function associating a strategy to each agent in coalition $A \subseteq Ag$.

Given a state $s \in S$, the *finite* (resp. *infinite*) outcome of a joint strategy σ_A from s, denoted $\mathrm{out}_f(s, \sigma_A)$ (resp. $\mathrm{out}_\infty(s, \sigma_A)$), is the set of finite (resp. infinite) paths π consistent with s and σ_A, that is, (i) $\pi_0 = s$; and (ii) for every $i < |\pi|$, there exists a joint action $J \in Jact$ such that $\pi_{i+1} = \delta(\pi_i, J)$ and $J(a) = \sigma_A(a)(\pi_{\leq i})$ for each agent $a \in A$. We now define a parametrised semantics for $x \in \{\infty, f\}$ to denote the infinite and finite trace interpretation, respectively.

$(G, s) \models_x p$	iff $p \in \lambda(s)$
$(G, s) \models_x \neg\varphi$	iff $(G, s) \not\models_x \varphi$
$(G, s) \models_x \varphi \wedge \varphi'$	iff $(G, s) \models_x \varphi$ and $(G, s) \models_x \varphi'$
$(G, s) \models_x \langle\langle A \rangle\rangle\psi$	iff $\exists\sigma_A \in \Sigma_R(G)$ s.t. $\forall\pi \in \mathrm{out}_x(s, \sigma_A)$, $(G, \pi) \models_x \psi$
$(G, \pi) \models_x \varphi$	iff $(G, \pi_0) \models_x \varphi$
$(G, \pi) \models_x \neg\psi$	iff $(G, \pi) \not\models_x \psi$
$(G, \pi) \models_x \psi_1 \wedge \psi_2$	iff $(G, \pi) \models_x \psi_1$ and $(G, \pi) \models_x \psi_2$
$(G, \pi) \models_x \mathsf{X}\psi$	iff $|\pi| > 1$ and $(G, \pi_{\geq 1}) \models_x \psi$
$(G, \pi) \models_x \psi_1 \mathsf{U} \psi_2$	iff $\exists j < |\pi|$ s.t. $(G, \pi_{\geq j}) \models_x \psi_2$ and for all $0 \leq k < j$, $(G, \pi_{\geq k}) \models_x \psi_1$

Notice that on infinite traces, the condition for $\mathsf{X}\psi$ whereby $|\pi| > 1$ is always satisfied, as $|\pi| = \infty$.

We write $G \models_x \varphi$ as shorthand for $(G, s_0) \models_x \varphi$, where s_0 is the initial state of G.

Model Checking. In the rest of this paper, we are interested in the following model checking problem.

Definition 2.2 (Model Checking Problem). Given a CGS G and an ATL^* formula ϕ, determine whether $G \models \phi$.

Complexity results for model checking ATL*, relevant fragments and extensions are reported in Table 1, for the case of perfect recall strategies.

2.2 Automata

Our approach relies on automata-theoretic techniques for model checking temporal logics. Our algorithms use deterministic finite automata (DFAs) to represent LTL$_f$ formulas in the finite trace interpretation, and deterministic parity automata (DPAs) to represent LTL formulas when interpreted over infinite traces. We define both these automata hereafter.

Definition 2.3 (DFA). A *deterministic finite automaton* is a tuple $\mathcal{A} = \langle Q, q_0, \Sigma, \delta, F\rangle$ where (i) Q is a finite set of *states*, with *initial state* q_0; (ii) Σ is the *alphabet*; (iii) $\delta : Q \times \Sigma \to Q$ is the *transition function*; (iv) $F \subseteq Q$ is a set of *final states*.

A *run* q on a DFA for a finite word $w \in \Sigma^*$ is a finite sequence $q_0 q_1, \dots, q_n$ of states such that for $i < |w|$, $q_{i+1} = \delta(q_i, w_i)$. A run is *accepting* if it ends in a final state, i.e., $q_n \in F$.

Definition 2.4 (DPA). A *deterministic parity automaton* is a tuple $A = \langle Q, q_0, \Sigma, \delta, F\rangle$, where Q, q_0, Σ, δ are defined as for DFAs, and F is now a parity acceptance condition $\{P_1, P_2, \dots, P_n\}$, where each $P_i \subseteq Q$. Moreover, an infinite run $\sigma = q_0 q_1 \dots$ is *accepting* if $\min_i (P_i \cap \mathrm{Inf}(\sigma) \neq \emptyset)$ is odd, where $\mathrm{Inf}(\sigma)$ is the set of states occurring infinitely often in σ.

Given an automaton $\mathcal{A}$, the *language* $\mathcal{L}(\mathcal{A})$ is the set of words that are accepted by some run in $\mathcal{A}$. Deterministic parity automata induce a natural class of two-player games, known as parity games, where acceptance conditions are rephrased in terms of winning strategies over infinite plays.

Definition 2.5 (Parity Game). A (min-)*parity game* is a tuple $\mathcal{G} = (V, V_0, V_1, E, \Omega)$, where (V, E) is a finite directed graph, $V = V_0 \cup V_1$ partitions positions between player 0 (Even) and player 1 (Odd), and $\Omega\colon V \to \mathbb{N}$ assigns each vertex a *priority*. A play is an infinite path $v_0 v_1 v_2 \cdots$; player 0 wins if the minimum priority occurring infinitely often is even, otherwise player 1 wins.

3 SYMBOLIC ALGORITHM FOR FINITE TRACES

The model checking procedure for ATL$^*_f$ follows a recursive labelling approach, where subformulas are evaluated bottom-up and annotated over the state space, as described in [3]. The key computational step occurs when evaluating a strategy formula of the form $\langle\langle A\rangle\rangle\psi$. Here, the path formula ψ is first rewritten into an equivalent LTL$_f$ formula by replacing all maximal state subformulas with fresh atoms. The core task then reduces to computing the set of states from which coalition A can enforce ψ. This is solved by invoking the `GameSolving` procedure, whose high-level structure is shown in Algorithm 1.

