Coalition Logic for Specification and Verification of Smart Contract Upgrades

Rustam Galimullin$^{(\boxtimes)1[0000-0003-4195-8189]}$ and Thomas Ågotnes[1,2]

[1] University of Bergen, Bergen, Norway
[2] Southwest University, Chongqing, China
`{rustam.galimullin, thomas.agotnes}@uib.no`

Abstract. It has been argued in the literature that logics for reasoning about strategic abilities, and in particular coalition logic (CL), are well-suited for verification of properties of smart contracts on a blockchain. Smart contracts, however, can be upgraded by providing a new version of a contract on a new block. In this paper, we extend one of the recent formalisms for reasoning about updating CL models with a temporal modality connecting a newer version of a model to the previous one. In such a way, we make a step towards verification of properties of smart contracts with upgrades. We also discuss some properties of the resulting logic and the complexity of its model checking problem.

Keywords: Coalition Logic · Smart Contracts · Blockchain · Model Checking.

1 Introduction

Smart contracts. Smart contracts (SCs) [3, Chapter 7] are programs, usually written in a high-level programming language like Solidity [1], that are stored on a blockchain [17] and executed by nodes of a given distributed ledger. Probably the most well-known blockchain that supports storing and execution of SCs is Ethereum [20]. Since SCs are stored on a blockchain, once deployed, they are immutable. This, however, does not necessarily mean that SCs cannot be amended, in the case of critical bugs, or upgraded. There are several ways to upgrade SCs, including using a separate contract as a proxy, and splitting one contract into several smaller ones. In this paper, we do not discriminate between different ways of upgrading SCs, and focus on upgrades themselves.

Verification and specification of smart contracts. Logic-based verification of blockchain systems and SCs is a nascent field (see a recent survey [19]). For example, authors of [11] use a classic runs-and-system approach to analyse the notion of consensus on a blockchain from an epistemic perspective. A logic for reasoning about blockchain updates was presented in [4], where the authors follow the lead of *dynamic epistemic logic* (DEL) [6]. In [14] a variant of temporal logic is used to focus on properties of a blockchain in a permissionless setting.

It is argued in [15, 16] that logics for reasoning about abilities of agents, in particular *coalition logic* (CL) [18] and ATL [2], have great potential for verification of properties of SCs. However, none of the proposed logical approaches allows one to reason about upgrades of SCs.

One conceptual difficulty we have to overcome is that in the context of blockchains we can have several iterations of an SC recorded on several blocks. New iterations may come about as updates of a contract triggered by new financial policies or amendments to the existing features. This highlights the fact that SCs on a blockchain allow for *introspection* [12]: it is possible to reason about current properties of a contract based on its earlier versions.

To the best of our knowledge, the only extension of CL or ATL that allows for updating agents' abilities is a recently introduced *dictatorial dynamic coalition logic* (DDCL) [7]. DDCL follows the paradigm which is pretty much ubiquitous in DEL: *model* + *update* = *updated model*. As its name suggests, DDCL can express updating dictatorial powers of single agents either by granting such powers or revoking them.

Contribution of the paper. In Section 2.1 we present an extension of DDCL with temporal 'yesterday' relations between a model and the corresponding updated model. Moreover, we consider chain models that are a natural extension of the 'model → updated model' approach. Introduced temporal relations refer to a previous version of a contract, thus adding introspection to the setting. After that, in Section 2.2, we tackle the model checking problem of the introduced logic by showing that it is P-complete. We conclude in Section 3.

2 Reasoning About Abilities of Agents on Chain Models

In this section we first introduce *temporal DDCL* (TDDCL), which is an extension of DDCL [7], and then study the complexity of its model checking problem.

2.1 Syntax and Semantics

Let P be a countable set of *propositional variables*, and A be a finite set of *agents*. Subsets C of A are called *coalitions*.

Definition 1. *The language $\mathcal{TDDCL}$ is given by the following BNF:*

$$\begin{aligned} \mathcal{TDDCL} \ni \varphi &::= p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid \Diamond\varphi \mid \langle\!\langle C \rangle\!\rangle\varphi \mid [+U]\varphi \mid [-U]\varphi \\ +U &::= (\varphi, a, \varphi)^+ \mid (\varphi, a, \varphi)^+, +U \\ -U &::= (\varphi, a, \varphi)^- \mid (\varphi, a, \varphi)^-, -U \end{aligned}$$

where $p \in P$, $a \in A$, $C \subseteq A$. The duals are defined as $[\![C]\!]\varphi := \neg\langle\!\langle C \rangle\!\rangle\neg\varphi$, $\langle +U\rangle\varphi := \neg[+U]\neg\varphi$, $\langle -U\rangle\varphi := \neg[-U]\neg\varphi$, and $\Box\varphi := \neg\Diamond\neg\varphi$.

