



# Blockchain-Based Business Processes: A Solidity-to-CPN Formal Verification Approach

Ikram Garfatta<sup>1,2(✉)</sup>, Kaïs Klai<sup>2</sup>, Mahamed Graïet<sup>3,4</sup>, and Walid Gaaloul<sup>5</sup>

<sup>1</sup> University of Tunis El Manar, National Engineering School of Tunis, OASIS,  
Tunis, Tunisia

<sup>2</sup> University Sorbonne Paris North, LIPN UMR CNRS 7030, Villetaneuse, France  
[ikram.garfatta@lipn.univ-paris13.fr](mailto:ikram.garfatta@lipn.univ-paris13.fr)

<sup>3</sup> Higher Institute for Computer Science and Mathematics, University of Monastir,  
Monastir, Tunisia

<sup>4</sup> National School for Statistics and Information Analysis, Rennes, France

<sup>5</sup> Institut Mines-Télécom, Télécom SudParis, UMR 5157, SAMOVAR, Évry, France

**Abstract.** With its span of applications widening by the day, the technology of Blockchain has been gaining more interest in different domains. It has intrigued many investors, but also numerous malicious users who have put different Blockchain platforms under attack. It is therefore an inescapable necessity to guarantee the correctness of smart contracts as they are the core of Blockchain applications. Existing verification approaches, however, focus on targeting particular vulnerabilities, seldom supporting the verification of domain-specific properties.

In this paper, we propose a translation of Solidity smart contracts into CPNs (Coloured Petri nets) and investigate the capability of *CPN Tools* to verify CTL (Computation Tree Logic) properties.

**Keywords:** Blockchain · Formal verification · Smart contract · Solidity · Coloured Petri nets · CTL properties

## 1 Introduction

Within the span of the two last decades, many advances have been made in the world of Blockchain, allowing this technology to expand its reach to a myriad of application domains including Business Process Management (BPM) [13]. A Blockchain platform can indeed provide a reliable execution of business processes (BPs) even within a trustless network, especially thanks to the concept of smart

---

Supervised by Kaïs Klai, University Sorbonne Paris North, LIPN UMR CNRS 7030, Villetaneuse, France, and Mahamed Graïet Higher Institute for Computer Science and Mathematics, University of Monastir, Monastir, Tunisia and National School for Statistics and Information Analysis, Rennes, France.

Co-directed by Walid Gaaloul, Institut Mines-Télécom, Télécom SudParis, UMR 5157, SAMOVAR, Paris, France.

© Springer Nature Switzerland AG 2021

H. Hacid et al. (Eds.): ICSOC 2020 Workshops, LNCS 12632, pp. 47–53, 2021.

[https://doi.org/10.1007/978-3-030-76352-7\\_7](https://doi.org/10.1007/978-3-030-76352-7_7)

contracts. In a BPM context, a smart contract can define business collaborations in general and inter-organizational BPs in particular. In fact, smart contracts are pieces of script code that act like autonomous software agents, used to enforce management rules on the execution of transactions on the Blockchain. They are stored on and executed by the Blockchain and therefore inherit its characteristics, particularly its immutability. This same feature can, however, turn into a weak spot for such contracts. In fact, as a smart contract cannot be altered once it has been deployed on the Blockchain, it cannot be corrected either, which makes verifying its correctness prior to its deployment an indispensable necessity. Furthermore, the correctness verification is an important aspect for the design of blockchain-based BPs. The assessment of such processes involves both requirements validation and consistency.