The algorithm first translates the LTL$_f$ formula into an equivalent DFA (line 1) [12]. The product between the CGS G and the DFA is then constructed as per Def. 3.1 (line 2), yielding a structure that simultaneously tracks the evolution of the MAS and the progress towards satisfying ψ. To initialise the fixpoint computation, an initial safety set is derived in line 3, it includes all product states that are either not final in the CGS or are accepting in the DFA. Intuitively, this set excludes precisely those states in which the system has terminated and the property is not yet satisfied. Finally, the greatest fixpoint iteration (line 4) computes the full set of CGS states from which the coalition can enfoce ψ through some joint strategy. We present novel symbolic algorithms for the two central steps in this procedure: product construction and fixpoint solution.

Algorithm 1 `GameSolving`

Input: CGS G, Strategic formula $\langle\langle A\rangle\rangle\psi$
Output: $\{q \in S \mid (G, q) \models \langle\langle A\rangle\rangle\psi\}$

1: Convert LTL$_f$ formula ψ into DFA $\mathcal{A}_\psi$
2: Construct product graph of G and $\mathcal{A}_\psi$
3: Derive safety set $Safe = \{(q, s) \in Q \times S \mid q \notin F_G \vee s \in F_{A_\psi}\}$
4: Solve safety game through greatest fixpoint equation $Y \mapsto Safe \cap Pre(Y)$.
5: **return** solution of the fixpoint equation

Symbolic Product Construction. The CGS and the formula DFA are encoded as BDDs [5]. Each structural component - states, actions, transition relation, and labelling function - is encoded as a BDD to enable scalable manipulation and fixpoint computation. These encodings are used in Algorithms 2 and 3 to symbolically construct the product graph and solve the safety game respectively following Def. 3.1 and 3.2.

Definition 3.1 (Product of CGS and DFA). Let $G = \langle Ag, AP, Q, q_0, Act_{a\in Ag}, \delta, F, \lambda\rangle$ be a CGS and $\mathcal{A}_\psi = \langle S, s_0, 2^{AP}, \Delta, F'\rangle$ a DFA encoding the temporal formula ψ. The product graph $G \otimes \mathcal{A}_\psi$ for a coalition A of agents is a structure whose set of states is $Q \times S$, with initial state (q_0, s'_0) for $s'_0 = \Delta(s_0, \lambda(q_0))$. Transitions $\to$ in the product are defined as

$$((q, s), \alpha_A) \to (q', s') \iff \exists \alpha_{-A}.(q, \alpha_A \cup \alpha_{-A}, q') \in \delta \wedge (s, \lambda(q'), s') \in \Delta$$

where α_{-A} are the joint actions from the non-coalition agents.

Algorithm 2 Symbolic Product Construction

Input: CGS G, DFA A_ψ, Coalition $A \subseteq Ag$
Output: Initial Safety set $Safe \subseteq Q \times S$

$\delta_A(\vec{q}, \vec{a}_A, \vec{q}') := \exists \vec{a}_{-A}.\ \delta(\vec{q}, \vec{a}_A, \vec{a}_{-A}, \vec{q}')$
$\delta'(\vec{q}, \vec{a}_A, \vec{q}', \vec{s}, \vec{s}') := \delta_A(\vec{q}, \vec{a}_A, \vec{q}') \wedge \Delta(\vec{s}, \vec{q}', \vec{s}')$
$q'_0 := \mathrm{SwapVariables}(\exists \vec{s}.(q_0 \wedge \mathrm{SwapVariables}(\Delta(\vec{s}, \vec{q}', \vec{s}'), \vec{q}', \vec{q})), \vec{s}', \vec{s})$
$R_0 := q'_0$; $R := \emptyset$
repeat
  $R \leftarrow R \cup R_0$
  $R_0 \leftarrow \mathrm{Post}(R_0) \setminus R$
until $R_0 = \emptyset$
$Safe := R \wedge (\neg F \vee F')$
return $Safe$

Algorithm 2 proceeds in three phases: (i) abstraction, (ii) composition, and (iii) pruning. The transition relation of the CGS is first abstracted w.r.t. coalition A, by existentially quantifying over non-coalition actions in line 1. This yields a BDD encoding all transitions that A can enforce. Next, the product transition relation is constructed by combining the abstracted CGS relation with the DFA's transition relation (line 2), synchronizing on propositional labellings. This is enabled by replacing the formulas over the propositions AP in the DFA labels by the states in the CGS in which those propositional formulas hold. A symbolic reachability fixpoint starting from the initial state in line 3 then computes the set of reachable product states (lines 4-8). Finally, in line 9 the safety set is computed as described in Def. 3.2. This set is the domain of the safety game solved in the next stage.

Definition 3.2 (Safety Set and Safety Game). Let $P \subseteq Q \times S$ be the set of product states. The initial safety set is defined as [3]:

$$Safe = \{(q, s) \in P \mid q \notin F \vee q \in F'\}$$

A safety game is a tuple $(V, \Sigma, E, Safe)$, where V are the vertices, Σ the action alphabet, $E \subseteq V \times \Sigma \times V$ the transition relation, and $Safe \subseteq V$ the safety set [18]. Solving the safety game consists of computing the greatest fixpoint operation $Y \mapsto Safe \cap Pre(Y)$ where $Pre(Y) = \{v \mid \exists \sigma . \forall v' . (v, \sigma, w) \in E \rightarrow w \in Safe\}$. Intuitively, this computes the set of states from which the coalition has a strategy to remain within $Safe$ indefinitely.