Formulas of the form $\langle\!\langle C \rangle\!\rangle\varphi$ are read as 'coalition C can force φ', and expressions $\Diamond\varphi$ mean that 'φ is the case in the previous version of a contract'. Constructs $+U$ are called *positive updates*, and formulas $[+U]\varphi$ are read as 'after (positive) update $+U$, φ is true'. Constructs $-U$ are called *negative updates*, and formulas $[-U]\varphi$ are read as 'after (negative) update $-U$, φ is true'[3].

[3] We will sometime use U to denote both positive and negative updates.

The fragment of $\mathcal{TDDCL}$ with only positive updates is $\mathcal{TDDCL}^+$; the fragment with only negative updates is $\mathcal{TDDCL}^-$. The *language of positive DDCL* $\mathcal{DDCL}^+$ is obtained from $\mathcal{TDDCL}$ by omitting $[-U]\varphi$ and $\Diamond\varphi$, and the *language of negative DDCL* $\mathcal{DDCL}^-$ is obtained from $\mathcal{TDDCL}$ by omitting $[+U]\varphi$ and $\Diamond\varphi$. Finally, the fragment of $\mathcal{TDDCL}$ that excludes updates and $\Diamond\varphi$ is called *coalition logic* $\mathcal{CL}$ [18].

Formulas of $\mathcal{TDDCL}$ are interpreted on *chain models* that, in turn, are sequences of *concurrent game models*.

Definition 2. *A* concurrent game model *(CGM), or a* model*, is a tuple* $M = (S, Act, act, out, L)$ *consisting of the following elements:*

- S *is a non-empty set of states, and* Act *is a non-empty set of actions.*
- *Function* $act : A \times S \to 2^{Act} \setminus \emptyset$ *assigns to each agent and each state a non-empty set of actions. A* C*-action at a state* $s \in S$ *is a tuple* α_C *such that* $\alpha_C(i) \in act(i, s)$ *for all* $i \in C$*. The set of all* C*-actions in* s *is denoted by* $act(C, s)$*. We will also write* $\alpha_{C_1} \cup \alpha_{C_2}$ *to denote a* $C_1 \cup C_2$*-action with* $C_1 \cap C_2 = \emptyset$*.*
  A tuple of actions $\alpha = \langle \alpha_1, \dots, \alpha_k \rangle$ *with* $k = \mid A \mid$ *is called an* action profile*. An action profile is* executable *in state* s *if for all* $i \in A$*,* $\alpha_i \in act(i, s)$*. The set of all action profiles executable in* s *is denoted by* $act(s)$*. An action profile* α extends *a* C*-action* α_C*, written* $\alpha_C \sqsubseteq \alpha$*, if for all* $i \in C$*,* $\alpha(i) = \alpha_C(i)$*.*
- *Function* out *assigns to each state* s *and each* $\alpha \in act(s)$ *a unique output state.*
- $L : S \to 2^P$ *is the valuation function.*

We will denote a CGM M *with a designated, or current, state* s *as* (M, s)*.Also, we will use superscript* M *to refer to elements of the corresponding tuple.*

Example 1. Consider model M_1 in Figure 1 on page 5. In the vein of [15, 16] we can think of CGMs as abstract descriptions of contracts specifying what participating parties can and cannot achieve. For the sake of the example, let agents a, b, and c be directors of a company. State s then corresponds to agents discussing whether they should issue stock options. The only way for stock options to be issued is by a and b agreeing to do so (choosing actions a_1 and b_1). All other agents' decisions lead to them staying in the discussion phase. Also observe that agent c does not influence the decision. If a and b agree on issuing stock options (action profile $a_1b_1c_0$), the system transitions to state t meaning that the options has been issued. From this state, agents can either return to the discussion phase (action profile $a_1b_1c_0$) or continue issuing options (all other action profiles).

