

Correctness of Broadcast via Multicast: Graphically and Formally

Wolfgang Jeltsch

Well-Typed
London, England
wolfgang@well-typed.com

Javier Díaz

Atix Labs (a Globant Division)
Buenos Aires, Argentina
javier.diaz@globant.com

Maintaining data consistency among multiple parties requires nodes to repeatedly send data to all other nodes. For example, the nodes of a blockchain network have to disseminate the blocks they create across the whole network. The scientific literature typically takes the ideal perspective that such data distribution is performed by broadcasting to all nodes directly, while in practice data is distributed by repeated multicast. Since correctness and security of consistency maintenance protocols usually have been established for the ideal setting only, it is vital to show that these properties carry over to real-world implementations. Therefore, it is desirable to prove that the ideal and the real behavior are equivalent.

In the work described in this paper, we take an important step towards such a proof by proving a simpler variant of this equivalence statement. The simplification is that we consider only a concrete pair of network topologies, which nevertheless illustrates important phenomena encountered with arbitrary topologies. For describing systems that distribute data, we use a domain-specific language of processes that corresponds to a class of Petri nets and is embedded in a general-purpose process calculus. This way, we can outline our proof using an intuitive graphical notation and leverage the rich theory of process calculi in the actual proof, which is machine-checked using the Isabelle proof assistant.

1 Introduction

Systems that maintain data consistency among multiple parties are becoming increasingly relevant. For example, blockchains have seen growing utilization in areas such as finance, identification, logistics, and real estate. Incorrect behavior of such systems may often result in serious damage. It is therefore valuable to formally prove their correctness and security.

Keeping data consistent requires nodes to regularly disseminate data to other nodes. A perspective often taken in the scientific literature is that such data is broadcast to all nodes directly. For example, the descriptions of the blockchain consensus protocols of the Ouroboros family [1, 6, 10] assume such direct communication. In practice, however, data is distributed via repeated multicast. Since correctness and security typically have been established for the ideal setting only, it is vital to show that they carry over to real-world implementations.

In this paper, we take an important step in this direction by showing that the ideal behavior of direct broadcast and the real behavior of broadcast via multicast are in a certain sense equivalent. Concretely, we make the following contributions:

- In Sect. 2, we define a restricted language of processes that are able to describe network communication. Processes in our language correspond to hierarchical Petri nets with exactly one input place per transition. We define our language via an embedding in a general-purpose process calculus. This approach enables us to leverage the rich theory of process calculi while allowing us to use an intuitive graphical notation similar to the one of Petri nets.

- In Sect. 3, we devise a notion of behavioral equivalence of networks that does not distinguish between different patterns of packet arrival. Our approach is to start with bisimilarity and weaken it by amending the involved processes to allow for additional behavior. Building on bisimilarity permits us to reason in a modular fashion.
- In Sect. 4, we present a proof of behavioral equivalence of broadcast via multicast and direct broadcast under the assumption that network communication may involve packet loss and duplication. Our proof is about a concrete pair of networks that nevertheless captures important general phenomena. The proof works by rewriting a process describing the former form of broadcast into a process describing the latter. For the individual rewriting steps, we rely on certain fundamental lemmas, not all of which have been proved so far. We outline our proof using the graphical notation for processes and show the proof’s first part in detail. The whole proof is formalized in Isabelle/HOL.

Afterwards, we discuss related work in Sect. 5 and give a conclusion and an outlook on ongoing and future work in Sects. 6 and 7.

2 A Language for Communication Networks

For describing communication networks, we use a custom language of processes that communicate via asynchronous channels. Let uppercase letters denote processes and lowercase letters denote channels. The syntax of our communication language is given by the following BNF rule:

$$Process ::= \mathbf{0} \mid P \parallel Q \mid \nu a. P \mid a \rightarrow b \mid a \Rightarrow [b_1, \dots, b_n] \mid \mathfrak{A}^? a \mid \mathfrak{A}^+ a \mid \mathfrak{A}^* a$$

A process is one of the following:

- The *stop process* $\mathbf{0}$, which does nothing
- A *parallel composition* $P \parallel Q$, which performs P and Q in parallel
- A *restricted process* $\nu a. P$, which behaves like P except that the channel a is local
- A *bridge* $a \rightarrow b$, which continuously forwards packets from channel a to channel b
- A *distributor* $a \Rightarrow [b_1, \dots, b_n]$, which continuously forwards packets from channel a to all channels b_i
- A *loser* $\mathfrak{A}^? a$, which continuously drops packets from channel a
- A *duplicator* $\mathfrak{A}^+ a$, which continuously duplicates packets in channel a
- A *duploser* $\mathfrak{A}^* a$, which continuously drops packets from and continuously duplicates packets in channel a

Parallel composition has lowest precedence, restriction has intermediate precedence, and all other constructs have highest precedence.