Algorithm 3 Symbolic Fixpoint Solution

Input: Product transition relation δ', initial safety set Y
Output: Set of states satisfying $\langle\langle A \rangle\rangle \psi$

$Y_0 \leftarrow Y$
repeat
  $T_{bad}(\vec{q}, \vec{s}, \vec{a}_A, \vec{q}', \vec{s}') \leftarrow \delta' \wedge \neg Y_i$
  $B(\vec{q}, \vec{s}, \vec{a}_A) \leftarrow \exists\, \vec{q}', \vec{s}'.\ T_{\text{bad}}$
  $G(\vec{q}, \vec{s}, \vec{a}_A) \leftarrow \neg B$
  $\text{Pre}(Y_i)(\vec{q}, \vec{s}) \leftarrow \exists\, \vec{a}_A.\ G$
  $Y_{i+1} \leftarrow Y_i \wedge \text{Pre}(Y_i)$
until $Y_{i+1} = Y_i$
return Y_i

Symbolic Fixpoint Solution Algorithm. The symbolic fixpoint algorithm presented in Alg. 3 computes the controllable safety region for coalition A. At each iteration, any states in the current safety set from which unsafe transitions exist are identified and removed via a pre-image operator that quantifies over coalition actions. The fixpoint converges to the maximal set of product states from which A can guarantee the satisfaction of ψ.

Theorem 3.3 (Soundness of Alg. 1). *The symbolic procedure presented in Alg. 1 correctly computes the set of states satisfying $\langle\langle A \rangle\rangle \psi$ interpreted over finite-trace semantics.*

This follows from Theorem 1 in [3], together with the fact that the binary encodings of the CGS and DFA as BDDs preserve their operational semantics (see Sec. **??** in the supplementary material), and the symbolic operations applied in Alg. 2 and 3 are sound implementations of the corresponding explicit-state procedures.

4 PARITY-GAME SYMBOLIC ALGORITHM FOR INFINITE TRACES

In this section we present a symbolic reduction of ATL* model checking on infinite traces to the solution of a parity game, which is a two-player game played on infinite traces on a graph.

The ATL* model checking algorithm again follows a recursive labelling approach, where the interesting cases are strategy formulas $\langle\langle A \rangle\rangle \psi$, where ψ is an LTL path formula. Intuitively, to verify these formulas we translate ψ into a deterministic parity automaton (DPA), encoded symbolically using BDDs. The corresponding parity game captures the strategic interactions between coalition A and the environment, with winning conditions derived from the DPA's acceptance condition. The complete procedure is shown in Alg. 4.

Symbolic Parity Game Construction. Given a CGS and a coalition A, we construct a two-player parity game described as follows:

Definition 4.1 (Parity Game from CGS and DPA). Given a CGS $G = \langle Ag, AP, Q, Act_{a \in Ag}, \delta, q_0, F, \lambda \rangle$ and a deterministic parity automaton (DPA) $A_\psi = \langle S, \Delta, s_0, \omega \rangle$, we define a parity game $\mathcal{G} = (V, E, \Omega)$ as follows:

- The set of vertices is $V = V_0 \cup V_1$, where:
  - $V_0 \subseteq Q \times S$ are positions controlled by player 0 (coalition A),
  - $V_1 \subseteq Q \times S \times JAct_A$ are positions controlled by player 1, i.e., $Ag \setminus A$, where $JAct_A$ is the set of joint actions available to coalition A.
- The edge relation $E = E_{0 \rightarrow 1} \cup E_{1 \rightarrow 0}$ is defined by:

$$E_{0 \rightarrow 1} = \{((q, s), (q, s, a_A)) \mid (q, s) \in V_0,\ a_A \in JAct_A\}$$
$$E_{1 \rightarrow 0} = \{((q, s, a_A), (q', s')) \mid (s, \lambda(q'), s') \in \Delta \wedge \exists a_{-A} \in JAct_{-A}.\ \ (q, a_A \cup a_{-A}, q') \in \delta\}$$

- The priority function $\Omega : V \rightarrow \mathbb{N}$ assigns to each vertex the priority of its automaton state:

$$\Omega((q, s)) = \omega(s), \quad \Omega((q, s, a_A)) = \omega(s)$$

The parity game is constructed over a symbolic state space that includes the current CGS state, the DPA state, and – at player 1's vertices – coalition A's chosen joint action. This split encodes the alternating move structure required for parity games. Transitions from player 0 to player 1 copy the current state and append a coalition action; transitions from player 1 execute the CGS and DPA transitions synchronously.

Solving Parity Games. The winning region for player 0 in the resulting game corresponds to the set of CGS states from which coalition A can enforce the satisfaction of ψ. The game can be solved using symbolic parity game solving algorithms such as the set-based progress measure algorithm from [8] to obtain a fully symbolic ATL* model checking procedure.

Theorem 4.2 (Soundness of Alg. 4). *The symbolic procedure presented in Alg. 4 correctly computes the set of states satisfying $\langle\langle A \rangle\rangle \psi$ interpreted over infinite-trace semantics.*

Theorem 4.2 follows from the correctness of the symbolic encoding and the general result in [11], which shows that control problems with ω-regular objectives (including LTL formulas) can be solved via parity games. By the correct encoding of CGS G and

Algorithm 4 ATL* Model Checking Symbolic Algorithm through Parity Games

Input: Symbolic CGS, ATL* formula $\langle\langle A\rangle\rangle\psi$
Output: Set W of states satisfying the formula.
Transform ψ into symbolic DPA $A_\psi = \langle S, \Delta, s_0, \Omega\rangle$
Compute reachable product states: $Reach(\vec{q}, \vec{s})$
$V_0(\vec{q}, \vec{s}) = Reach(\vec{q}, \vec{s})$
$V_1(\vec{q}, \vec{s}, \vec{a_A}) = V_0 \wedge Act_A(\vec{q}, \vec{a_A})$
$E_{0\to 1} = V_0(\vec{q}, \vec{s}) \wedge Id(\vec{q}, \vec{q'}) \wedge Id(\vec{s}, \vec{s'}) \wedge Act_A(\vec{q'}, \vec{a'_A})$
$E_{1\to 0} = V_1 \wedge \delta(\vec{q}, \vec{a_A}, \vec{a}_{-A}, \vec{q'}) \wedge \Delta(\vec{s}, \vec{q}, \vec{s'})$
Construct game $P = (V_0, V_1, E = E_{0\to 1} \cup E_{1\to 0}, \Omega)$
Solve parity game P
return Winning region W for Player 0

DPA $\mathcal{A}_\psi$ as BDDs (see Sec. **??** in the supplementary material), and constructing the parity game as defined in Def. 4.1, we obtain a sound symbolic reduction. The BDD-based operations faithfully implement the semantics of the parity game, and the computed winning region characterizes the set of states from which coalition A can enforce ψ.

