Wallet ATL: Towards Reliable Smart Contract Verification

Angelo Ferrando
University of Modena and Reggio Emilia
Modena, Italy
angelo.ferrando@unimore.it

Blondelle Kana Zanlefack
University of Modena and Reggio Emilia
Modena, Italy
blondellekana0@gmail.com

Vadim Malvone
Télécom Paris
Institut Polytechnique de Paris
Palaiseau, France
vadim.malvone@telecom-paris.fr

ABSTRACT

The exponential growth of Decentralized Finance (DeFi) has underscored the critical need for formal verification methods that can reason about the financial properties of smart contracts. Traditional formal methods such as *Alternating-time Temporal Logic (ATL)* cannot express liquidity properties—guarantees about users' ability to access assets based on wallet balances. We introduce *Wallet ATL (WATL)*, an extension of ATL with wallet predicates and financially constrained strategic operators. WATL ensures that actions are both strategically and economically feasible. We formalize the semantics of WATL, provide model checking algorithms within the *VITAMIN* framework, and address scalability through the *Meta-Agent Abstraction*, which collapses all non-coalition agents into a single meta-agent with a sum-aggregated wallet. This abstraction preserves liquidity properties while significantly reducing the verification space. Through case studies such as a *crowdfunding smart contract*, we demonstrate how WATL formally specifies and verifies liquidity guarantees. Our results show that WATL, implemented in the VITAMIN tool, bridges the gap between multi-agent strategic reasoning and financial correctness, providing a practical step towards the formal verification of smart contracts with liquidity-awareness.

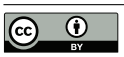
KEYWORDS

Model Checking; Multi-Agent Systems; Smart Contracts; Liquidity; Strategic Logics

ACM Reference Format:
Angelo Ferrando, Blondelle Kana Zanlefack, and Vadim Malvone. 2026. Wallet ATL: Towards Reliable Smart Contract Verification. 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/

1 INTRODUCTION

Smart contracts are self-executing programs deployed on blockchain platforms[26], where digital assets are transferred and managed according to predefined rules. Their transparency, immutability, and automation enable novel applications in finance, supply chains, and decentralized governance[5, 26]. However, the same properties that make smart contracts powerful also make them error-prone and difficult to repair: once deployed, vulnerabilities may lead to irrevocable financial loss [5, 6, 14].



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/

A particularly critical aspect of smart contract correctness is *liquidity* [12, 24]—the guarantee that agents can withdraw or transfer their funds according to the contract's intended rules. Unlike safety or liveness[25], liquidity properties depend not only on the logical structure of the contract but also on the availability of financial resources. Ensuring liquidity requires reasoning simultaneously about strategies of interacting agents and their wallet balances.

Traditional logics for Multi-Agent Systems, like Alternating-time Temporal Logic (ATL) [4, 21], provide powerful tools for reasoning about strategic abilities of coalitions. Yet, they are insufficient for smart contracts because they cannot express whether a strategy is financially feasible. In particular, ATL assumes that agents can execute any action permitted by the transition system, ignoring whether the action requires tokens, deposits, or payments.

To address this limitation, we introduce *Wallet Alternating-time Temporal Logic (WATL)*, an extension of ATL with wallet predicates and wallet-constrained strategy operators. WATL allows specifying and verifying properties such as: "Coalition A can ensure eventual payout, provided each member has sufficient funds to perform their part of the strategy". We formalize WATL's syntax and semantics over Wallet Concurrent Game Structures (WCGS), and we present a model checking algorithm[9, 22] that extends fixed-point techniques used in ATL.

Since the state space of WCGS grows rapidly with the number of agents and wallet values, we further propose an abstraction technique—*Meta Agent abstraction*—that collapses all non-coalition agents into a single meta-agent with a sum-aggregated wallet. This reduces the complexity of verification while preserving wallet-based strategic properties of the coalition.

Our contributions are threefold:

(1) A new logic with wallet constraint (WATL) to express liquidity properties in smart contracts.
(2) A verification algorithm integrated into the *VITAMIN* [19] tool, extending model checking to WATL.
(3) Introduce WATL Meta Agent abstraction to reduce state space and enable tractable verification.

Through detailed case studies, including auction and crowdfunding contracts, we demonstrate that WATL provides a precise and efficient framework for the formal verification of liquidity in smart contracts. Note that, although smart contracts are the primary motivation, WATL applies broadly to MAS scenarios involving financial or resource transfers, such as auctions, economic simulations, and market-driven multi-agent environments.

1.1 Related Work

Our work on WATL intersects with several lines of research, primarily in formal verification of smart contracts, secondly in strategic

reasoning with resource constraints and finally the abstraction that ensures practical scalability.

Formal Verification of Smart Contracts. A significant body of work applies formal methods to smart contracts, ranging from theorem proving [17] to model checking [12, 23]. The *Solvent* tool [12] focuses on detecting insolvency vulnerabilities but operates at the code level. In contrast, *WATL* offers a logical framework for specifying and verifying general strategic liquidity properties across agents. Game-theoretic models [17] capture strategic behavior but lack integration with temporal logic, hindering automated verification of dynamic financial interactions.

Strategic Logics with Resource Constraints. ATL [4, 18] provides the basis for reasoning about coalitional abilities but assumes that all actions are feasible. *Resource-Bounded ATL (RB-ATL)* [2, 15] incorporates resource limits but employs static, global bounds that cannot model the dynamic, transferable nature of blockchain tokens. It also lacks per-agent wallet predicates, which are essential for expressing balance-dependent feasibility. Other extensions—such as bounded temporal logics [20] or probabilistic ATL (*pATL*) [16]—address temporal or stochastic constraints rather than economic ones. *WATL* extends these frameworks by introducing *wallet-awareness* and *financial executability* as first-class concepts.

Abstraction and Scalability. State-space explosion is a persistent challenge in multi-agent verification. Existing abstraction methods [7, 8, 10, 11] reduce complexity but are not designed for financially constrained systems. Our *Meta-Agent Abstraction* specifically targets *WCGS*, collapsing all non-coalition agents into an aggregated meta-agent while preserving wallet-sensitive strategies. This approach maintains the correctness of liquidity properties and drastically reduces computational complexity.

Moreover, while existing research has addressed correctness, resource-bounded logics, or scalability independently, *WATL* fills this gap by combining strategic expressiveness with explicit financial feasibility, supported by an abstraction technique that ensures practical scalability.