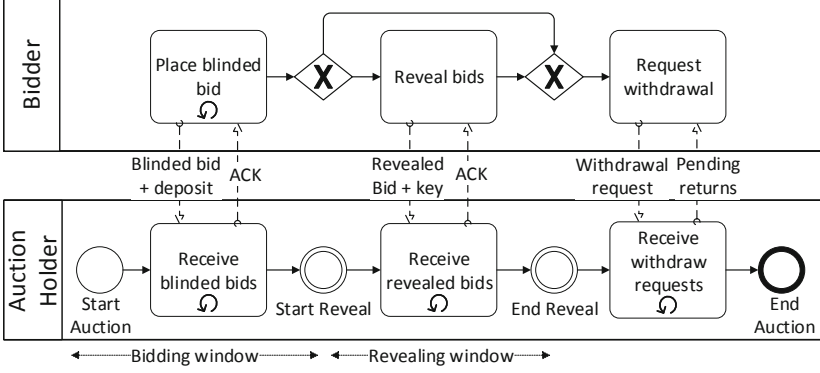
The main long-term objective of this thesis is therefore to develop an approach that allows to construct correct blockchain-based BPs. In this paper, we present our progress for the first milestone towards this goal, which we define as the verification of smart contracts in a general context. We are interested in Ethereum smart contracts as it is currently the second largest cryptocurrency platform after Bitcoin besides being the inaugurator of smart contracts, and particularly those written in Solidity [1] as it is the most popular language used by Ethereum. The contribution described herein is a first step towards a formal verification approach based on CPNs [7] for Solidity smart contracts.

Existing studies on the formal verification of smart contracts follow two main streams. The first group of studies are based on theorem proving [2–4]. In this case, the verification is not automated and requires the user’s expertise in the manipulation of the used theorem prover as well as manual intervention in discharging proofs. The second group of studies are based on symbolic model checking coupled with complementary techniques such as symbolic execution and abstraction [5, 8, 11, 12, 14]. In order to use symbolic execution to generate the traces that would be used for the verification, the proposed approaches usually use under-approximation (e.g., in the form of loop bounds) which means that critical violations can be overlooked. This explains the presence of false negatives and/or positives in their reported results. We also note that most of the existing studies target specific vulnerabilities in smart contracts, and few are those that allow expressing customizable properties, in which case they are control flow-related properties. In fact, none of these studies target data-related properties. It is worth mentioning that most of the proposed approaches operate on the EVM bytecode rather than on the Solidity code because of the latter’s lack of formal semantics. This, however, results in loss of contextual information, and consequently limits the range of properties that can be verified on the contract.

To overcome these shortcomings, we propose an algorithm for the translation of a Solidity smart contract into a hierarchical CPN model over which CTL properties can be verified. This work can easily be integrated as an extension layer into existing studies that rely on the translation of BP models into smart contracts as in [10] which generates Solidity code from models written in BPMN (Business Process Model and Notation), to verify their output and therefore check the correctness of the initial BP models.

## 2 Solidity-to-CPN Translation

To illustrate our approach and prove its feasibility, we adapt the Blind Auction example included in [1]. Figure 1 describes the workflow of the blind auction system. For a full description of the use case, we refer the reader to the *Solidity2CPN* document available at this repository<sup>1</sup>.



**Fig. 1.** Blind auction workflow

The general idea of our approach is to start from a CPN model representing the general workflow of the smart contract (level-0 model) and then to build on it by embedding it with submodels representing the smart contract functions (level-1 models). In a level-0 model, we distinguish the *user's behaviour* part which models the way users can interact with the system and the *smart contract's behaviour* part which represents the system. These two are linked via *communication places*. Figure 2 shows the level-0 model of the previously described blind auction use case.

We see a smart contract as a set of statements. A statement can be either a compound, a simple or a control one. A simple statement can be an assignment, a variable declaration, a sending or a returning statement. A control statement can be a requirement, a selection or a loop (a *for* or a *while* loop).

### 2.1 Translation Algorithm

Our proposed algorithms are structured as follows:

- EXTENDMODEL takes as input the level-0 CPN model and builds the extended hierarchical model by calling INSERTSUBMODEL for each transition corresponding to a function in the Solidity smart contract.

<sup>1</sup> <https://github.com/garfatta/solidity2cpn>.