In the model, we have that no agent can force p on their own, written as $(\mathcal{C}_1, M_1, s) \models [\![a]\!]\neg p \wedge [\![b]\!]\neg p \wedge [\![c]\!]\neg p$. At the same time, a coalition consisting of agents a and b can force a state satisfying p (by choosing actions a_1 and b_1) since agent c has no action to preclude such a transition: $(\mathcal{C}_1, M_1, s) \models \langle\!\langle \{a, b\} \rangle\!\rangle p$.

Updates allow us to modify a CGM by adding or removing action profiles. Given a positive update $+U$ and a CGM M, we denote *updated* CGM as M^{+U}. Similarly, we write M^{-U} in the case of a negative update $-U$.

Unfortunately, due to the lack of space, we cannot present formal definitions for updated CGMs, and instead resort to intuitive explanations and examples. Since the particularities of such updates are not the focus of this work, we believe that this omission does not hinder the readability of the paper. For definitions and details the interested reader is referred to [7, 9].

To provide the intuition behind updates, we first define forcing actions, i.e. actions that allow a particular single agent to force a specific outcome no matter which actions other agents choose. Formally, the set of *forcing actions* for agent i and state s, denoted as $\mathfrak{f}(i, s)$, is $\{\alpha_i \in act(i, s) \mid \forall \alpha, \beta \in act(s) : (\alpha_i \sqsubseteq \alpha \text{ and } \alpha_i \sqsubseteq \beta) \text{ implies } out(\alpha, s) = out(\beta, s)\}$.

Now, the intuition behind $[+U]\varphi$ is as follows. Each $(\varphi, a, \psi)^+$ specifies between which states agent a will be granted a *new* forcing action. In case if multiple states satisfy φ or ψ, agent a will have a new action for each pair of states. Negative updates, on the other hand, specify between which states forcing actions should be *preserved.* In other words, if agent a has a forcing action from a state satisfying φ to a state satisfying ψ, then they will retain the action if triple $(\varphi, a, \psi)^-$ appears in $-U$, and lose it otherwise.

Example 2. Consider once again CGM M_1 depicted in Figure 1. Assume that the initial version of the contract has led to a and b abusing their power and issuing stock options (staying in state t) without ever discussing it (not transitioning to state s). To mitigate this, a new policy has been issued: agent c should decide whether the company continues issuing stock options or enters a discussion state. Moreover, the new policy also requires that agent a's decision is enough to issue stock. Such a policy can be described by update $+U^1 = \{(\neg p, a, p)^+, (p, c, \neg p)^+\}$. The result of updating the existing contract with this new policy is model M_2. In M_2 we indeed have that in all $\neg p$-states (state s) agent a has a new forcing action a_2 to force p-states (state t). Similarly, agent c now has a new forcing action c_1 to force state s from state t. We thus have that $(\mathcal{C}_1, M_1, t) \models \langle\!\langle \{a, b\} \rangle\!\rangle p \wedge [+U^1][\![\{a, b\}]\!]\neg p$ meaning that in state t of CGM M_1 a coaltion of agents a and b can force a p state, but after update $+U^1$ they lose such a power.

As a result of the update, model M_2 has the following non-empty sets of forcing actions: $\mathfrak{f}(a, s) = \{a_2\}$ and $\mathfrak{f}(c, t) = \{c_1\}$. Continuing with the company setting, assume that the next policy was to revoke the ability of agent c to force the discussion phase (state s) from the state of issuing stocks (state t). Such a policy can be represented by a negative update $-U^2 = \{(\neg p, a, p)^-\}$ where triple $(\neg p, a, p)^-$ prescribes to preserve a's power to force a p-state from a $\neg p$-state. Since there are no more triples in $-U^2$, we remove all other forcing actions and corresponding action profiles. The result of updating M_2 with $-U^2$ is M_3, where there is only forcing action left is $\mathfrak{f}(a, s) = \{a_2\}$. Observe that action profiles containing c_1 were removed from the model. Formally, it holds that $(\mathcal{C}_1, M_2, t) \models \langle\!\langle c \rangle\!\rangle \neg p \wedge [-U^2][\![c]\!]p$ meaning that in state t of CGM M_2, agent c can force $\neg p$, but after negative update $-U^2$ she loses such an ability.