2 BACKGROUND

The formal verification of multi-agent systems requires logics capable of expressing strategic interactions between autonomous entities. Alternating-time Temporal Logic (ATL) is a powerful formal logic specifically designed for reasoning about the strategic capabilities of agents in Multi-Agent Systems (MAS) [4]. Unlike traditional temporal logics such as LTL or CTL [9, 13], which focus on what must or can happen in a system, ATL explicitly addresses what a coalition of agents can enforce or achieve through their coordinated actions, regardless of the choices made by other agents. This makes ATL particularly suitable for analyzing systems where autonomous entities interact and make decisions that collectively determine the system's evolution, such as in game theory [4, 6], distributed systems [14], and, increasingly, blockchain-based smart contracts [17, 24].

The syntax of ATL is defined by the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\langle A\rangle\rangle X\varphi \mid \langle\langle A\rangle\rangle G\varphi \mid \langle\langle A\rangle\rangle \varphi\, U\, \varphi$$

where p is an atomic proposition, $A \subseteq Ag$ represents a coalition of agents, and X (next), G (globally), and U (until) are temporal operators. The operator F (eventually) is derived as $F\varphi \equiv \text{true}\, U\, \varphi$.

The semantics of ATL is defined over *Concurrent Game Structures* (CGS) = $(Ag, Q, \Pi, \pi, Act, d, \delta)$ which include a set of agents Ag, a finite set of states Q, a valuation function π that specifies which propositions hold in each state, and a transition function δ that determines the next state based on the current state and the actions selected by all agents. The key innovation of ATL is the *strategy-based semantics* [4]: a state satisfies $\langle\langle A\rangle\rangle\varphi$ if there exists a collective strategy for coalition A such that, when A follows this strategy, the temporal property φ is guaranteed to hold regardless of how other agents behave.

Strategies in ATL can be either *perfect recall strategies* (depending on the entire history of states) or *memoryless strategies* (depending only on the current state). A collective strategy for coalition A is a tuple containing individual strategies for each agent in A. The outcome function $out(q, s_A)$ returns all possible paths that may occur when coalition A follows strategy s_A from state q.

The model-checking complexity for ATL is PTIME-complete, making it computationally more tractable than its extension ATL*, where arbitrary nesting of temporal operators yields 2EXPTIME-completeness [1, 4]. This balance between expressiveness and complexity has made ATL a fundamental framework for specifying and verifying strategic abilities in multi-agent systems, serving as the basis for numerous extensions, including the *Wallet Alternating-time Temporal Logic (WATL)* variant we present in this work.

The strategic perspective of ATL provides the foundation for our work on WATL, which extends these concepts to incorporate resource constraints that directly affect agents' strategic capabilities.

3 OUR MODEL

Resource availability—particularly liquidity in the sense of wallet balances—is central to reasoning about smart contracts. Standard Concurrent Game Structures (CGS) [4] capture multi-agent strategies but assume that every action listed in the protocol is always feasible. This is unrealistic in blockchain systems[14, 28], where agents are constrained by their wallets. To address this, we extend CGS into *Wallet Concurrent Game Structures (WCGS).*

3.1 Formal Definition

Definition 3.1. A *Wallet Concurrent Game Structure (WCGS)* is a tuple $G = (Ag, Q, \Pi, \pi, Act, d, \delta, W)$, where:

- Ag: a finite set of agents including the contract itself.
- Q: a finite set of global system states.
- Π: a set of smart contract atomic propositions[1].
- $\pi : Q \to 2^{\Pi}$: a labeling function that assigns to each state the set of propositions that hold in it.
- Act: the set of parameterized actions, partitioned into three disjoints categories: $Act = Act^{\downarrow} \cup Act^{\uparrow} \cup Act^{0}$, where $Act^{\downarrow} \cap Act^{\uparrow} \cap Act^{0} = \emptyset$.

[1]Since the verification process is related to checking the correctness of the contract, the atoms are tied to its evolution and do not convey information about the other agents involved in the MAS. Information about the latter can be derived through the wallet function.

- **Consumption actions** ($Act^{\downarrow}$): actions that reduce the wallet of the executing agent, e.g., $Act^{\downarrow} = \{deposit(v), bid(v), stake(v), transfer(v, recipient), donate(v)\}, v \in \mathbb{N}^{+}$, An action $\alpha_c \in Act^{\downarrow}$ is feasible for agent ag in state q only if $W(q, a) \geq v$.
- **Income actions** ($Act^{\uparrow}$): actions that increase the wallet of the executing agent, e.g., $Act^{\uparrow} = \{withdraw(v), reclaim(v), redeem(v), refund(v), unstake(v)\}$.
- **Neutral actions** (Act^{0}): actions that do not change the wallet balance of the executing agent, e.g., $Act^{0} = \{close(), openAuction(), approve(), reject()\}$. These actions may change the system state but have no direct financial cost or reward.

- $d : Ag \times Q \rightarrow 2^{Act}$: the protocol function specifying the set of moves available to an agent $ag \in Ag$ in state $q \in Q$. It considers both contract rules and wallet feasibility: $d(a, q) = \{\alpha_c \in Act \mid \alpha_c \text{ enabled in } q \wedge (\alpha_c \in Act^{\uparrow} \cup Act^{0} \vee (\alpha_c \in Act^{\downarrow} \wedge W(q, a) \geq v))\}$.
- $\delta : Q \times (Act_{a_1} \times \cdots \times Act_{a_n}) \rightarrow Q$: the transition function, mapping a state and the joint actions of all agents to a successor state.
- $W : Q \times Ag \rightarrow \mathbb{Q}^{+}$: the wallet function, assigning each agent a non-negative balance in every state.

This definition extends the standard Concurrent Game Structure (CGS) with perfect information used in ATL[4] by introducing *economic feasibility* as a constraint on available actions. In classical ATL, the protocol function d depends only on the game rules: every enabled action can be executed. In contrast, in WCGS, action availability additionally depends on the agent's wallet balance. Actions in $Act^{\uparrow}$ are always permitted, as they increase the wallet. Actions in $Act^{\downarrow}$ are restricted to agents with sufficient balance $W(q, a)$. This ensures that strategies in WATL are not only strategically possible but also *financially executable*, thereby capturing liquidity constraints inherent to smart contracts.

Example 3.2. As an example, we now define a simplified crowdfunding campaign where a project seeks to raise funds from two potential contributors, namely Alice (Agent 1) and Bob (Agent 2). The campaign is managed by a smart contract (Agent 3), which acts as an autonomous and impartial agent for the collection of funds. The contract's role is to accept contributions, track the total amount raised, and ultimately determine if the funding goal has been met. For the campaign, Alice and Bob want to support a *community-driven project* such as the renovation of a local library. The contract guarantees that their contributions are secure and transparent: if enough funds are collected, the project can proceed; otherwise, the backers retain the ability to withdraw their deposits.