5 IMPLEMENTATION

We implemented the algorithms presented in Sec. 3 and 4 in our verification tool, ATL*AS, which supports both finite- and infinite-trace semantics of ATL*. The repository for this tool is provided as supplementary material. The tool integrates three model checking backends: an explicit-state and a symbolic-state solver for ATL^*_f, and a symbolic parity-game-based solver for ATL*. The overall architecture of the tool is illustrated in Fig. 1, highlighting the main components and their interactions across (i) the frontend, (ii) automata generation pipeline, and (iii) backend solvers.

Input Language. ATL*AS uses a subset of the ISPL language [22], extended to support the declaration of final states. The parser has been extended to support the full syntax of ATL*.

Explicit and Symbolic Backends. For the explicit algorithms, transitions and state sets are implemented using adjacency lists and dynamic bitsets from the Boost Graph Library [28]. Symbolic representations use the CUDD package [29] to manipulate BDDs. Both explicit and symbolic pipelines share a common preprocessing stage that handles the LTL_f-to-DFA conversion described next.

Automata Translation for Finite Traces. Translation of LTL_f path formulas into deterministic finite automata (DFA) is a critical bottleneck in ATL^*_f model checking. To mitigate this, ATL*AS adopts a parallel translation strategy that exploits the complementary performance of existing tools across varied formula structures. Specifically, we run translators Lisa2 [2], Lydia [17], and LTLF2DFA [31] concurrently as subprocesses. The first tool to produce a valid DFA terminates the others, and its output is used in the subsequent verification procedure. This asynchronous execution model significantly reduces automaton generation in practice.

Automata Translation for Infinite Traces. In the infinite-trace setting, LTL path formulas are translated into deterministic parity automata (DPA) using Rabinizer4 [20] which implements the Safraless translation through limit-deterministic Büchi automata (LDBA) [16]. Rabinizer4 outputs a transition-based DPA, where priorities are assigned to transitions. We then translate it into a state-based DPA in order to build the parity game.

Parity Game Solving Backend. ATL*AS integrates the symbolic, set-based algorithm from [8] to solve parity games in the infinite-trace ATL* setting. This algorithm achieves quasi-polynomial complexity in symbolic operations and linear symbolic space, enabling efficient, fully symbolic game solving.

6 EVALUATION AND RESULTS

We evaluate the performance and scalability of ATL*AS using a synthetic benchmark and a real-world cybersecurity scenario. The synthetic benchmark is based on a multi-agent counter system with two parameters: the maximum counter value C and the maximum number of steps S in the finite-trace case. This is inspired by the single-counter LTL_f synthesis benchmark suite [30], but adapted to strategic reasoning. On the synthetic benchmark, two experiments were conducted: one scaling the size of the CGS and another scaling the size of the input ATL* formula. Since there are no specific ATL* model checking tools, we compare the parity-game-based ATL*AS model checker against the MCMAS-SL[1G] tool on the fair scheduler synthesis benchmark. Finally, we assessed the symbolic ATL^*_f algorithm on a cybersecurity scenario to evaluate its practical applicability in a real-world multi-agent setting.

All experiments were run on an Intel Core™ i7-10700 CPU (2.90GHz) with 16GB RAM and SSD storage.

6.1 Evaluation on Finite Traces

Experiment 1: Scaling the CGS.. We varied the benchmark parameters C and S to generate a range of models. The formula to verify is $\langle\langle A\rangle\rangle\,\mathsf{F}\,$`counter_max`, where `counter_max` holds in states where the counter reaches its maximum value C. Figure 2 shows the average runtime over five runs for each instance. The explicit checker exhibits a steep increase in runtime, with performance degrading significantly beyond 100k states. This is likely due to the cost of enumerating the full state-space product and executing repeated fixpoint computations. For the largest instance (nearly 500k states, $C = 800$, $S = 800$), verification required over 27 hours. In contrast, the symbolic checker scales more efficiently. While still showing polynomial growth (approximately quadratic), runtimes remain far lower. Even with over 3M states ($C = 2000$, $S = 2000$), the symbolic checker completes in under 1.5 hours, demonstrating significantly better scalability.

Experiment 2: Scaling the formula size. To assess the impact of temporal formula complexity, we fixed the CGS parameters to $C = 40$, $S = 35$, yielding approximately 1k states, and varied the depth of a nested formula of the form:

$$\langle\langle A, B\rangle\rangle\,\mathsf{F}\,p_1 \wedge \mathsf{X}(\mathsf{F}\,p_2 \wedge \mathsf{X}(\mathsf{F}\,p_3 \wedge \ldots \wedge \mathsf{X}(\mathsf{F}p_n)))$$

where each p_i holds in states where the counter equals i. Formula size n ranges from 1 to 20. Figure 3 reports average runtimes and indicates the tool used for DFA construction.

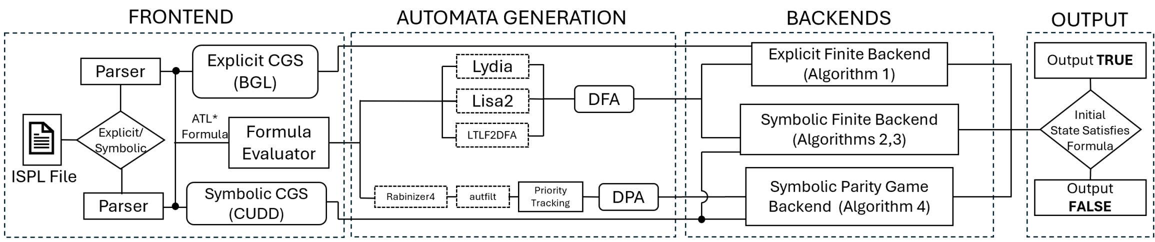

Figure 1: Overview of ATL*AS tool architecture, showing the modular design of the frontend, automata generation pipeline, and solver backends. Dashed rectangles indicate the external components used.

