BitML2MCMAS: Strategic Reasoning for Bitcoin Smart Contracts

Demonstration Track

Luigi Bellomarini
Bank of Italy
Rome, Italy
luigi.bellomarini@bancaditalia.it

Marco Favorito
Bank of Italy
Rome, Italy
marco.favorito@bancaditalia.it

Giuseppe Galano
Bank of Italy
Rome, Italy
giuseppe.galano2@bancaditalia.it

ABSTRACT

We present BitML2MCMAS, a formal verification tool for analyzing Bitcoin smart contracts, when specified in BitML, through Atl model checking using the MCMAS model checker. We developed a translation procedure from a BitML contract to an MCMAS model that simulates the BitML semantics, allowing for strategic reasoning on BitML smart contracts. We tested our tool over several case studies, showing that we can verify smart contract specifications that capture interesting multi-agent strategic interactions.

KEYWORDS

Bitcoin Smart Contracts; Strategic reasoning; Atl model checking

ACM Reference Format:
Luigi Bellomarini, Marco Favorito, and Giuseppe Galano. 2025. BitML2MCMAS: Strategic Reasoning for Bitcoin Smart Contracts: Demonstration Track. In *Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Detroit, Michigan, USA, May 19 – 23, 2025*, IFAAMAS, 3 pages.

1 INTRODUCTION

The increasing popularity of Distributed Ledger Technologies (DLTs), such as Bitcoin [20] and Ethereum [13], and the adoption of smart contracts [25], highlighted the need for formal verification methods that check whether smart contracts behave as expected [17, 26], both at implementation level and design level. For example, with *model checking* [15], we can automatically explore the state space of a model of the smart contract to check all possible states and transitions against the desired *specifications* of its functioning. The most used specification languages are *Computational Tree Logic* (Ctl) [14] and *Linear-time Temporal Logic* (Ltl) [22, 23]. However, the underlying assumption of most tools for smart contracts is that, at any time, any participant can do any permissible action. While this assumption makes sense when we aim to detect violations of the specifications across the entire state space, it provides little insight into the *strategic abilities* of participants, e.g., which outcome can be enforced or prevented by a single user or coalition [9].

On the other hand, Atl (*Alternating-time Temporal Logic*) [5] is a temporal logic that can model game properties between multi-agent systems [24, 27], and for which the model checking problem is decidable. In particular, it supports path quantifiers $\langle\langle A \rangle\rangle \varphi$, meaning that coalition A has a strategy to enforce φ to hold on all paths. We can use this expressive power to check relevant strategic properties of smart contracts, e.g., whether a participant/coalition can withdraw a certain amount of funds regardless of the behavior of the other participants or whether a participant/coalition can prevent others' course of action. In a recent work [11], we proposed a technique to perform Atl model checking on Bitcoin smart contracts when they are specified in the BitML domain-specific language [10]. The approach works by constructing a *turn-based asynchronous concurrent game structure (CGS)* [5] that can simulate the semantics of the BitML contract. By imposing certain restrictions on the input contract and by suitably abstracting the semantics, we can keep the state-space of the game finite and so maintain the decidability of the model-checking problem.

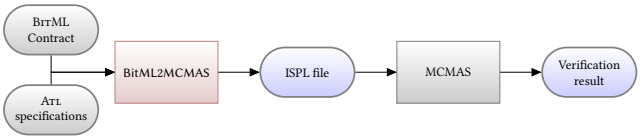

Figure 1: Tool workflow.

In this paper, we present BitML2MCMAS, a prototype tool for performing Atl model checking of a BitML smart contract. The software takes in input a BitML contract, specified in the BitML DSL [8], and a set of Atl specifications to verify, and outputs a program in the *Interpreted System Programming Language* (ISPL) [1], that can simulate any behavior of the participants interacting with the BitML contract. The translation rules are described in [12], which are designed for the ISPL language instead of the CGS model formalism as in [11]. Then, the generated program can be given in input to MCMAS [18], an efficient model checker for multi-agent systems, to verify the target Atl specifications. A high-level diagram of the just described tool workflow is shown in Figure 1. This is the URL to the demonstration video: https://drive.google.com/file/d/14cWDwtoJT9xkLvd6GoGL6Ie8koqPHI44.

Related works. The BitML toolchain [8] allows the verification of various forms of *liquidity* properties and Ltl specifications, with the possibility to fix participant strategies beforehand. Nam and Kil [21] propose a technique to translate an Ethereum smart contract written in a subset of the Solidity language into ISPL and then use MCMAS to verify certain Atl specifications for two-player games. Andrychowicz et al. [7] use *Timed Automata* to model Bitcoin smart contracts, which are specified in the Uppaal model checker [16] that can verify specifications in *Timed Computation Tree Logic*. In all cases, either they do not support strategic reasoning as in Atl model checking, or the focus is not on Bitcoin contracts.