Alice and Bob may choose to deposit a fixed amount of tokens ($D20$), withdraw ($W20$), or remain idle (I). We encode the contract actions using numbers which represent the *id* of each agent and 0 is for no operation for the contract agent, that is: 1 (accept from Alice), 2 (accept from Bob), 0 (no operation). For an intuitive definition of the crowdfunding WCGS, see Figure 1. Note that, each self-loop transition labeled $(*, *, *)$ represents all possible combinations of actions by the agents (Alice, Bob, and the Contract) that are not explicitly defined by another outgoing transition from that state.

Here we have $k = 3$ players: Alice, Bob, and the Smart Contract. The set of states is defined as $Q = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6\}$. The initial state s_0 corresponds to both Alice and Bob holding their full balance and the pool being empty. States s_1 and s_2 represent a single agent deposit, while s_3 corresponds to both agents having contributed. States s_4 and s_5 capture partial withdrawals, and s_6 represents the final outcome.

The set of propositions consists of seven elements:
$AP = \{pool\_empty, partial\_deposited, deposited, withdraw\_done, goal\_achieved, campaign\_closed\}$.

For clarity in state of the model in Figure 1, we use the following abbreviations: *pe* for *pool_empty*, *pd* for *partial_deposited*, *dp* for *deposited*, *ga* for *goal_achieved*, *wd* for *withdraw_done*, and *cc* for *campaign_closed*. The labeling function and the wallet can be read directly from the graph. For the actions we have,

$$\text{Act}_{\downarrow} = \{D20\} \quad \text{(deposit reduces wallet)},$$
$$\text{Act}_{\uparrow} = \{W20\} \quad \text{(withdraw increases wallet)}, I \text{ is neutral.}$$

For simplicity, we will omit a formal definition of the action function d and the transition function δ, as these are implicitly defined by the transition-arcs of the graph.

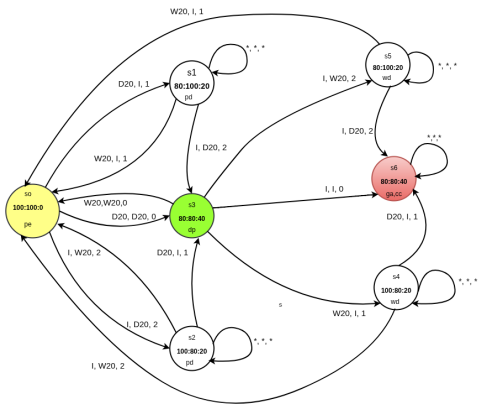


Figure 1: WCGS for Crowdfunding Campaign

The color-coded state transitions visualize the campaign lifecycle: ■ (s_0) represents the initial open campaign with an empty pool; (s_1, s_2) denote partial commitments, where one of the two contributors has deposited funds. The campaign has progressed but is still in an intermediate phase, awaiting further action. ■ (s_3) marks the critical threshold where both parties have contributed, and the funding target has been met. In this phase, the campaign remains open, and contributors still have the right to withdraw their funds. (s_4, s_5) signify completed withdrawals where contributors can reclaim funds; and ■ (s_6) denotes the successful campaign closure where the funding goal is achieved and the project can proceed.

3.2 Dynamics

The wallet function W evolves along with the system state, updated according to the transition function δ.

Consumption Action. If in state q an agent a executes a consumption action $Act^{\downarrow}$ (e.g., token contributions) towards the contract agent c, then in the successor state q' the wallets are updated as:

$$W(q', a) = W(q, a) - v, \qquad W(q', c) = W(q, c) + v,$$

while all other wallets remain unchanged.

Income Action. If in state q an agent a executes an income action $Act^{\uparrow}$ (e.g., reclaiming tokens) from the contract agent c, then in the successor state q':

$$W(q', a) = W(q, a) + v, \qquad W(q', c) = W(q, c) - v,$$

while other wallets remain unchanged.

Example 3.3. **Illustration (based on Figure 1).** Consider the initial state s_0, where: $W(s_0, Alice) = 100, W(s_0, Bob) = 100, W(s_0, Contract) = 0$. If Alice executes a consumption action $deposit(20)$, the system transitions to state s_1, with updated wallets: $W(s_1, Alice) = 100 - 20 = 80, W(s_1, Contract) = 0 + 20 = 20$, while Bob's balance remains unchanged: $W(s_1, Bob) = 100$.

This dynamic reflects the economic semantics of smart contracts: actions consume or reallocate liquidity, shaping which future actions remain feasible.

4 OUR LOGIC

The *Wallet Alternating-time Temporal Logic* (WATL) is a strategic logic designed to specify and reason about the abilities of agents and coalitions under wallet constraints. It extends standard temporal and strategic logics to account explicitly for the availability of financial resources.

4.1 Syntax

WATL formulas are built from atomic propositions using standard Boolean and temporal operators, extended with a *wallet predicate* and a *wallet-constrained strategy operator*. Let Ag be the set of agents. WATL formulas are defined by the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid wallet(a, \triangleright v) \mid \langle\!\langle A : \bigwedge_{a \in Ag} wallet(a, \triangleright v_a) \rangle\!\rangle \psi$$

$$\psi ::= X\psi \mid \psi_1 \, U \, \psi_2 \mid G\psi$$

where $p \in \Pi$ is an atomic proposition; $wallet(a, \triangleright v)$ is a *wallet predicate*, which expresses that agent $a \in Ag$ satisfies the wallet condition relative to value v with the comparison operator $\triangleright \in \{=, <, \leq, >, \geq\}$; and $\langle\!\langle A : \bigwedge_{a \in Ag} wallet(a, \triangleright v_a) \rangle\!\rangle \psi$ states that coalition A has a strategy to enforce the path formula ψ, provided all wallet constraints are satisfied.

4.2 Semantics

The semantics of WATL is defined over a WCGS $G = (Ag, Q, \Pi, \pi, Act, d, \delta, W)$, which includes categorized actions and wallet-dependent protocols (a protocol is wallet-dependent if the availability of certain actions depends on the agent's current wallet balance) . We denote by Q^{ω} the set of infinite sequences of states (*computations*). For a computation $\lambda = q_0 q_1 \ldots \in Q^{\omega}$ we write $\lambda[i] = q_i$ for the i-th state, and $\lambda[i, j] = q_i \ldots q_j$ for the subsequence from q_i to q_j. The set of finite non-empty sequences of states is denoted by Q^{+}.

Definition 4.1 (Feasible Actions). Given a WCGS G, a state $q \in Q$, and an agent $a \in Ag$, an action $\alpha \in Act$ is *feasible* for a in q if:

(1) $\alpha \in d(a, q)$ (enabled by protocol), and
(2) if $\alpha \in Act^{\downarrow}$ with parameter v, then $W(q, a) \geq v$.