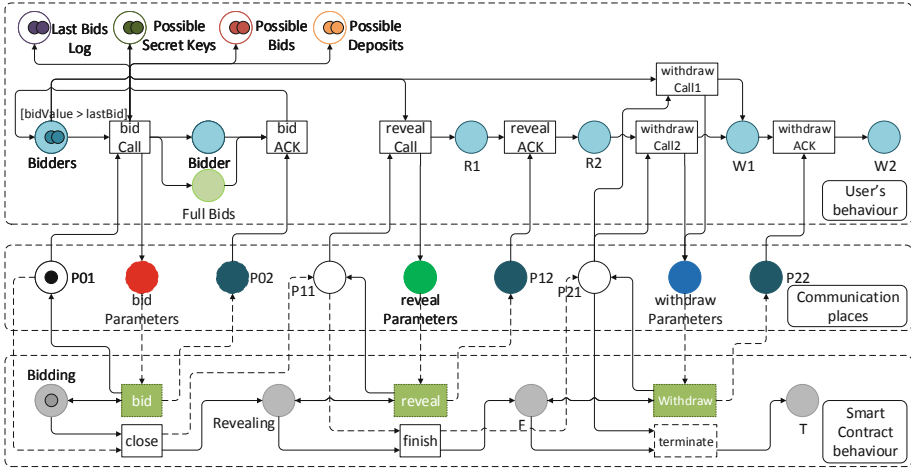


Fig. 2. Blind auction - level-0 model

- INSERTSUBMODEL is responsible for replacing a transition by its corresponding level-1 submodel and connecting it to the level-0 model.
- CREATESUBMODEL is the main algorithm. It generates the level-1 submodel for each transition by browsing the body of its corresponding function recursively and creates CPN patterns according to the type of the processed statement that interconnect to create the transition's submodel.

## 2.2 Application on the Blind Auction Use Case

The application of the algorithm on the level-0 model of the Blind Auction use case (see Fig. 2) yields a hierarchical CPN model whose level-1 submodels are created by the execution of CREATESUBMODEL. Figure 3 shows the submodel corresponding to transition *withdraw* in the level-0 model. The rest of the submodels can be found in the online Solidity2CPN document (See footnote 1).

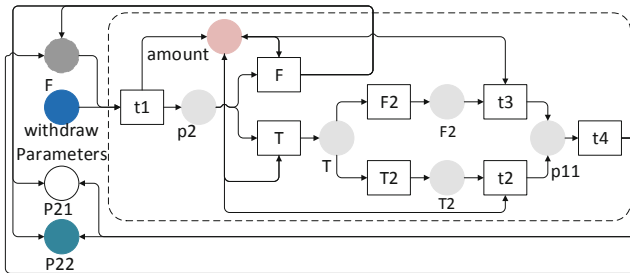


Fig. 3. SubModel of transition *withdraw*

### 3 Smart Contract Verification via CPN Tools

Having established the CPN model for a smart contract, verifying properties of the smart contract would come down to verifying properties on the CPN model. We have implemented the CPN model for our Blind Auction use case using *CPN Tools* which leverages explicit model checking techniques, and investigated its potential in the verification of behavioural and contract-specific properties.

In Table 1 we present state space analysis statistics for different initial marking values. We note that the unprovided values mean that the state space generation had not finished after several hours of execution. This is due to the infamous state space explosion problem associated with explicit state space exploration.

**Table 1.** State Space Analysis Results for different initial markings

Bidders		1	2	3	4	5	1	1	1
Possible bids		1	1	1	1	1	2	3	4
Possible secret keys		1	1	1	1	1	2	3	4
Possible deposits		1	1	1	1	1	2	3	4
State space generation time	Without hierarchy	4 s	4 s	6 s	252 s	–	4 s	30 s	–
	With hierarchy	5 s	5 s	10 s	1001 s	–	5 s	74 s	–
#Nodes	Without hierarchy	24	235	3118	47621	–	484	19984	–
	With hierarchy	44	583	9166	156117	–	1424	65513	–
#Arcs	Without hierarchy	26	378	7106	145062	–	555	22980	–
	With hierarchy	46	900	19784	446326	–	1545	70704	–
#Dead Markings	Without hierarchy	3	10	35	124	–	49	1999	–
	With hierarchy	3	10	35	124	–	49	1999	–

The state space report generated by *CPN Tools* allows the deduction of several general behavioural properties. For instance, in our use case application, the report confirms the boundedness of all the places of the modelled system. More specific properties can be verified by elaborating CTL properties. For instance, we can formulate a *termination* property to check the model’s capability to always reach a terminal state (a dead marking) where certain conditions are met. We include the definition of such a property in the Solidity2CPN document<sup>6</sup>.