Definition 3. *A minimal chain model $\mathcal{C}$ is a pair $(\mathcal{M}, \leftsquigarrow)$, where $\mathcal{M}$ is a non-empty set of CGMs, and $\leftsquigarrow$ is a possibly empty relation between $S^M \times S^N$,* *where M and N are CGMs such that $M, N \in \mathcal{M}$. A minimal chain model with a designated state s of CGM M is denoted by $(\mathcal{C}, M, s)$.*

Definition 4. *Let $\mathcal{C} = (\mathcal{M}, \leftsquigarrow)$ be a minimal chain model, $M \in \mathcal{M}$ be a CGM, and U be an update. The result of executing U in CGM M on $\mathcal{C}$ is $\mathcal{C}^U = (\mathcal{M}^U, \leftsquigarrow^U)$, which we call* updated minimal chain model*, where $\mathcal{M}^U = \mathcal{M} \cup \{M^U\}$, and $\leftsquigarrow^U = \leftsquigarrow \cup \{(s,s) \mid \pi_1((s,s)) \in S^M$ and $\pi_2((s,s)) \in S^{M^U}\}$* [4].

Updating a model M in a minimal chain model $\mathcal{C}$ adds updated CGM M^U to $\mathcal{C}$ and connects each state of M^U with a corresponding state of M with temporal relation $\leftsquigarrow$ (represented by dashed arrows in Figure 1).

Example 3. The previously considered examples of positive and negative updates in Figure 1 can be viewed now as a single chain structure. CGM M_1 can be

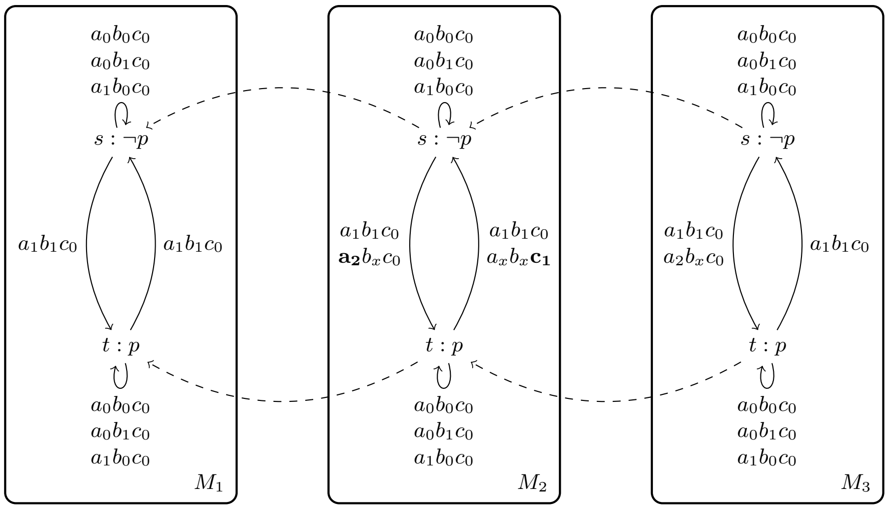

Fig. 1. Chain model $\mathcal{C}_1$ consisting of CGMs M_1, M_2, and M_3. Dashed arrows represent temporal relations, and bold actions depict new action profiles required by an update. Symbol x stands for elements of $\{0, 1\}$.

considered as an initial version of an SC. After incorporating new policy $+U^1$, a new version of the contract, M_2, is added to the initial one. The two contracts are now also connected by a temporal relation such that M_1 is reachable from M_2. Adding a third version M_3 as a result of updating M_2 with $-U^2$ makes the chain longer So, having the latest version of a contract, M_3, we can express, for example, that now agent a can on her own force the company to issue stocks (transition to state t) while it was not possible two iterations of the contract ago. Formally, $(\mathcal{C}_1, M_3, s) \models \langle\langle a\rangle\rangle p \wedge \Diamond\Diamond\neg\langle\langle a\rangle\rangle p$, where $\mathcal{C}_1$ refers to the whole chain, and M_3 refers to the third version of the contract.

[4] π_1 and π_2 are left and right projections of an ordered pair.

Upgrading smart contracts in the blockchain setting preserves history i.e. updating a chain model does not alter the already existing CGMs and their order. Thus, we can reason introspectively about the *evolution* of a smart contract.