Definition 4.2 (Joint Moves and Outcomes). A *joint move* for coalition $A \subseteq Ag$ at state q is a tuple $m_A = (m_a)_{a \in A}$ where each $m_a \in d(a, q)$ is feasible. The set of all such moves is $D_A(q)$.

The *outcome* of $m_A \in D_A(q)$ is

$$out(q, m_A) = \{\, q' \in Q \mid \exists m \in D(q) : m_A = (m_a)_{a \in A} \wedge q' = \delta(q, m) \,\}.$$

Definition 4.3 (Strategies). A *strategy* for coalition $A \subseteq Ag$ is a function $\sigma_A : Q^{+} \to \bigcup_{q \in Q} D_A(q)$ mapping every history $\lambda q \in Q^{+}$ to a feasible move $\sigma_A(\lambda q) \in D_A(q)$.

A computation $\lambda = q_0 q_1 \ldots \in Q^{\omega}$ is *consistent* with σ_A if for all $i \geq 0$, $\lambda[i+1] \in out(\lambda[i], \sigma_A(\lambda[0, i]))$.

The set of all such computations from q is $out(q, \sigma_A)$.

Definition 4.4 (Wallet Update). For a transition $q \xrightarrow{m} q'$ with $m = (m_a)_{a \in Ag}$, wallets update as:

$$\begin{aligned} &\text{if } m_a \in Act^{\downarrow}(v): && W(q', a) = W(q, a) - v, \\ &\text{if } m_a \in Act^{\uparrow}(v): && W(q', a) = W(q, a) + v, \\ &\text{otherwise } m_a \in Act^{0}: && W(q', a) = W(q, a). \end{aligned}$$

Definition 4.5 (C-Consistent Computations). Given a constraint $C = \bigwedge_{a \in A} wallet(a, \triangleright v_a)$, a computation $\lambda \in out(q, \sigma_A)$ is *C-consistent* if in the initial state q,

$$\forall a \in A: \quad W(q, a) \triangleright v_a.$$

We denote by $out(q, \sigma_A, C)$ the set of all such computations.

Truth Definition for WATL. Given a WCGS G and state $q \in Q$, the satisfaction relation $G, q \models \varphi$ is defined inductively as follows:

$$\begin{aligned} G, q \models p &\iff p \in \pi(q) \\ G, q \models \neg\varphi &\iff G, q \not\models \varphi \\ G, q \models \varphi_1 \vee \varphi_2 &\iff G, q \models \varphi_1 \text{ or } G, q \models \varphi_2 \\ G, q \models wallet(a, \triangleleft v) &\iff W(q, a) \ \triangleright \ v \\ G, q \models \langle\!\langle A : C \rangle\!\rangle \psi &\iff \exists \text{ C-strategy } \sigma_A \\ &\qquad \forall \lambda \in out(q, \sigma_A, C),\ G, \lambda \models \psi \end{aligned}$$

The satisfaction relation for path formulas ψ is defined over computations $\lambda = q_0, q_1, q_2, \ldots$ as follows:

$$G, \lambda \models X\psi \iff G, \lambda^1 \models \psi, \text{ where } \lambda^1 = q_1, q_2, \ldots$$

$$G, \lambda \models G\psi \iff \forall i \geq 0,\ G, \lambda^i \models \psi$$

$$G, \lambda \models \psi_1 U \psi_2 \iff \exists i \geq 0,\ G, \lambda^i \models \psi_2 \ \wedge \ \forall 0 \leq j < i,\ G, \lambda^j \models \psi_1$$

This semantics captures financial dynamics of smart contracts, where actions directly modify wallets and strategies depend both on protocol rules and economic feasibility.

4.3 Relation with existing logics

WATL is a conservative extension of ATL [4]. If all wallet constraints C in a formula are trivially satisfied (e.g., $v_a = 0$ for all $a \in A$), the operator $\langle\langle A : C\rangle\rangle\psi$ coincides with the standard ATL operator $\langle\langle A\rangle\rangle\psi$. Likewise, if the wallet function in a WCGS imposes no constraints on action availability, the model is behaviorally identical to a standard CGS. In this sense, ATL forms a syntactic and semantic fragment of WATL.

The novelty of WATL does not lie in merely accounting for resource usage, but in capturing *wallet-constrained strategic feasibility*. Unlike RB±ATL [2, 3], which reasons about cumulative path costs along executions, WATL embeds per-agent wallet balances directly into the system state and enforces that actions are enabled only if the executing agent can afford them at the time of execution. As a result, WATL rules out strategies that require temporary overdrafts or infeasible intermediate transitions, which cannot be excluded in RB±ATL. For example, an agent with an initial wallet of 10 tokens may, in RB±ATL, be allowed to execute a strategy that incurs a cost of 100 tokens and later gains 95 tokens, since the net cost is 5. In WATL, such a strategy is impossible, as the 100-token action is not enabled in the initial state. Consequently, WATL and RB±ATL are semantically incomparable, and WATL is not subsumed by RB±ATL; only WATL can exclude strategies that exceed an agent's available resources at any point during execution.

5 MODEL CHECKING ALGORITHM

The verification of WATL properties requires extending the standard ATL model checking procedure [4, 18, 23] to incorporate wallet-based constraints. In this section, we formalize the algorithm used to decide whether a WATL formula holds in a given Wallet-Constrained Game Structure (WCGS).

The main difference from classical ATL is the introduction of a *wallet predicate filter* that prunes states where the coalition cannot afford the required actions. This mechanism ensures that only feasible strategies (those consistent with agents' wallets) are considered.

This algorithm has been implemented in the VITAMIN [19] tool, which has been extended to support WATL model checking. The implementation allows users to specify WATL formulas, construct Wallet Concurrent Game Structures, and automatically verify liquidity-related properties. In particular, VITAMIN integrates the wallet predicate filter into its symbolic fixed-point computation, ensuring that the semantics of WATL are faithfully captured in practice.

5.1 Model Checking Procedure

The main model checking algorithm mcheck_watl(G, φ) proceeds bottom-up on the structure of the formula, extending the standard ATL fixed-point computation [4, 9, 22]. For each subformula, the algorithm computes the set of states where it holds.

The novelty lies in the treatment of strategic operators constrained by wallets:

$$\langle\langle A : \bigwedge_{a\in Ag} wallet(a, \geq v_a)\rangle\rangle\psi.$$

Before computing the pre-image for coalition A, we apply the `wPre` filter to ensure only states where the coalition can afford the strategy are retained.