### 4 Conclusion

The goal of our work is to propose a formal approach for the verification of smart contracts. In this context, we propose in this paper a translation algorithm that generates a hierarchical CPN model representing a given Solidity smart contract, including both its control-flow and data aspects. CTL properties are then verified

on the CPN model to check corresponding properties on the smart contract, unrestrictedly to certain predefined vulnerabilities.

In view of the results presented in this paper, it may be concluded that *CPN Tools* does not hold much potential for the verification of properties on CPN models of smart contracts due to the state space explosion problem. We do prove, however, that the idea of using CPNs as a representation formalism is promising for it allows the consideration of the data aspect, and thus the formulation of contract-specific properties. To overcome the encountered limitations, we intend to investigate the potential of *Helena* [6] as an analyzer for High Level Nets. This tool offers on-the-fly verification of LTL properties, which unlike the verification of CTL properties offered by *CPN Tools*, does not always require the generation of the whole state space. To further improve the tool's performance, we also intend to work on *Helena*'s model checker by embedding it with an extension to an existing technique previously developed to deal with the state space explosion problem in regular Petri nets [9] and applying it on CPNs.

## References

1. Solidity documentation. <https://solidity.readthedocs.io/en/latest/>
2. Formal verification for solidity contracts - ethereum community forum, October 2015. <https://forum.ethereum.org/discussion/3779/formal-verification-for-solidity-contracts>
3. Amani, S., Bégel, M., Bortin, M., Staples, M.: Towards verifying ethereum smart contract bytecode in isabelle/hol. In: Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, pp. 66–77 (2018)
4. Bhargavan, K., et al.: Formal verification of smart contracts: short paper. In: Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security, PLAS@CCS 2016, Vienna, Austria, 24 October 2016, pp. 91–96 (2016)
5. Chen, T., Li, X., Luo, X., Zhang, X.: Under-optimized smart contracts devour your money. In: IEEE 24th International Conference on Software Analysis, Evolution and Reengineering, SANER 2017, Austria, 20–24 February 2017, pp. 442–446 (2017)
6. Evangelista, S.: High level petri nets analysis with helena. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 455–464. Springer, Heidelberg (2005). [https://doi.org/10.1007/11494744\\_26](https://doi.org/10.1007/11494744_26)
7. Jensen, K., Kristensen, L.M.: Coloured Petri Nets: Modelling and Validation of Concurrent Systems, 1st edn. Springer, Heidelberg (2009). <https://doi.org/10.1007/b9511210.1007/b95112>
8. Kalra, S., Goel, S., Dhawan, M., Sharma, S.: ZEUS: analyzing safety of smart contracts. In: 25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, 18–21 February 2018 (2018)
9. Klai, K., Poitrenaud, D.: MC-SOG: an LTL model checker based on symbolic observation graphs. In: 29th International Conference Applications and Theory of Petri Nets, PETRI NETS 2008, China, 23–27 June, Proceedings. pp. 288–306 (2008)
10. López-Pintado, O., García-Bañuelos, L., Dumas, M., Weber, I., Ponomarev, A.: Caterpillar: a business process execution engine on the ethereum blockchain. *Softw. Pract. Exp.* **49**(7), 1162–1193 (2019)

11. Luu, L., Chu, D., Olickel, H., Saxena, P., Hobor, A.: Making smart contracts smarter. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016, p. 254–269 (2016)
12. Mavridou, A., Laszka, A., Stachtari, E., Dubey, A.: Verisolid: correct-by-design smart contracts for ethereum. In: Financial Cryptography and Data Security - 23rd International Conference, St. Kitts and Nevis, 18–22 February 2019, p. 446–465 (2019)
13. Mendling, J., Weber, I.: Blockchains for business process management - challenges and opportunities. *EMISA Forum* **38**(1), 22–23 (2018)
14. Torres, C.F., Schütte, J., State, R.: Osiris: hunting for integer bugs in ethereum smart contracts. In: Proceedings of the 34th Annual Computer Security Applications Conference, ACSAC 2018, PR, USA, 03–07 December 2018, p. 664–676 (2018)