Definition 5. *Let $(\mathcal{C}, M, s)$ be a pointed minimal chain model. The semantics of TDDCL are defined recursively as follows, where Boolean cases are omitted:*

$$\begin{array}{ll}
(\mathcal{C}, M, s) \models \Diamond\varphi & \textit{iff } \exists N \in \mathcal{M}, t \in S^N : t \leftsquigarrow s \textit{ and } (\mathcal{C}, N, t) \models \varphi \\
(\mathcal{C}, M, s) \models \langle\langle C \rangle\rangle\varphi & \textit{iff } \exists \alpha_C, \forall \alpha_{\overline{C}} : (\mathcal{C}, M, t) \models \varphi, \textit{ where } t = out(s, \alpha_C \cup \alpha_{\overline{C}}) \\
(\mathcal{C}, M, s) \models [+U]\varphi & \textit{iff } +U \textit{ is executable in } M \textit{ implies } (\mathcal{C}^{+U}, M^{+U}, s) \models \varphi \\
(\mathcal{C}, M, s) \models [-U]\varphi & \textit{iff } -U \textit{ is executable in } M \textit{ implies } (\mathcal{C}^{-U}, M^{-U}, s) \models \varphi
\end{array}$$

We call a formula φ valid *if for all $(\mathcal{C}, M, s)$ it holds that $(\mathcal{C}, M, s) \models \varphi$.*

The *executability* condition in the above definition stems from the fact that not all updates can be executed. More on this in [7].

Remark 1. Note that CGMs do *not* correspond to specific blocks on a given blockchain. They are rather abstract specifications of different versions of contracts. Thus, our approach encompasses *software versioning* in general. We will stick, however, with SCs as the prime source of our intuitions.

The class of minimal chain models is too broad for purposes at hand. For example, it allows minimal chain models to have CGMs with several previoius CGMs. This goes against the intuition of having upgrades of an SC on a blockchain since for each block there is at most one previous block. Below we define a subclass of minimal chain models that satisfy our intuitions about blockchains.

Definition 6. *Let $\mathcal{C} = (\mathcal{M}, \leftsquigarrow)$ be a minimal chain model, and $M \in \mathcal{M}$ be a CGM. We call M* the initial CGM *if for all $s \in S^M$, relation $\leftsquigarrow$ is empty.*

Let $\ldots \leftsquigarrow t_1 \leftsquigarrow \ldots \leftsquigarrow t_n \leftsquigarrow s$ be a sequence of $\leftsquigarrow$-relations of maximal length ending in s. We call the length of the sequence depth *of s, and write $d(s)$. Note that $d(s)$ can be infinite. All states in the initial CGM have depth 0.*

Definition 7. *Minimal chain model $\mathcal{C} = (\mathcal{M}, \leftsquigarrow)$ is called a* chain model *if it satisfies the following properties.*

1. The initial CGM*: there is a single $M \in \mathcal{M}$ such that M is the initial CGM.*
2. CGM-definedness*: for all states s, t, and u, if $t \leftsquigarrow s$ and $u \leftsquigarrow s$, then $t = u$. In other words, for each state in a CGM there is only one previous state.*
3. Depth-definedness*: $d(s) \neq \infty$ for all s.*
4. Valuation preservation*: for all states s and t belonging to some M and N, if $s \leftsquigarrow t$, then $s \in L^M(p)$ if and only if $t \in L^N(p)$. The property reflects that updates do not change valuations of propositional variables.*
5. Update-definedness*: for all $s \in S^M$, $t \in S^N$, if $s \leftsquigarrow t$, then there is a U s.t. $M^U = N$. Each new CGM is a result of updating the previous one.*

Depth-definedness implies *acyclicity* and *irreflexivity* of $\leftsquigarrow$. On the other hand, depth-definedness does not imply finiteness. Indeed, our chain models are, in fact, tree-like, and thus they allow a possibly infinite number of children at a given node. Moreover, *intransitivity* of $\leftrightsquigarrow$ follows from irreflexivity and CGM-definedness. Also note that both chain and minimal chain models do not allow agents to force transitions between different blocks (the *synchronicity* property).

Theorem 1. *Let $\mathcal{C} = (\mathcal{M}, \leftrightsquigarrow)$ be a chain model, and let U be an update. The result $\mathcal{C}^U$ of updating $\mathcal{C}$ with U is a chain model.*

Proposition 1. *The following formulas are valid on chain models.*
1. $\Diamond(\varphi \land \psi) \leftrightarrow \Diamond\varphi \land \Diamond\psi$ *2.* $\neg\Diamond\varphi \leftrightarrow \Diamond\neg\varphi$ *3.* $\varphi \to [U]\Diamond\varphi$

The first formula expresses the property that there is only one previous CGM. The second property states that $\Diamond$ is its own dual. Observe that the first two items are not valid on minimal chain models. Finally, the third validity shows that adding a new CGM does not change the existing CGMs.