Algorithm 1 Model Checking WATL

Require: φ: a WATL state formula
Require: $G = (Ag, Q, \Pi, \pi, d, Act, \delta, W)$: Wallet CGS
for all $\varphi' \in \text{Sub}(\varphi)$ **do**
  if $\varphi' = p$ **then**
    $[[\varphi']] \leftarrow \text{Reg}(p)$
  else if $\varphi' = \neg\theta$ **then**
    $[[\varphi']] \leftarrow Q \setminus [[\theta]]$
  else if $\varphi' = \theta_1 \vee \theta_2$ **then**
    $[[\varphi']] \leftarrow [[\theta_1]] \cup [[\theta_2]]$
  else if $\varphi' = \langle\langle A : \bigwedge_{a\in A} wallet(a, \triangleright v_a)\rangle\rangle X\theta$ **then**
    $[[\varphi']] \leftarrow wPre(G, A, \{\triangleright v_a\}_{a\in A}, Pre(A, [[\theta]]))$
  else if $\varphi' = \langle\langle A : \bigwedge_{a\in A} wallet(a, \triangleright v_a)\rangle\rangle G\theta$ **then**
    $\rho \leftarrow [[true]]$
    $\tau \leftarrow [[\theta]]$
    while $\rho \neq \tau$ **do**
      $\rho \leftarrow \tau$
      $\tau \leftarrow wPre(G, A, \{\triangleright v_a\}_{a\in A}, Pre(A, \rho)) \cap [[\theta]]$
    end while
    $[[\varphi']] \leftarrow \rho$
  else if $\varphi' = \langle\langle A : \bigwedge_{a\in A} wallet(a, \triangleright v_a)\rangle\rangle \theta_1 \; U \; \theta_2$ **then**
    $\rho \leftarrow [[false]]$
    $\tau \leftarrow [[\theta_2]]$
    while $\rho \neq \tau$ **do**
      $\rho \leftarrow \tau$
      $\tau \leftarrow \rho \cup (wPre(G, A, \{\triangleright v_a\}_{a\in A}, Pre(A, \rho)) \cap [[\theta_1]])$
    end while
    $[[\varphi']] \leftarrow \rho$
  end if
end for
return $[[\varphi]]$

Algorithm 1 – Model Checking WATL.

(1) For atomic and Boolean subformulas: proceed as in ATL[4].
(2) The function *Reg*, when given a proposition $p \in \Pi$, returns the set of states in Q that satisfy p.
(3) For strategic operators $\langle\langle A : C\rangle\rangle\psi$:
  (a) Apply the wallet predicate filter $\mathsf{wPre}(A, C, Q_s)$ to restrict feasible states.
  (b) Compute the fixed-point set of states from which coalition A has a strategy to enforce ψ, restricted to feasible states.
(4) Return the set of states where φ holds.

5.2 Wallet Predicate Filter Function

The function $\mathsf{wPre}(G, A, \{\triangleright v_a\}_{a\in A}, Q_s)$ is a core component of the model checking algorithm. It filters a set of states Q_s to return only those states where every agent a in the coalition A satisfies its wallet constraint $wallet(a, \triangleright v_a)$. This acts as a precondition for enabling coalition strategies.

Algorithm 2 – Wallet Predicate Filter.

- **Input:**
  - $G = (Ag, Q, \Pi, \pi, d, Act, \delta, W)$: the Wallet CGS
  - $A \subseteq Ag$: coalition of agents.
  - $\{v_a\}_{a \in A}$: mapping assigning each $a \in A$ a minimum required balance.
  - $Q_s \subseteq Q$: the set of states to filter the wallet constraint if it holds.
- **Output:** a set of states $Q' \subseteq Q$ such that for every $q \in Q'$ and all $a \in Ag$, we have $W(q, a) \rhd v_a$ holds.

Algorithm 2 wPre$(G, A, \{\rhd v_a\}_{a \in Ag}, Q_s)$

```
Q' ← ∅
for all q ∈ Q_s do
    ok ← true
    for all a ∈ A do
        if not W(q, a) ▷ v_a then
            ok ← false
            break
        end if
    end for
    if ok then
        Q' ← Q' ∪ {q}
    end if
end for
return Q'
```

Theorem 5.1 (Complexity of WATL Model Checking). *The model checking problem for WATL is PTIME-complete in the size of the model and of the formula*[2]*, the same as for ATL.*

Proof. It is well established that the ATL model checking problem is PTIME-complete in the size of the model [4, 9, 27]. WATL extends ATL by enriching the semantics of strategic operators with wallet constraints. The only additional step introduced by this extension is the evaluation of the *wPre* filter, which ensures that every agent in a coalition satisfies the required wallet balance before executing a strategy.

Let $|Q|$ denote the number of states, $|Ag|$ the number of agents, and $|Act|$ the size of the joint action space. As in ATL, the evaluation of each strategic operator is polynomial in $|Q|$ and $|Act|$. The computation of *wPre* requires scanning the balances of the coalition members at each state, which incurs an overhead bounded by $O(|Q| \cdot |A|)$, where $|A|$ is the coalition size.

This additional factor remains polynomial and does not alter the asymptotic complexity of the model checking procedure. Therefore, WATL model checking inherits the PTIME-completeness of ATL. □

6 META-AGENT ABSTRACTION FOR WCGS

In this section we introduces an abstraction technique designed to reduce the complexity of verifying properties in Wallet-Concurrent Game Structures (WCGS). The key idea is to reduce the size of the strategic space by focusing on the coalition of interest, while aggregating the behavior of all other agents into a single *meta-agent*. This abstraction makes model checking more tractable, while preserving the essential strategic interactions relevant to the coalition. In the following subsections, we present the intuition behind the abstraction, the formal construction of the abstract model and prove its soundness.

6.1 Intuition and Overview

A primary challenge in model checking multi-agent systems is *state space explosion*, where computational complexity grows intractably with the number of agents. The Wallet Concurrent Game Structure (WCGS) model is particularly susceptible, as the strategy space is influenced by the parametric actions and their possible values.

To verify properties of a specific coalition A (e.g., a protocol's participants), we introduce a ***meta-agent abstraction***. The core idea is to preserve the coalition A of interest in the strategic operator explicitly while collapsing all other agents $O = Ag \setminus A$ into a single meta-agent m. This leads to a relevant reduction in the strategic interaction space that a model checker must analyze.

6.2 Formal Definition

Let $G = (Ag, Q, \Pi, \pi, d, Act, \delta, W)$ be a concrete WCGS and φ a WATL formula. Let $A \subseteq Ag$ be the coalition of interest in φ and $O = Ag \setminus A$ be the set of opponents to be abstracted.

We define the abstract WCGS $G' = (Ag', Q', \Pi, \pi', d', Act', \delta', W')$ as follows:

Agents. The agent set consists of the original coalition A and a new meta-agent m, which represents the collective behavior of all opponents in O. That is, $Ag' = A \cup \{m\}$.