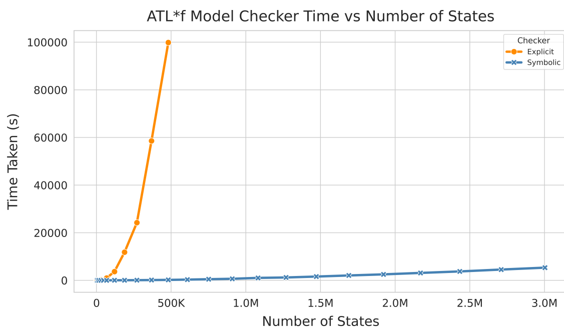

Figure 2: Exp. 1 – Comparison of runtime performance between the explicit and symbolic model checkers.

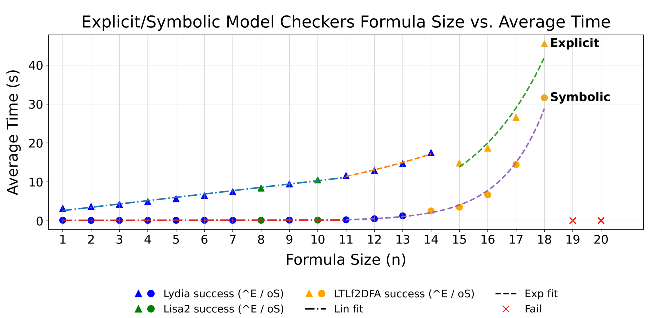

Figure 3: Exp. 2 – Runtime of the explicit and symbolic model checkers as formula size increases. Marker colour indicates the tool used for DFA generation. Dashed lines show fitted linear and exponential curves.

Both the explicit and symbolic implementation scale linearly up to $n = 11$, with the symbolic version consistently faster due to more efficient product construction and fixpoint solving. For $n \geq 12$, exponential growth emerges, dominated by the cost of automaton translation, although the symbolic checker remains more efficient, maintaining a practical runtime even for the largest successful instance of $n = 18$.

For $n \geq 19$, all automata generation tools fail due to resource exhaustion. The parallelised translation strategy, which selects

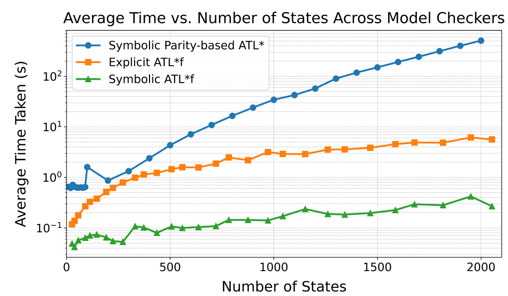

Figure 4: Exp. 1: Comparison of runtime versus model size for finite-trace (explicit and symbolic) and infinite-trace (parity-game–based) model checkers.

Lydia and Lisa2 for small formulas and LTLF2DFA for larger ones, proves critical. Without this, failure would occur at $n = 17$. This hybrid approach improves robustness across the range of formula sizes.

Discussion. The experiments demonstrate our symbolic model checking algorithm's superior scalability and efficiency over the explicit implementation across both dimensions of the problem: system size and formula complexity. The modular use of multiple translation tools further enhances reliability and performance.

6.2 Finite Traces vs. Infinite Traces Comparison

Figures 4 and 5 compare the performance of all model checking algorithms – explicit finite-trace, symbolic finite-trace, and parity-based infinite-trace – on Experiments 1 and 2, respectively.

In both experiments, the finite-trace model checkers, particularly the symbolic variant, significantly outperform the parity-based infinite-trace approach, often by several orders of magnitude. This gap is especially pronounced in the formula-scaling experiment, since in practice, LTL_f-to-DFA conversions often yield automata far smaller than their worst-case bounds, while the corresponding ω-automata for LTL can remain large even when generated with optimized tools like Rabinizer4. These results confirm the hypothesis presented in [3] that when the system being modelled allows a finite-trace representation, ATL_f^* model checking is substantially more tractable. Obviously, modelling system executions as finite traces is not always appropriate/possible.

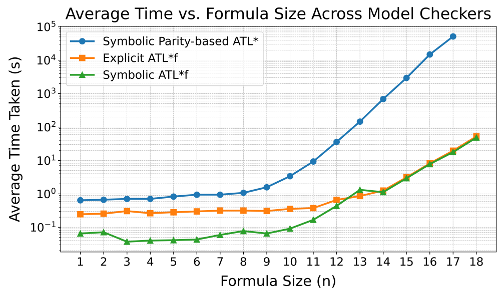

Figure 5: Exp. 2: Comparison of runtime versus formula size for finite-trace (explicit and symbolic) and infinite-trace (parity-game–based) model checkers.

6.3 Comparison with MCMAS-SL[1G]

While no existing tools implement model checking specifically for ATL*, verification can be performed indirectly using logics that strictly subsume it. One such logic is the one-goal fragment of Strategy Logic (SL[1G]), whose syntax and semantics generalise those of ATL*, and whose model checking problem is also 2EXPTIME-complete [24]. This makes SL[1G] a natural reference point for comparison, particularly via the SL[1G] model checking extension of the MCMAS model checker [7].