2 BITML IN A NUTSHELL

BitML [10] is a domain-specific language and calculus for Bitcoin smart contracts, which allows a set of *participants* to exchange

Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Y. Vorobeychik, S. Das, A. Nowé (eds.), May 19 – 23, 2025, Detroit, Michigan, USA. © 2025 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

```
(participant "A" "ce6b...ef66") ; A's public key
(participant "B" "e0bb...6eb1") ; B's public key

(contract
  (pre
    (deposit "A" 1 "f0b0...3e17@0") ; tx output (1BTC)
    (secret  "A" a "b39e...b745")   ; hash of A's secret
    (deposit "B" 1 "5346...a079@2") ; tx output (1BTC)
    (secret  "B" b "c0c3...f752"))  ; hash of B's secret
  (choice
    (reveal (a) (choice
                  (reveal (b) (split
                                (1 -> (withdraw "A"))
                                (1 -> (withdraw "B"))))
                  (after 10 (withdraw "A"))))
    (after 5 (withdraw "B"))))
```

Figure 2: Specification of the *mutual timed commitment* contract in the DSL of BitML.

cryptocurrency according to pre-agreed contract rules. A BitML contract specification is made of two components: the *preconditions*, describing requirements that participants must fulfill to stipulate the contract and the actual *contract* that specifies the rules to transfer bitcoins (₿). For example, consider the BitML formalization of a *mutual-timed commitment* contract [6], shown in Figure 2. Intuitively, the contract works as follows: we have two participants, `A` and `B`, each one choosing a secret and depositing 1₿ of cryptocurrency. The goal of the contract is to ensure that each participant will either learn the other participant's secret or receive the other participant's deposit as compensation. The contract gives the participants some time to reveal their secrets. If a participant reveals his secret in time, then he can get her deposit back; otherwise, after the timeout expires, the other participant can withdraw his penalty. The `pre` section defines the contract preconditions (e.g., deposits that participants hold and secrets they must commit to). The `contract` section specifies the contract logic: in the example, the first `choice` operator defines two mutually exclusive alternative branches: one in which `A` reveals his secret `a`, and the other where the first timeout expires, in which case `B` can `withdraw` all the funds (his ₿ plus `A`'s penalty); the second `choice` is analogous to the first one but with the roles exchanged. In case also secret `b` is revealed by `B`, the contract funds are split into equal parts, and both participants can recover their original funds. A full specification of the BitML syntax and semantics can be found in [10].

3 TRANSLATION IN ISPL AND VERIFICATION

BitML2MCMAS takes in input a BitML specification (as the one in Figure 2) and a set of ATL specifications and produces the ISPL model of the BitML contract, according to the approach presented in [12]. Due to lack of space, we cannot fully detail each translation rule, but in a nutshell, the translation is structured as follows: *(a)* in the ISPL file, the behavior of the BitML contract is managed by the Environment agent, while each contract participant `Pi` is associated with agent Agent_Pi of the ISPL program; *(b)* at each step, the environment nondeterministically chooses (i.e., "schedules") an agent whose action is the only one that takes effect to determine the next state. This turn-based asynchronous pattern is used to simulate the concurrency of the BitML calculus rules; *(c)* the global time is incremented by the environment, but only if all agents agree, so to prevent unfair executions *(d)* The BitML semantics (deposits, secret commitments, contract execution, etc.) are encoded in ISPL by means of agents' state variables, protocols, and evolution rules. *(e)* evaluation rules defining the propositions of the ATL specifications, e.g. `part_A_total_deposits_is_at_least_1`, `timeout_1_expired`, `private_secret_a_is_invalid`, etc. Other custom evaluations must be explicitly provided.

Once the ISPL file has been produced, we can use MCMAS to verify the target ATL specifications. The ATL semantics to use must include *(i)* the handling of *fairness constraints* for ensuring fair scheduling of participants, and *(ii)* the handling of imperfect information to treat the secret commitment and revealment mechanism. This means that, when calling MCMAS, we must provide the option `--atlk 2`. In case we can assume perfect information, we can provide the option `--atlk 1`, often resulting in faster verification.

4 IMPLEMENTATION

The prototype tool BitML2MCMAS is written in Python and published open-source on GitHub [19]. The Python library provides the following features: *(a)* parser for the Lisp-like DSL for BitML contracts using the Lark parsing toolkit [2]; *(b)* programmatic definition of BitML contracts by following an object-oriented paradigm; *(c)* creation and manipulation of ISPL programs, with functionality to export them into ISPL program files compatible with MCMAS; this module could be of independent interest to MCMAS users and to the MAS community; *(d)* the BitML-to-ISPL translation rules of the previous section.

Figure 1 shows the workflow of usage of our tool. First, we can use the API of the Python library BitML2MCMAS to generate an ISPL file that models the BitML contract as a multi-agent system, plus the ATL specification to verify. Then, the generated ISPL file can be given as input to the MCMAS model checker. We inherit several additional functionalities by using MCMAS, including: *(a)* simulation of the BitML contract via the ISPL model; *(b)* generation of counterexamples/witnesses; *(c)* visualization of the model.