State. The original state space is partitioned into equivalence classes based on wallet distributions:

$$[q] = \{\, q' \in Q \mid \{W(q,i)\}_{i \in A} = \{W(q',i)\}_{i \in A} \wedge \sum_{j \in O} \{W(q,j)\} = \sum_{j \in O} \{W(q',j)\}\}$$

This means, two states are equivalent if they:

- preserve the exact wallet information for coalition agents $i \in A$, and
- preserve the aggregate wallet sum of all opponents $j \in O$.

The abstract state space is then defined as: $Q' = \{[q] \subseteq Q \mid q \in Q\}$.

Propositions and Labeling. Since the atomic propositions are related to the contract, we have $\Pi' = \Pi$. For the same reason, for all $[q] \in Q'$, $\pi'([q]) = \pi(q)$.

Action Sets.

- For each agent $i \in A$: $Act'_i = Act_i$ (the actions available to each agent in A remain unchanged).
- For the meta-agent m: $Act'_m = \prod_{j \in O} Act_j$. An action for m is a tuple $\alpha_m = (\alpha_j)_{j \in O}$, specifying one action for each opponent in O.

Wallet Function.

$$W'(q,i) = \begin{cases} W(q,i) & \text{if } i \in A, \\ \sum_{j \in O} W(q,j) & \text{if } i = m. \end{cases}$$

[2]Note that, wallet values are encoded in unary, following the same encoding used in most of strategic logics, such as RB-ATL and RB±ATL.

The wallet of each coalition agent $i \in A$ is preserved exactly, while the wallet of the meta-agent m is the sum of the wallets of all opponents it represents. This captures the total financial resources available to the opposition.

Protocol Function.

- For each agent $i \in A$: $d'(i, q) = d(i, q)$ (the available actions for A are unchanged).
- For the meta-agent m:

  $$[\alpha_m] = \{\alpha'_m \in \prod_{j \in O} Act_j \mid \alpha'_m \text{ is a permutation of } \alpha_m\}$$

  Formally,

$$[\alpha_m] = \{\, \alpha'_m \in \prod_{j \in O} Act_j \;\mid \forall\{\alpha_i\}_{i \in O}\ \exists!\{\alpha'_j\}_{j \in O}.\ \alpha_i = \alpha'_j \;\wedge\; \forall\{\alpha'_i\}_{i \in O}\ \exists!\{\alpha_j\}_{j \in O}.\ \alpha'_i = \alpha_j \,\}$$

$$d'(m, q) = \{[\alpha_m] \in \prod_{j \in O} d(j, q)\}$$

  That is, the meta-agent can choose any tuple of actions that are individually available to the opponents in the current state.

Transition Function.

$$\delta'([q], (\alpha_A, \alpha_m)) = \delta(q, (\alpha_A, \alpha_O))$$

where:

$$\alpha_A = (\alpha_i)_{i \in A}, \quad \alpha_m = (\alpha_j)_{j \in O}, \quad \alpha_O = \alpha_m.$$

Thus, the transition function of the abstract model G' is defined by executing the concrete transition function δ with the actions prescribed by the abstract agents.

6.3 Soundness of the Abstraction

The following theorem establishes that our abstraction is sound, meaning any definitive result from the abstract model holds in the concrete one.

Theorem 6.1 (Soundness of Meta-Agent Abstraction). *Let G be a concrete WCGS and G' its meta-agent abstraction w.r.t. a coalition $A \subseteq Ag$. Let φ be a WATL state formula where all strategic operators are limited to coalition A. Then, for any state $q \in Q$:*

(1) ***Preservation of Truth:*** $(G', [q]) \models \varphi \implies (G, q) \models \varphi$.
(2) ***Preservation of Falsehood:*** $(G', [q]) \not\models \varphi \implies (G, q) \not\models \varphi$.

Proof. By structural induction on φ. We present the case for $\varphi = \langle\langle A : C \rangle\rangle \psi$; other cases follow from standard preservation results and the Induction Hypothesis (IH).

Induction Hypothesis (IH). The theorem holds for all subformulas, including ψ. **Part 1: Preservation of Truth (tt).**
Assume $(G', [q]) \models \langle\langle A : C \rangle\rangle \psi$. Then the following hold:

(1) The constraint C holds in the abstract state $[q]$
(2) There exists a strategy σ'_A in G' such that

$$\forall \lambda \in out([q], \sigma'_A) : (G', \lambda) \models \psi.$$

We now prove that $(G, q) \models \langle\langle A : C \rangle\rangle \psi$.

(1) *Constraint C:* For $a \in A$, $W'([q], a) = W(q, a)$ by abstract construction. Thus, the wallet constraints also hold in the concrete model G.
(2) *Strategy:* Lift the abstract strategy: define $\sigma_A(a, h) = \sigma'_A(a, h)$ for $a \in A$. Let σ_O be an arbitrary opponent strategy in G. Construct σ'_m in G' from σ_O: $\sigma'_m(h') = (\sigma_o(h'))_{o \in O}$. By definition of δ', the path $\lambda = out(q, (\sigma_A, \sigma_O))$ in G is identical to $\lambda' = out([q], (\sigma'_A, \sigma'_m))$ in G'. By assumption, $(G', \lambda') \models \psi$. By IH, $(G, \lambda) \models \psi$. Since σ_O was arbitrary, σ_A is a winning strategy.

Part 2: Preservation of Falsehood (ff). This part is the symmetric of the above proof.

□

7 CASE STUDY: VERIFYING LIQUIDITY IN CROWDFUNDING

Building upon the formal WCGS model defined in Example 3.2 and presented in Figure 1, we now demonstrate how WATL enables the verification of crucial liquidity properties in smart contracts. This case study illustrates the practical application of our framework to analyze strategic behaviors under financial constraints.

We can specify some illustrative properties in WATL as follows:

$$\Phi_1 = \langle\langle \{Alice, Bob\} : wallet(Alice, \geq 20) \wedge wallet(Bob, \geq 20) \rangle\rangle \mathbf{F}\, ga$$

If Alice and Bob have at least 20 tokens each, they (as a coalition) can force that the campaign eventually reaches the funding goal.

$$\Phi_2 = \langle\langle \{Alice\} : wallet(Alice, \geq 20) \rangle\rangle \mathbf{F}(\neg ga \rightarrow \mathbf{F}\, wd)$$

Alice has a wallet-feasible strategy that guarantees that whenever the goal is not reached, she can eventually withdraw her funds.

$$\Phi_3 = \langle\langle \{Alice, Contract\} : wallet(Alice, \geq 20) \rangle\rangle \mathbf{F}\, pd$$

If Alice has at least 20 tokens and cooperates with the smart contract, then they can ensure that her donation is eventually processed.

$$\Phi_4 = \langle\langle \{Alice, Bob, Contract\} : wallet(Alice, \geq 20) \wedge wallet(Bob, \geq 20) \rangle\rangle \mathbf{G}\, (ga \rightarrow cc)$$

The coalition of Alice, Bob, and the Contract, provided each of them Alice and Bob have at least 20 tokens, has a strategy to eventually reach a state where goal_achieved *then* campaign_closed *will be true.*

As the reader can easily infer, the formula Φ_1 is false (without the contract's cooperation the goal cannot be achieved), whereas Φ_2, Φ_3, and Φ_4 are true.