To compare ATL*AS to MCMAS-SL[1G], we use the fair scheduler synthesis benchmark in [7], parameterised by the number of processes n competing for access to a shared resource. The model has $n + 1$ agents in total, n competing processes and 1 scheduler. The property to be verified ensures fairness for each process: if a process is waiting (atom wt_i), it will eventually cease waiting. This property can be encoded in ATL* as:

$$\bigwedge_{i=1}^{n} \langle\langle P_i \rangle\rangle \, \mathsf{G}(wt_i \rightarrow \mathsf{F}\neg wt_i)$$

Since SL[1G] subsumes ATL* this property can also be expressed in SL[1G]. The runtime comparison is shown in Table 2.

Table 2: Runtime comparison between MCMAS-SL[1G] and ATL*AS as the number of processes increases.

#Processes	#States	ATL*AS Runtime (s)	MCMAS-SL[1G] Runtime (s)
2	9	0.94	0.15
3	21	1.63	3.96
4	49	3.17	234.25
5	113	6.85	4339.47
6	257	21.40	60851.20

ATL*AS outperforms MCMAS-SL[1G] consistently from $n = 3$ onwards, with the performance gap widening exponentially. For $n = 2$, MCMAS-SL[1G] is marginally faster, likely due to lower fixed overheads for small models. However, as model size increases, ATL*AS achieves runtimes up to four orders of magnitude faster. Two factors contribute to this advantage. First, ATL*AS leverages modern external tools for automata generation. Specifically, the DPA is produced using a Safra-less translation based on LDBA [16], which yields smaller automata than the Generalised Büchi Automata (GBA)-based approach used in MCMAS-SL[1G]. Second, for game solving, ATL*AS employs symbolic quasi-polynomial parity game algorithms, while MCMAS-SL[1G] relies on a symbolic implementation of Zielonka's algorithm [33]. These combined improvements in algorithmic and architectural design, along with a simpler logic, enable ATL*AS to achieve significantly better scalability in practice.

6.4 Cybersecurity Modelling Scenario

To demonstrate the applicability of ATL^*_f verification in real-world contexts, we model a multi-agent cybersecurity scenario inspired by the CyMARL simulation environment [32].

Scenario Description. The system consists of five critical servers, protected by two defenders and targeted by a single intelligent attacker. Each server represents a critical service within an enterprise infrastructure. Defenders act as automated intrusion prevention systems (IPSs) constrained by a shared budget, while the attacker models an advanced persistent threat (APT) executing targeted campaigns to compromise one of three objectives:

- **Confidentiality Breach**: gain administrator-level access to exfiltrate sensitive data.
- **Integrity Violation**: insert or modify files to corrupt system behaviour.
- **Availability Attack**: deploy denial-of-service malware to render a host unusable.

Each defender action incurs a cost that reflects its real-world impact on system performance. To preserve decidability while capturing the defenders' partial observability, action availability is restricted dynamically via heuristic protocols that determine what responses are permissible based on observed evidence.

The model is finite-horizon, reflecting fixed-duration attacks, and is formalised as a Concurrent Game Structure (CGS) with final states suitable for ATL^*_f verification.

State Space and Observables. Each global state includes:

- **Attacker position** $p \in \{0, \ldots, 4\}$: the server on which the attacker currently resides.
- **Privilege levels** $\text{priv}[i] \in \{0, 1, 2\}$ for each server i: 0 = no access, 1 = user, 2 = admin.
- **Vulnerability map** $\text{scanned}[i] \in \{0, 1\}$: whether server i has been scanned.
- **Availability map** $\text{down}[i] \in \{0, 1\}$: whether server i is unavailable due to a DoS payload.
- **Defender budgets** B: remaining combined budget for defenders D_1 and D_2.
- **Suspicion buckets** $\sigma_{j,i} \in \{0, 1, 2\}$: current suspicion level that defender D_j assigns to server i.
- **Step counter** t: the current time step, $0 \leq t \leq T$.
- **Server flags** $F_i = (F_{i,1}, \ldots, F_{i,6}) \in \{0, 1\}^6$: six Boolean indicators per server i.

When the defender performs a $\texttt{monitor}(i)$ action on a server they observe the set of flags describing what events have ocurred on that server. We refer to [32] for full details on this scenario.

Agent Actions. At each timestep, the attacker and both defenders each choose a single action. Logical preconditions (e.g., scanning before exploit) and access constraints are enforced by the protocol function.

Defender actions.

- `monitor`(i): query server i to observe its flag vector F_i.
- `remove`(i): evict any user-level malicious process on i.
- `restore`(i): re-image i, clearing all sessions and processes.
- `do_nothing`: skip action.

In the *integrity* scenario, defenders also have an `analyze`(i) action to inspect file-system metadata to detect tampering and a `data_repair`(i) action to remove or revert modified files on i.

Attacker actions.

- `scan`(i): probe i for vulnerabilities.
- `exploit`(i): leverage a known vulnerability to obtain user-level access.
- `escalate`(i): promote from user to admin privileges on i.
- `do_nothing`: skip action.

Additionally in the *integrity* scenario: `tamper`(i), modifying or corrupting files (requires appropriate privileges) and in the *availability* scenario: `deny`(i), install a DoS-style payload to render i unavailable.

Imperfect Information Abstraction. While ATL* assumes perfect information, defenders in this scenario must act under partial observability. Since model checking ATL* with imperfect information is undecidable under perfect recall strategies [4], we approximate this setting with the perfect-information framework using a dynamic action restriction protocol.

Each defender maintains a per-server suspicion bucket $\sigma_{j,i} \in \{0, 1, 2\}$ which is updated based on observed flags after a `monitor` action. The bucket determines which defensive actions D_j may invoke on i. This mechanism prevents defenders from unrealistically reacting to attacker actions they cannot observe.

Suspicion Heuristics. To compute suspicion levels from observed flags F_i, we define four heuristics reflecting different defensive philosophies.

