5. Yanhong A. Liu & Scott D. Stoller (2020): *Knowledge of Uncertain Worlds: Programming with Logical Constraints*. In: Proceedings of the International Symposium on Logical Foundations of Computer Science (LFCS 2020), Lecture Notes in Computer Science 11972. Springer, pp. 111–127, doi:10.1007/978-3-030-36755-8_8. Also https://arxiv.org/abs/1910.10346.
6. Yanhong A. Liu & Scott D. Stoller (2020): *Recursive Rules with Aggregation: A Simple Unified Semantics*. Computing Research Repository arXiv:2007.13053 [cs.DB]. http://arxiv.org/abs/2007.13053.
7. Allen Van Gelder (1992): *The Well-Founded Semantics of Aggregation*. In: Proceedings of the 11th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp. 127–138, doi:10.1145/137097.137854.

Modeling Bitcoin Lightning Network by Logic Programming (Extended Abstract)

Damiano Azzolini (*University of Ferrara*)
Elena Bellodi (*University of Ferrara*)
Alessandro Brancaleoni (*University of Ferrara*)
Fabrizio Riguzzi (*University of Ferrara*)
Evelina Lamma (*University of Ferrara*)

Bitcoin is one of the first decentralized, peer to peer, payment systems based on the Proof-of-Work consensus algorithm. The network suffers from a scalability issue due to several limitations such as the restriction imposed on the blocks' size. For this reason, several approaches have been proposed, among which the so-called "Layer 2 solutions", where a layer of channels is built on top of a blockchain. This allows users to send transactions through these channels, without broadcasting them on the blockchain, increasing the number of transactions per second that can be managed. One of the main examples of this last approach is the Lightning Network: in this paper we propose to represent this network and to query its properties by means of Logic Programming, in order to study its evolution over time.

Bitcoin [3] is one of the most famous decentralized payment system based on a *blockchain*. The algorithm used to append block to the blockchain, Proof of Work (POW) for Bitcoin, ensures that blocks, once discovered, cannot be easily tampered with. However, it also represents one of the main bottlenecks since only few transactions per second can be supported in the network. Currently, one of the main approaches to increase Bitcoin capacity is represented by the so-called "Layer 2 solutions". This type of systems creates a layer of channels on top of a blockchain. With these channels, users can overcome some blockchain limitations by issuing off-chain transactions. One of the most famous systems is the *Lightning Network* (LN) [4]. Here, we encode the Lightning Network by means of Logic Programming, since it has been proved effective in several Bitcoin scenarios [1,2]. The use of Prolog both as the representation language of the Lightning Network and as the query language allows us to easily analyse its characteristics requiring a minimal effort in the code implementation.

The Lightning Network can be represented as a graph $G = (V, E)$ where V is the set of vertices (nodes) and E is the set of edges (payment channels). The degree of a node is defined as the number of edges incident to the node. A path of length n between two nodes $v_a, v_b \in V$ is a sequence of payment channels $e_1, \ldots, e_n \in E$ such that $e_1 = (x_1, x_2)$, $e_2 = (x_2, x_3), \ldots, e_n = (x_{n-1}, x_n)$ where $x_1 = v_a$ and $x_n = v_b$. The distance between two nodes is defined as the shortest path that connects those nodes. The capacity of an edge is defined as the maximum payment that can be routed through it. One of the problem of routing in LN is that the capacity of a channel between two nodes is known but the distribution of the capacity in each direction is unknown, since it is a feature introduced to increase the security of the system: this makes routing a complicated task, since several attempts may be required to send a payment.

We represent a channel of capacity `Amount` between two nodes `Source` and `Dest` with a Prolog fact of the form `edge(Source,Dest,Amount)`. The channels in the network are also characterized by other values (i.e., fee base and fee rate), but we ignore them since they are not needed in our experiments. Theoretically, the amount locked in a channel is split between the two connected nodes, so a payment channel should be represented by two directed edges. However, the amount distribution is unknown, so we suppose that nodes are connected by only one undirected edge. The Prolog representation of this situation, between a source node a and a destination node b, is given by a single fact `edge(a,b,8)`. The whole network is a set of ground facts `edge/3`.

Starting from the dataset related to [6], we trace the LN evolution along the months of February, March and April 2020 and how its properties and connections vary over time, through a series of experiments. We analyze the structure of the network in terms of node degree distribution: the majority of the nodes of the network (more than 65% for all three datasets) has degree between 1 and 5. Then, we compute how the total network capacity varies by removing the edges with the top capacity values and the nodes with the highest associated degree. The goal of this experiment is to show how much the capacity of the network is centralized in the hands of few. By removing the edges, the network capacity substantially reduces after 50 removals. Instead, removing the top 100 nodes decreases the total network capacity approximately by 90%. We analyse the *rebalancing* operation (i.e., a node sends a payment to itself) and, as expected, as the node degree increases, the maximum rebalancing amount increases as well. Finally, we compute the number of paths of length 2 and 3 among equal degree nodes, with the degree varying between 1 and 10. We focus on short paths since, in practice, the average length of the shortest path is 2.8 [7]. Moreover, longer paths also imply higher fees to pay. We discover that the number of paths drops after the 3rd or 4th degree for all networks in both cases. As a future work, we plan to extend our analysis using Probabilistic Logic Programming [5].