To demonstrate the practical use of our framework, the crowdfunding model was encoded in the *VITAMIN* verification environment and analyzed under both the concrete and abstract semantics introduced in Section 6. All four properties were automatically verified using the extended *WATL* engine. The results confirmed the expected truth values of the formulas, validating that the proposed semantics correctly captures wallet-dependent strategic ability. Moreover, this case study provides the foundation for the experimental analysis discussed in the following section, where the proposed verification framework is implemented and evaluated on a wider range of automatically generated *WCGS* models under the *WATL* semantics.

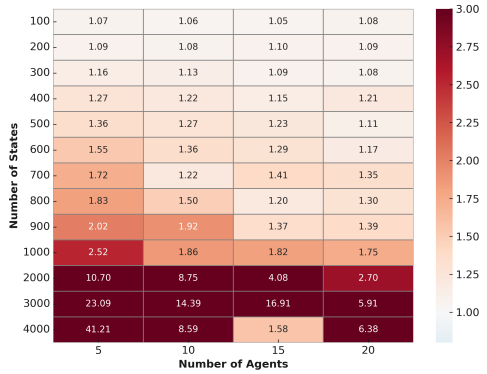


Figure 2: Mean speedup of the abstraction-based verification over the concrete procedure (ConcreteTime/AbstractTime) across different numbers of agents and concrete states. Values greater than 1 indicate configurations where abstraction achieved faster verification.

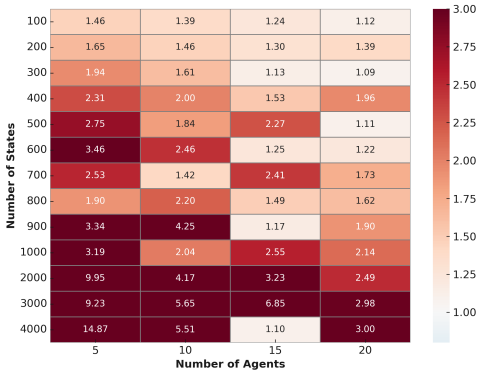


Figure 3: Mean model size reduction achieved by abstraction (ConcreteSize/AbstractSize), combining concrete states, agents, and transitions. Values greater than 1 indicate that the abstract model is smaller than its concrete counterpart.

8 IMPLEMENTATION AND EXPERIMENTS

The proposed approach has been implemented as an extension of the *VITAMIN* model checker. The integration introduces a new module for the verification of *WCGS* under the *WATL* semantics, supporting both concrete and abstract model checking.

Implementation. The framework has been implemented in Python as an extension of the *VITAMIN*[3] model checker [19]. The new module introduces explicit support for *WCGS* and extends the ATL verification engine with *WATL* semantics. The implementation supports both concrete model checking on WCGS (Section 5) and abstraction-based verification (Section 6). All components are fully integrated into VITAMIN's existing infrastructure, ensuring compatibility with its input language, model representation, and fixed-point computation engine.

Experimental setup. We evaluated the framework on randomly generated *WCGS* with varying numbers of agents, states, and wallet bounds, where wallet bounds were sampled uniformly from the interval $[0, 100]$. For each configuration, multiple WATL formulas were automatically generated, involving different coalitions and wallet constraints. Both the concrete and abstraction-based verification procedures were executed on each instance to compare performance and scalability. Experiments considered up to 20 agents; this upper bound was chosen solely to avoid overly cluttered figures and does not reflect any hardware or implementation limitation. All experiments were conducted on a machine equipped with an Intel® Core™ i7–7700HQ CPU @ 2.80,GHz, with four cores, eight threads, and 16,GB of DDR4 RAM.

Results and discussion. Figure 2 reports the mean verification time ratio between concrete and abstract procedures across all configurations. For smaller models (below roughly 500 states), abstraction yields limited benefit, as generation overhead may offset its gains. As model size increases, abstraction progressively outperforms concrete verification. For example, a model with 4000 states and 15 agents required 1.16 s for concrete verification, while its abstract version—containing only 258 states—was verified in 0.33 s. Overall, concrete verification times ranged from 0.78 s to 44.06 s (median 4.69 s), whereas abstract verification ranged from 0.27 s to 25.47 s (median 0.60 s), yielding an approximate eightfold median speedup. These results show that abstraction improves average performance and mitigates worst-case runtime spikes as complexity grows. A small number of outliers for configurations with 15 agents and around 4000 states arise from randomly generated transition structures less amenable to abstraction and do not affect the overall trend. Figure 3 complements these results by comparing the size of concrete and abstract models. The abstract representation consistently reduces states and transitions, with reduction ratios of up to two to three in the largest configurations. This effect strengthens as model size and agent count grow, confirming that abstraction systematically lowers computational cost. The close correlation between reduced model size (Figure 3) and improved runtime (Figure 2) shows that the technique preserves verification correctness while providing a clear scalability benefit.

[3] https://vitamin-organisation.github.io/website/

9 CONCLUSIONS AND FUTURE WORKS

Strategic logics such as ATL and ATL* [4] support multi-agent reasoning but fail to capture financial feasibility, a key aspect of blockchain systems where action executability depends on wallet balances. We introduce *Wallet Alternating-time Temporal Logic (WATL)*, which extends ATL with wallet predicates and wallet-constrained strategic operators. Built on *Wallet-Concurrent Game Structures (WCGS)*, WATL ensures that strategies are both strategically valid and financially executable. We present a PTIME-complete model checking algorithm implemented in *VITAMIN* [19], enabling automated verification of liquidity properties in smart contracts. To address scalability, we propose a *Meta-Agent Abstraction* that aggregates all non-coalition agents into a single meta-agent with summed wallet resources, preserving wallet-aware properties while alleviating state-space explosion. Future work includes extending WATL to WATL* to allow arbitrary temporal nesting, as well as investigating quantitative, probabilistic, and imperfect-information extensions to model token values and uncertainty in DeFi protocols.

REFERENCES

[1] T. Ågotnes, V. Goranko, and W. Jamroga. 2007. Alternating-time temporal logics with irrevocable strategies. In *Proceedings of TARK'07*. 15–24. https://doi.org/10.1145/1324249.1324256

[2] Natasha Alechina, Brian Logan, Nguyen H. Nga, and Aftab Rakib. 2010. Resource-Bounded Alternating-Time Temporal Logic. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*. IFAAMAS.