1. **Conservative** (risk weighted): each flag $F_{i,k} \in \{0, 1\}$ carries a danger weight w_k (e.g., higher for exploit or admin process flags). Defender D_j computes the weighted sum $S_i = \sum_{k=1}^{6} w_k F_{i,k}$, and applies thresholds $0 \le T_1 < T_2$ so that $\sigma_{j,i} = 0$ if $S_i < T_1$, $\sigma_{j,i} = 1$ if $T_1 \le S_i < T_2$, and $\sigma_{j,i} = 2$ otherwise.
2. **Aggressive** (zero-trust): immediate escalation to $\sigma = 2$ if critical flags are seen (e.g., admin process).
3. **Proportional**: normalised version of the conservative heuristic, following percentile-based risk matrices approaches.
4. **Diversity**: suspicion based on the number of flag types seen, inspired by the MITRE cyber kill-chain phases [23].

Evaluation. Using the symbolic finite-trace version of ATL*AS, we evaluate verification performance across three attacker objectives, as the model size increases (via time steps n and defender budget b), using the formula $\langle\langle$Attacker$\rangle\rangle$FG compromised_1. Figure 6 shows that verification remains tractable even for models with over one million states, completing in under one hour. We also verify strategic properties over defender coalitions. For instance, we compute the minimal defender budget required to ensure that

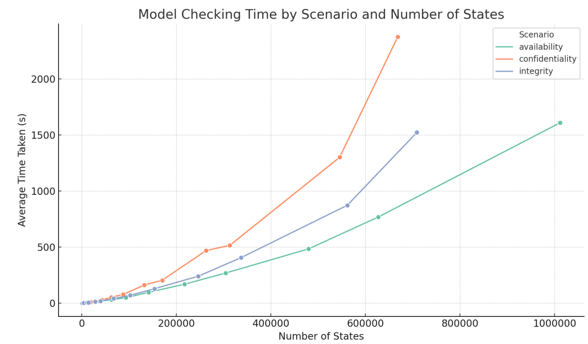

Figure 6: Runtime of symbolic model checker for different attacker goals as number of states increases.

no server remains compromised in the confidentiality scenario after 10 steps, encoded as $\langle\langle D_1, D_2\rangle\rangle$FG¬compromised_1. We find the minimal budget $b \in [1..20]$ for which the formula holds under four defender heuristics, and thus also the most cost-effective strategy for the defenders in this model.

Heuristic	Minimum Budget
Proportional	10
Aggressive	12
Conservative	14
Diversity	14

Table 3: Minimum budget required to guarantee defence under each heuristic.

These results show how ATL$^*_f$ model checking can offer formal guarantees on the effectiveness of defensive strategies, revealing optimal operational thresholds under uncertainty.

7 CONCLUSIONS

In this paper we introduced symbolic algorithms for model checking strategy logics ATL$^*_f$ (Sec. 3) and ATL* (Sec. 4) over finite and infinite traces respectively. In the ATL*AS model checker, alongside an explicit ATL$^*_f$ baseline (Sec. 5). Experimental results (Sec. 6) demonstrate that the symbolic ATL$^*_f$ algorithm achieves significant performance gains over its explicit counterpart, and that the symbolic ATL* model checker outperforms the current state-of-the-art MCMAS-SL[1G] tool despite their shared 2EXPTIME complexity. The evaluation further confirms that finite-trace verification is considerably more efficient than infinite-trace approaches due to simpler automata constructions and model checking procedures. The tool's applicability was demonstrated through a cybersecurity scenario, showcasing its utility in verifying attacker-defender strategies in complex multi-agent systems (Sec. 6.4). Extending the proposed approach to multi-valued variants of ATL*, where strategic abilities are interpreted in a graded manner, is a promising direction for future work.

Acknowledgements. The research presented in this paper was supported by the EPSRC grant number EP/X015823/1, "An abstraction-based technique for Safe Reinforcement Learning".

REFERENCES

[1] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. 2002. Alternating-time temporal logic. *J. ACM* 49 (09 2002), 672–713. https://doi.org/10.1145/585265.585270

[2] Suguman Bansal, Yash Kankariya, and Yong Li. 2024. DAG-Based Compositional Approaches for LTLf to DFA Conversions. *Formal Methods in Computer-Aided Design* (2024). https://doi.org/10.34727/2024/isbn.978-3-85448-065-5

[3] Francesco Belardinelli, Alessio Lomuscio, Aniello Murano, and Sasha Rubin. 2018. Alternating-time Temporal Logic on Finite Traces. , 77-83 pages. https://www.ijcai.org/Proceedings/2018/0011.pdf

[4] Raphaël Berthon, Bastien Maubert, and Aniello Murano. 2017. Decidability Results for ATL* with Imperfect Information and Perfect Recall. *Adaptive Agents and Multi-Agents Systems* (05 2017), 1250–1258. https://doi.org/10.5555/3091125.3091299

[5] Randal E. Bryant. 1986. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. Comput.* C-35 (08 1986), 677–691. https://doi.org/10.1109/tc.1986.1676819

[6] Petr Cermák, Alessio Lomuscio, Fabio Mogavero, and Aniello Murano. 2014. MCMAS-SLK: A Model Checker for the Verification of Strategy Logic Specifications. *Computer Aided Verification* 8559 (2014), 525–532.

[7] Petr Cermák, Alessio Lomuscio, and Aniello Murano. 2015. Verifying and Synthesising Multi-Agent Systems against One-Goal Strategy Logic Specifications. *Proceedings of the AAAI Conference on Artificial Intelligence* 29 (2015). Issue 1.