2.2 Model checking

The model checking problem for DDCL$^+$ and DDCL$^-$ is investigated in [9], where it is shown to be P-complete for both logics. The provided algorithms label each state of a given model by a subformula of a given formula. The labelling is inspired by the classic model checking algorithm for CTL [5]. There are two separate algorithms, one for DDCL$^+$ and one for DDCL$^-$, because it is not yet clear how we can combine positive and negative updates in such a way that the resulting complexity is still in P. Thus, we will present extentions of the algorithms from [9] so that they work with formulas of $\mathcal{TDDCL}^+$ and $\mathcal{TDDCL}^-$.

Given a formula φ we organise the list of its subformulas in such a way that formulas within updates are evaluated before formulas that are in the scope of these updates. It allows us to know the effect of an update before we have to evaluate formulas that are affected by the update. As an example, consider formula $\varphi := [(q, b, \neg q)^+]\Diamond p$ with $+U^1 := (q, b, \neg q)^+$. The ordered labelled list $sub(\varphi)$ looks as follows: q, $\neg q$, $(p)^{+U^1}$, $(\Diamond p)^{+U^1}$, $[(q, b, \neg q)^+]\Diamond p$.

Algorithm 1 An algorithm for global TDDCL$^+$ model checking

```
1: procedure GlobalTDDCL+($\mathcal{C}, M, \varphi$)
2:     for all $\psi^\sigma \in sub(\varphi)$ do
3:         for all $M \in \mathcal{M}$ do
4:             for all $s \in S^M$ do
5:                 case $\psi^\sigma = (\Diamond\chi)^\sigma$
6:                     if there is a $t$ such that $t \leftrightsquigarrow s$ and $t$ is labelled with $\chi^\sigma$ then
7:                         label $s$ with $(\Diamond\chi)^\sigma$
8: end procedure
```

Algorithm 1 is a modification of the corresponding algorithm for DDCL$^+$. The main idea behind it is that while checking $\psi^\sigma = (\langle\langle C\rangle\rangle\chi)^\sigma$, we need to 'model' the effects of positive updates $+U$. To do this we check for each update in σ, starting from the last one, whether it affected agents from C.

Compared to the algorithm for DDCL$^+$, GlobalTDDCL$^+$ has an additional loop over CGMs in the given blockchain model $\mathcal{C}$ (line 3), and an additional case for $(\Diamond\chi)^\sigma$ (lines 5–7). All other case are the same as in [9] and we omit them. It is clear Algorithm 1 follows the semantics, and thus its correctness can be shown by the induction on φ. Moreover, the case for $(\Diamond\chi)^\sigma$ takes polynomial time.

Theorem 2. *Complexity of TDDCL$^+$ model checking problem is P-complete.*

Model checking TDDCL$^-$ is similar to model checking TDDCL$^+$. It also requires preparation of list $sub(\varphi)$ with the only difference that now after we are done with labelling subformulas within update $-U$, we include $-U$ in the list right after the subformulas. For example, having a formula $\varphi := [(q, b, \neg q)^-]\Diamond p$ with $-U^1 := (q, b, \neg q)^+$, the ordered labelled list $sub(\varphi)$ looks as follows: q, $\neg q$, $-U^1$, $(p)^{+U^1}$, $(\Diamond p)^{+U^1}$, $[(q, b, \neg q)^+]\Diamond p$.

The model checking algorithm for TDDCL$^-$ is a modification of the one for DDCL$^-$ exactly in the same way as in Algorithm 1.To model negative updates, the DDCL$^-$ algorithm marks action profiles with a sequence of negative updates meaning that the corresponding action profile has been preserved after these updates. Similarly to GlobalTDDCL$^+$, case $(\Diamond\chi)^\sigma$ takes polynomial time.

Theorem 3. *Complexity of TDDCL$^-$ model checking problem is P-complete.*

3 Conclusion

We presented TDDCL, a logic that allows us to reason about SC upgrades in the blockchain setting: we considered its properties and the model checking problem. Since this is a first step towards specification of SCs on a blockchain, there is a plethora of open problems.

First, our model checking algorithm works with fragments of $\mathcal{TDDCL}$, either with the positive or negative version. So far, it is not clear how one can combine both types of updates while maintaining polynomial time complexity. In the future, we would like to provide a general algorithm and implement it as an extension of one the tools for strategic logics (e.g. MCMAS [13], MCK [10]).