5 DEMONSTRATION

In the demonstration video, we walk through a tutorial [4] where we showcase the features of the tool and provide an overview of the verification process for interesting ATL specifications on the mutual timed commitment contract, all of which hold true:

- If participant `A` publishes an invalid commit, then he cannot recover his funds:

```
AG((private_secret_a_is_invalid) ->
    !<Agent_A> F( part_A_total_deposits_at_least_1))
```

- If the first timeout has expired and commitment for secret `a` is invalid, then `B` can recover his funds plus `A`'s penalty without revealing its secret `b`:

```
AG((!public_secret_b_is_valid and timeout_1_expired
    and private_secret_a_is_invalid)
  -> <Agent_B>F(part_B_total_deposits_is_at_least_2
                and !public_secret_b_is_valid))
```

- If participant `B` commits to his secret and the first timeout has not expired yet, then it is guaranteed that `B` always has a strategy to take at least 1₿:

```
AG((contract_is_initialized and !timeout_1_expired
    and private_secret_b_is_valid)
  -> <Agent_B>F(part_B_total_deposits_is_at_least_1))
```

We also prepared a tutorial [3] with other case studies implemented in our tool (e.g., an escrow contract and a zero-coupon bond).

REFERENCES

[1] 2015. MCMAS v1.2.2: User Manual.
[2] 2017. lark-parser/lark. https://github.com/lark-parser/lark
[3] 2025. More contracts - BitML2MCMAS. https://marcofavorito.github.io/bitml2mcmas/tutorial/
[4] 2025. Tutorial: Mutual Timed Commitment Contract - BitML2MCMAS. https://marcofavorito.github.io/bitml2mcmas/mutual-timed-commitment-tutorial/
[5] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. 2002. Alternating-time temporal logic. *J. ACM* 49, 5 (2002), 672–713.
[6] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. 2014. Fair Two-Party Computations via Bitcoin Deposits. In *Financial Cryptography Workshops (Lecture Notes in Computer Science, Vol. 8438)*. Springer, 105–121.
[7] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. 2014. Modeling Bitcoin Contracts by Timed Automata. In *FORMATS (LNCS, Vol. 8711)*. Springer, 7–22.
[8] Nicola Atzei, Massimo Bartoletti, Stefano Lande, Nobuko Yoshida, and Roberto Zunino. 2019. Developing secure bitcoin contracts with BitML. In *ESEC/SIGSOFT FSE*. ACM, 1124–1128.
[9] Massimo Bartoletti. 2020. Smart Contracts Contracts. *Frontiers Blockchain* 3 (2020), 27.
[10] Massimo Bartoletti and Roberto Zunino. 2018. BitML: A Calculus for Bitcoin Smart Contracts. In *CCS*. ACM, 83–100.
[11] Luigi Bellomarini, Marco Favorito, and Giuseppe Galano. 2024. Strategic Reasoning for BitML Smart Contracts. In *3rd Workshop on Strategies, Prediction, Interaction, and Reasoning in Italy (SPIRIT)*.
[12] Luigi Bellomarini, Marco Favorito, and Giuseppe Galano. 2025. Strategic Reasoning of BitML Smart Contracts using the MCMAS Model Checker (to appear). In *6th Workshop on Blockchain theoRy and ApplicatIoNs*. https://bit.ly/3WboLMy
[13] Vitalik Buterin. 2014. Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform. (2014).
[14] Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. 1986. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. *ACM Trans. Program. Lang. Syst.* 8, 2 (1986), 244–263.
[15] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. 2001. *Model checking*. MIT Press.
[16] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. 1997. UPPAAL in a Nutshell. *Int. J. Softw. Tools Technol. Transf.* 1, 1-2 (1997), 134–152.
[17] Jing Liu and Zhen-Tian Liu. 2019. A Survey on Security Verification of Blockchain Smart Contracts. *IEEE Access* 7 (2019), 77894–77904.
[18] Alessio Lomuscio, Hongyang Qu, and Franco Raimondi. 2017. MCMAS: an open-source model checker for the verification of multi-agent systems. *Int. J. Softw. Tools Technol. Transf.* 19, 1 (2017), 9–30.
[19] marcofavorito. 2025. BitML2MCMAS. https://github.com/marcofavorito/bitml2mcmas.
[20] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).
[21] Wonhong Nam and Hyunyoung Kil. 2022. Formal Verification of Blockchain Smart Contracts via ATL Model Checking. *IEEE Access* 10 (2022), 8151–8162.
[22] Amir Pnueli. 1977. The Temporal Logic of Programs. In *FOCS*. IEEE Computer Society, 46–57.
[23] Amir Pnueli. 1981. The Temporal Semantics of Concurrent Programs. *Theor. Comput. Sci.* 13 (1981), 45–60.
[24] Yoav Shoham and Kevin Leyton-Brown. 2009. *Multiagent Systems - Algorithmic, Game-Theoretic, and Logical Foundations*.
[25] Nick Szabo. 1997. Formalizing and Securing Relationships on Public Networks. *First Monday* 2, 9 (1997).
[26] Palina Tolmach, Yi Li, Shangwei Lin, Yang Liu, and Zengxiang Li. 2022. A Survey of Smart Contract Formal Specification and Verification. *ACM Comput. Surv.* 54, 7 (2022), 148:1–148:38.
[27] Michael J. Wooldridge. 2009. *An Introduction to MultiAgent Systems*.