[8] Krishnendu Chatterjee, Wolfgang Dvořák, Monika Henzinger, and Alexander Svozil. 2018. Quasipolynomial Set-Based Symbolic Algorithms for Parity Games. *EPiC series in computing* 57 (10 2018), 233–211. https://doi.org/10.29007/5z5k

[9] Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. 2010. Strategy logic. *Inf. Comput.* 208, 6 (2010), 677–693. https://doi.org/10.1016/J.IC.2009.07.004

[10] Taolue Chen and Jian Lu. 2007. Probabilistic Alternating-time Temporal Logic and Model Checking Algorithm. In *Fourth International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2007, 24-27 August 2007, Haikou, Hainan, China, Proceedings, Volume 2*, J. Lei (Ed.). IEEE Computer Society, 35–39. https://doi.org/10.1109/FSKD.2007.458

[11] L de Alfaro, T.A Henzinger, and R Majumdar. 2002. From verification to control: dynamic programs for omega-regular objectives. *Proceedings 16th Annual IEEE Symposium on Logic in Computer Science* 208 (11 2002), 279–290. https://doi.org/10.1109/lics.2001.932504

[12] Giuseppe De Giacomo and Moshe Y. Vardi. 2013. Linear temporal logic and linear dynamic logic on finite traces. *IJCAI '13: Proceedings of the Twenty-Third international joint conference on Artificial Intelligence* (09 2013), 854–860. https://dl.acm.org/doi/10.5555/2540128.2540252

[13] Giuseppe De Giacomo and Moshe Y. Vardi. 2015. Synthesis for LTL and LDL on finite traces. *IJCAI'15: Proceedings of the 24th International Conference on Artificial Intelligence* (07 2015), 1558–1564. https://dl.acm.org/doi/10.5555/2832415.2832466

[14] Ali Dorri, Salil S. Kanhere, and Raja Jurdak. 2018. Multi-Agent Systems: A Survey. *IEEE Access* 6 (2018), 28573–28593. https://doi.org/10.1109/access.2018.2831228

[15] E.Allen Emerson, Charanjit S. Jutla, and A.Prasad Sistla. 2001. On model checking for the mu-calculus and its fragments. *Theoretical Computer Science* 258 (04 2001), 491–522. https://doi.org/10.1016/S0304-3975(00)00034-7

[16] Javier Esparza, Jan Křetínský, Jean-François Raskin, and Salomon Sickert. 2017. From LTL and Limit-Deterministic Büchi Automata to Deterministic Parity Automata. *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2017)* 10205 (03 2017), 426–442. https://link.springer.com/chapter/10.1007/978-3-662-54577-5_25

[17] Giuseppe De Giacomo and Marco Favorito. 2021. Compositional Approach to Translate LTLf/LDLf into Deterministic Finite Automata. *Proceedings of the International Conference on Automated Planning and Scheduling* 31 (05 2021), 122–130. https://doi.org/10.1609/icaps.v31i1.15954

[18] Erich Grädel, Wolfgang Thomas, and Thomas Wilke. 2002. *Automata logics, and infinite games : a guide to current research.* Springer.

[19] Wojciech Jamroga and Wiebe van der Hoek. 2004. Agents that Know How to Play. *Fundam. Informaticae* 63, 2-3 (2004), 185–219. http://content.iospress.com/articles/fundamenta-informaticae/fi63-2-3-05

[20] Jan Křetínský, Tobias Meggendorfer, Salomon Sickert, and Christopher Ziegler. 2018. Rabinizer 4: From LTL to Your Favourite Deterministic Automaton. *Lecture Notes in Computer Science* 10981 (2018), 567–577. https://doi.org/10.1007/978-3-319-96145-3_30

[21] Eelke Landsaat. 2022. *A Model Checker for Game Logic via Parity Games.* Ph.D. Dissertation. University of Groningen. https://fse.studenttheses.ub.rug.nl/28126/

[22] Alessio Lomuscio, Hongyang Qu, and Franco Raimondi. 2015. MCMAS: an open-source model checker for the verification of multi-agent systems. *International Journal on Software Tools for Technology Transfer* 19 (04 2015), 9–30. https://doi.org/10.1007/s10009-015-0378-x

[23] Lockheed Martin. 2025. Cyber Kill Chain. https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html

[24] Fabio Mogavero, Aniello Murano, Giuseppe Perelli, and Moshe Y. Vardi. 2014. Reasoning About Strategies. *ACM Transactions on Computational Logic* 15 (08 2014), 1–47. https://doi.org/10.1145/2631917

[25] Fabio Mogavero, Aniello Murano, Giuseppe Perelli, and Moshe Y. Vardi. 2014. Reasoning About Strategies: On the Model-Checking Problem. *ACM Trans. Comput. Log.* 15, 4 (2014), 34:1–34:47. https://doi.org/10.1145/2631917

[26] Amir Pnueli. 1977. The temporal logic of programs. *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)* (09 1977). https://doi.org/10.1109/sfcs.1977.32

[27] Amir Pnueli. 1986. Applications of Temporal Logic to the Specification and Verification of Reactive Systems: A Survey of Current Trends. In *Current Trends in Concurrency, Overviews and Tutorials*, J. W. de Bakker, Willem P. de Roever, and Grzegorz Rozenberg (Eds.). Lecture Notes in Computer Science, Vol. 224. Springer, 510–584. https://doi.org/10.1007/BFB0027047

[28] Jeremy Sieke, Lie-Quan Lee, and Andrew Lumsdaine. 2025. *Boost Graph Library: Adjacency List - 1.87.0.* Boost.org. https://www.boost.org/doc/libs/1_87_0/libs/graph/doc/adjacency_list.html

[29] Fabio Somenzi. 2025. CUDD: CU Decision Diagram Package Release 2.4.1. https://web.mit.edu/sage/export/tmp/y/usr/share/doc/polybori/cudd/cuddIntro.html

[30] whitemech. 2021. GitHub - whitemech/finite-synthesis-datasets: Datasets for Finite Synthesis. https://github.com/whitemech/finite-synthesis-datasets/tree/main

[31] whitemech. 2021. GitHub - whitemech/LTLf2DFA: From LTLf / PPLTL to Deterministic Finite-state Automata (DFA). https://github.com/whitemech/LTLf2DFA

[32] Jacob Wiebe, Ranwa Al Mallah, and Li Li. 2023. Learning Cyber Defence Tactics from Scratch with Multi-Agent Reinforcement Learning. https://arxiv.org/abs/2310.05939

[33] Wieslaw Zielonka. 1998. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science* 200 (06 1998), 135–183. https://doi.org/10.1016/s0304-3975(98)00009-7