[3] Natasha Alechina, Brian Logan, Hoang Nga Nguyen, and Franco Raimondi. 2014. Decidable Model-Checking for a Resource Logic with Production of Resources. In *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014) (Frontiers in Artificial Intelligence and Applications, Vol. 263)*, Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan (Eds.). IOS Press, 9–14. https://doi.org/10.3233/978-1-61499-419-0-9

[4] R. Alur, T. A. Henzinger, and O. Kupferman. 2002. Alternating-time temporal logic. *J. ACM* 49, 5 (2002). https://doi.org/10.1145/585265.585270

[5] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. 2017. A Survey of Attacks on Ethereum Smart Contracts SoK. In *Proceedings of the 6th International Conference on Principles of Security and Trust (POST 2017) (Lecture Notes in Computer Science, Vol. 10204)*. Springer, 164–186. https://doi.org/10.1007/978-3-662-54455-6_8

[6] Nay Thiha Aung and Thang Tonthat. 2021. Formal Verification of Smart Contracts Based on User-Defined Rules in a Game-Theoretic Model. *IEEE Access* 9 (2021), 117255–117267. https://doi.org/10.1109/ACCESS.2021.3106711

[7] Guy Avni, Shibashis Guha, and Orna Kupferman. 2017. An Abstraction-Refinement Methodology for Reasoning about Network Games. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI'17)*. https://doi.org/10.24963/ijcai.2017/11

[8] Guy Avni, Shibashis Guha, and Orna Kupferman. 2017. An abstraction-refinement methodology for reasoning about network games. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*. 70–76.

[9] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of Model Checking*. MIT Press, Cambridge, MA.

[10] Thomas Ball and Orna Kupferman. 2006. An abstraction-refinement framework for multi-agent systems. In *Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, 379–388.

[11] Thomas Ball and Orna Kupferman. 2006. An Abstraction-Refinement Framework for Multi-Agent Systems. In *Proceedings of LICS'06 (Logic in Computer Science)*. IEEE, 379–388.

[12] Massimo Bartoletti, Angelo Ferrando, Enrico Lipparini, and Vadim Malvone. 2024. Solvent: Liquidity Verification of Smart Contracts. In *Integrated Formal Methods: 19th International Conference, IFM 2024, Manchester, UK, November 13–15, 2024, Proceedings (Lecture Notes in Computer Science)*. Springer, 256–266. https://doi.org/10.1007/978-3-031-76554-4_14

[13] S. Basu, P. S. Roop, and R. Sinha. 2007. Local Module Checking for CTL Specifications. *Electronic Notes in Theoretical Computer Science* 176, 2 (2007), 125–141. https://doi.org/10.1016/j.entcs.2006.02.035

[14] Vitalik Buterin. 2013. A Next-Generation Smart Contract and Decentralized Application Platform. https://github.com/ethereum/wiki/wiki/White-Paper. Accessed: April 4, 2018.

[15] Davide Catta, Angelo Ferrando, and Vadim Malvone. 2024. Resource Action-Based Bounded ATL: A New Logic for MAS to Express a Cost Over the Actions. In *PRIMA 2024: Principles and Practice of Multi-Agent Systems - 25th International Conference, Kyoto, Japan, November 18-24, 2024, Proceedings (Lecture Notes in Computer Science, Vol. 15395)*, Ryuta Arisaka, Víctor Sánchez-Anguix, Sebastian Stein, Reyhan Aydogan, Leon van der Torre, and Takayuki Ito (Eds.). Springer, 206–223. https://doi.org/10.1007/978-3-031-77367-9_16

[16] Taolue Chen, Jan Křetínský, Alessio Lomuscio, and James Nyman. 2013. Verification of Uniform Strategies in Probabilistic Multi-Agent Systems. *Journal of Artificial Intelligence Research* 46 (2013), 367–437. https://doi.org/10.1613/jair.3771

[17] Aldar Dika and Mariusz Nowostawski. 2018. Security Properties for Game-Based Models of Smart Contracts. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 1109–1116. https://doi.org/10.1109/Cybermatics_2018.2018.00213

[18] Cezar Dima and Cristian Enea. 2009. Applying ATL to Requirement Specification. In *Proceedings of the Asia-Pacific Software Engineering Conference (APSEC 2009)*. IEEE, 123–130.

[19] Angelo Ferrando and Vadim Malvone. 2025. VITAMIN: A Compositional Framework for Model Checking of Multi-Agent Systems. In *Proceedings of the 17th International Conference on Agents and Artificial Intelligence, ICAART 2025 - Volume 1, Porto, Portugal, February 23-25, 2025*, Ana Paula Rocha, Luc Steels, and H. Jaap van den Herik (Eds.). SCITEPRESS, 648–655. https://doi.org/10.5220/0013349600003890

[20] Xiaoxiao Huang, Chen Luo, and Jian Mei. 2017. Bounded Model Checking of ATL under Imperfect Information. *Science of Computer Programming* 135 (2017), 1–20. https://doi.org/10.1016/j.scico.2016.11.002

[21] Wojciech Jamroga. 2015. *Logical Methods for Specification and Verification of Multi-Agent Systems*. Institute of Computer Science, Polish Academy of Sciences, Warsaw.

[22] W. Jamroga and A. Murano. 2014. On module checking and strategies. In *Proceedings of AAMAS'14*. IFAAMAS/ACM, 701–708.

[23] W. Jamroga and A. Murano. 2015. Module Checking of Strategic Ability. In *Proceedings of AAMAS'15*. ACM, 227–235.

[24] Uri Kirstein. [n.d.]. Formal Verification helps finding insolvency bugs — Balancer V2 Bug Report. https://medium.com/certora/formal-verification-helps-finding-insolvency-bugs-balancer-v2-bug-report-1f53ee7dd4d0. Accessed on August 30, 2024.

[25] Jonas Schiffl and Bernhard Beckert. 2024. A Practical Notion of Liveness in Smart Contract Applications. In *Formal Methods in Blockchain (FMBC) (OASIcs, Vol. 118)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 8:1–8:13. https://doi.org/10.4230/OASICS.FMBC.2024.8

[26] Nick Szabo. 1996. Smart Contracts: Building Blocks for Digital Markets. http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html. Accessed: 2025-09-21.

[27] Michael Wooldridge, Wiebe van der Hoek, and Eric Walther. 2005. Model Checking Logics of Strategic Ability: Complexity. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*. ACM. https://doi.org/10.1145/1082473.1082559

[28] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. 2017. An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends. In *2017 IEEE International Congress on Big Data (BigData Congress)*. IEEE, 557–564. https://doi.org/10.1109/BigDataCongress.2017.85