Updates of CGMs in our case are restricted to granting and revoking dictatorial powers of single agents. Finding other, more general, ways of updating SCs is an exciting avenue of further research. An approach we find most tempting is incorporating chain structures to a recent dynamic CL with action models [8]. Similar goals can be set out for base languages that are more expressive than CL, like ATL and ATL*.

References

1. Solidity programming language. https://soliditylang.org, accessed: 20-01-2022

2. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. Journal of the ACM **49**, 672–713 (2002). https://doi.org/10.1145/585265.585270
3. Antonopoulos, A.M., Wood, G.: Mastering Ethereum. O'Reilly (2018)
4. Brünnler, K., Flumini, D., Studer, T.: A logic of blockchain updates. Journal of Logic and Computation **30**(8), 1469–1485 (2020). https://doi.org/10.1093/logcom/exaa045
5. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching-time temporal logic. In: Kozen, D. (ed.) Logics of Programs. LNCS, vol. 131, pp. 52–71. Springer (1981). https://doi.org/10.1007/BFb0025774
6. van Ditmarsch, H., van der Hoek, W., Kooi, B.: Dynamic Epistemic Logic, Synthese Library, vol. 337. Springer (2008)
7. Galimullin, R., Ågotnes, T.: Dynamic coalition logic: Granting and revoking dictatorial powers. In: Ghosh, S., Icard, T. (eds.) Proceedings of the 8th LORI. LNCS, vol. 13039, pp. 88–101. Springer (2021). https://doi.org/10.1007/978-3-030-88708-7_7
8. Galimullin, R., Ågotnes, T.: Action models for coalition logic. In: Proceedings of the 4th DaLí. (to appear) (2022)
9. Galimullin, R., Ågotnes, T.: Dictatorial dynamic coalition logic. Manuscript. Submitted to a journal (2022), https://rgalimullin.gitlab.io/DDCL/ddcldraft.pdf
10. Gammie, P., van der Meyden, R.: MCK: model checking the logic of knowledge. In: Alur, R., Peled, D.A. (eds.) Proceedings of the 16th CAV. LNCS, vol. 3114, pp. 479–483. Springer (2004). https://doi.org/10.1007/978-3-540-27813-9_41
11. Halpern, J.Y., Pass, R.: A knowledge-based analysis of the blockchain protocol. In: Lang, J. (ed.) Proceedings of the 16th TARK. EPTCS, vol. 251, pp. 324–335 (2017). https://doi.org/10.4204/EPTCS.251.22
12. Herlihy, M., Moir, M.: Blockchains and the logic of accountability: Keynote address. In: Grohe, M., Koskinen, E., Shankar, N. (eds.) Proceedings of the 31st LICS. pp. 27–30. ACM (2016). https://doi.org/10.1145/2933575.2934579
13. Lomuscio, A., Qu, H., Raimondi, F.: MCMAS: an open-source model checker for the verification of multi-agent systems. International Journal on Software Tools for Technology Transfer **19**(1), 9–30 (2017). https://doi.org/10.1007/s10009-015-0378-x
14. Marinkovic, B., Glavan, P., Ognjanovic, Z., Studer, T.: A temporal epistemic logic with a non-rigid set of agents for analyzing the blockchain protocol. Journal of Logic and Computation **29**(5), 803–830 (2019). https://doi.org/10.1093/logcom/exz007
15. van der Meyden, R.: On the specification and verification of atomic swap smart contracts. CoRR **abs/1811.06099** (2018), http://arxiv.org/abs/1811.06099
16. van der Meyden, R.: On the specification and verification of atomic swap smart contracts (extended abstract). In: Proceedings of the 1st ICBC. pp. 176–179. IEEE (2019). https://doi.org/10.1109/BLOC.2019.8751250
17. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf (2008)
18. Pauly, M.: A modal logic for coalitional power in games. Journal of Logic and Computation **12**(1), 149–166 (2002). https://doi.org/10.1093/logcom/12.1.149
19. Tolmach, P., Li, Y., Lin, S., Liu, Y., Li, Z.: A survey of smart contract formal specification and verification. ACM Computing Surveys **54**(7), 148:1–148:38 (2022). https://doi.org/10.1145/3464421
20. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum Project (2014), https://ethereum.github.io/yellowpaper/paper.pdf