References

1. Damiano Azzolini, Fabrizio Riguzzi & Evelina Lamma (2019): *Studying Transaction Fees in the Bitcoin Blockchain with Probabilistic Logic Programming*. Information10(11), p. 335, doi:10.3390/info10110335.
2. Damiano Azzolini, Fabrizio Riguzzi, Evelina Lamma, Elena Bellodi & Riccardo Zese (2018): *Modeling Bitcoin Protocols with Probabilistic Logic Programming*. In Elena Bellodi & Tom Schrijvers, editors: Proceedings of the 5th International Workshop on Probabilistic Logic Programming, PLP 2018, co-located with the 28th International Conference on Inductive Logic Programming (ILP 2018), Ferrara, Italy, September 1, 2018,CEUR Workshop Proceedings2219, CEUR-WS.org, pp. 49–61. Available at http://ceur-ws.org/Vol-2219/paper6.pdf.
3. Satoshi Nakamoto (2008): *Bitcoin: A peer-to-peer electronic cash system*.

4. Joseph Poon & Thaddeus Dryja (2016): *The bitcoin lightning network: Scalable off-chain instant payments*.
5. Fabrizio Riguzzi (2018): *Foundations of Probabilistic Logic Programming*. River Publishers, Gistrup, Denmark.
6. Elias Rohrer, Julian Malliaris & Florian Tschorsch (2019): *Discharged Payment Channels: Quantifying theLightning Network's Resilience to Topology-Based Attacks*. In: 2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), IEEE, pp. 347–356, doi:10.1109/EuroSPW.2019.00045.
7. István András Seres, László Gulyás, Dániel A Nagy & Péter Burcsi (2020): *Topological analysis of bitcoin's lightning network*. In: Mathematical Research for Blockchain Economy, Springer, pp. 1–12.

Accountable Protocols in Abductive Logic Programming (Extended Abstract)

Marco Gavanelli (*University of Ferrara*)
Marco Alberti (*University of Ferrara*)
Evelina Lamma (*University of Ferrara*)

Abstract

Finding the responsible of an unpleasant situation is often difficult, especially in artificial agent societies.

*S*CIFF is a language to define formal rules and protocols in agent societies, and an abductive proof-procedure for compliance checking. However, how to identify the responsible for a violation is not always clear.

In this work, a definition of accountability for artificial societies is formalized in *S*CIFF. Two tools are provided for the designer of interaction protocols: a guideline, in terms of syntactic features that ensure accountability of the protocol, and an algorithm (implemented in a software tool) to investigate if, for a given protocol, non-accountability issues could arise.

1 Introduction

The current economic world is strongly based on large corporations, that have been able to provide large economic benefits, such as cheaper prices for everyday goods and better employment rates, but that also represented large problems in case of failure. Every person can list problems in his/her own country in which a large firm misbehaved, e.g., polluting the environment, or by failing in a disastrous way causing huge losses for small investors. In many cases, the firm itself cannot be punished for its misbehavior, because a company cannot be sent to jail. One hopes that the culprit of the misbehavior (e.g., the CEO) is punished, but in many cases the complex behavior of an organization depends on the policies established by previous members (that might even be dead), by common practices, or by the fact that many individuals contributed to the disaster each for an inconceivably small amount.

In the literature of moral responsibility, there exist different notions of responsibility. Van de Poel et al. [3] distinguish five moral meanings of responsibility: accountability, blameworthiness, liability, obligation and virtue.

> The ascription of responsibility-as-accountability has the following implication: i is responsible-as-accountable for ϕ implies that i should account for (the occurrence of) ϕ, in particular for i's role in doing, or bringing about ϕ, or for i's role in failing to prevent ϕ from happening, where i is some agent, and ϕ an action or a state-of-affairs.

and

> Accountability implies blameworthiness unless the accountable agent can show that a reasonable excuse applies that frees her from blameworthiness. So holding an agent i accountable shifts the burden of proof for showing that i is not blameworthy to the agent i: the agent is now to show - by giving an account - that she is not blameworthy.

In this work, we focus on the *S*CIFF system [1], a complete system for defining and checking the compliance of agents to interaction protocols. It includes a language to define agent interaction protocols and to relate a current state of affairs with one or more expected behaviors of the agents, formalized as a set of expectations. The language was designed to leave freedom to agents, not overconstraining them to follow statically pre-defined paths, but, instead, to assert explicitly the obligatory actions and those that are forbidden, while leaving everything not explicitly stated as a possible action that an agent can perform if it is convenient. An abductive proof-procedure accepts asynchronous events and reasons about them through the protocol definition, generates the expected behavior of the agents, and checks if the actual behavior matches with the expectations.

*S*CIFF lacks a concept of responsibility, because expectations, unlike commitments, are not characterized by a debtor, due to different language design objectives: while *S*CIFF is able to detect violations of the protocols, it is not always clear which agent is responsible for the wrong state of affairs.

In this work, we address the problem by adopting the accountability version of responsibility. Accountability in the proposed setting stands for the possibility to account for the wrong state of affairs of an agent that is the one that performed (or did not perform) the action in its expected behavior. The agent might then, by reasoning on the protocol and the state of affairs (again, using the *S*CIFF proof procedure), be able to *account for* its own behavior. The agent might be able to find an explanation in which its expected behavior is fulfilled, and in such a case it cannot be held responsible for the violation. In some cases, this might happen because another agent is actually responsible for the violation, but in other cases it might be due to a wrong design of the interaction protocol.

For this reason, we define formally a notion of *accountability* of the interaction protocol. The idea is that a protocol is accountable if it allows to identify the agent (or agents) responsible for each possible violation. If the interactions in an organization or society are ruled by an accountable protocol, then, for each possible undesirable state of affairs, one or more agents will be unambiguously held responsible.