Our communication language is embedded in the \mathfrak{P} -calculus¹ (pronounced “thorn calculus”), a general-purpose process calculus that is itself embedded in Isabelle/HOL. Parallel composition, restriction, and the stop process are in fact constructs of the \mathfrak{P} -calculus, which we directly use in the communication language. Besides these three constructs, the \mathfrak{P} -calculus provides constructs for sending

¹See <https://github.com/input-output-hk/thorn-calculus>.

and receiving: $a \triangleleft x$ is a process that sends value x to channel a , and $a \triangleright x.P$ is a process that receives a value x from channel a and continues like P , where P may mention x . The operational semantics of the \mathcal{P} -calculus is essentially the one of the asynchronous π -calculus [9].

The constructs of our communication language other than those that stem directly from the \mathcal{P} -calculus are defined in terms of \mathcal{P} -calculus constructs as follows:

$$a \Rightarrow [b_1, \dots, b_n] = a \triangleright x. (b_1 \triangleleft x \parallel \dots \parallel b_n \triangleleft x \parallel a \Rightarrow [b_1, \dots, b_n]) \quad (1)$$

$$a \rightarrow b = a \Rightarrow [b] \quad (2)$$

$$\mathfrak{A}^? a = a \Rightarrow [] \quad (3)$$

$$\mathfrak{A}^+ a = a \Rightarrow [a, a] \quad (4)$$

$$\mathfrak{A}^* a = \mathfrak{A}^? a \parallel \mathfrak{A}^+ a \quad (5)$$

Note that distributors with a common source channel compete for the packets in that channel. As a consequence, the presence of a loser or duplicator for some channel does not mean that all packets sent to this channel are lost or are duplicated forever, because other distributors might get hold of packets in this channel and thus prevent the loser or duplicator from fetching them. Furthermore, observe that, while the \mathcal{P} -calculus has the mobility feature of the π -calculus, this feature cannot be exploited by processes of the communication language, since such processes cannot specifically send and receive channels, but only forward data, treating it as black boxes.

The sublanguage formed by $\mathbf{0}$, \parallel , ν , and \Rightarrow corresponds to hierarchical Petri nets with exactly one input place per transition, with the constructs of the two formalisms corresponding to each other as follows:

$$\begin{aligned} \text{channel} &\leftrightarrow \text{place} \\ \text{distributor} &\leftrightarrow \text{transition} \\ \text{stop process} &\leftrightarrow \text{Petri net without transitions} \\ \text{parallel composition} &\leftrightarrow \text{Petri net join that identifies equal places} \\ \text{restricted process} &\leftrightarrow \text{subnet} \end{aligned}$$

This correspondence allows us to use the graphical notation for hierarchical Petri nets to depict processes of that fragment of the communication language. We modify the representation of restricted processes and add custom notation for the remaining constructs, leading to the graphical notation shown in Fig. 1. We call diagrams that use this notation *communication nets*.

Observe that the notation for bridges just reflects their definition in terms of distributors, as shown in (2), and therefore actually does not constitute an extension to Petri net notation. The notation for losers, duplicators, and duplosers is really an extension, but since these constructs are defined in terms of distributors as well, their notation is only syntactic sugar, which we could drop in favor of the explicit representations shown in Fig. 2.

Communication nets identify bisimilar (and thus weakly bisimilar) processes to some degree. This can ease reasoning, as it means that certain transformations of processes into bisimilar ones become no-ops when working on the more abstract level of communication nets. Concretely, the following

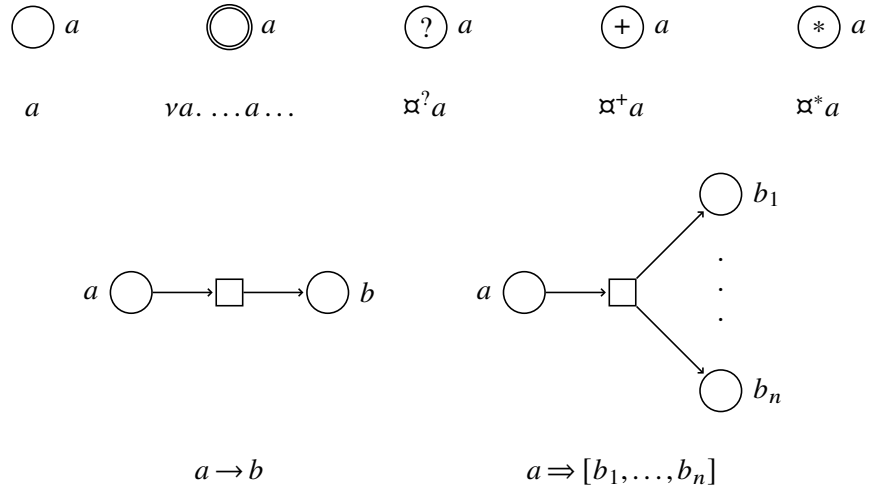


Figure 1: Elements of communication nets

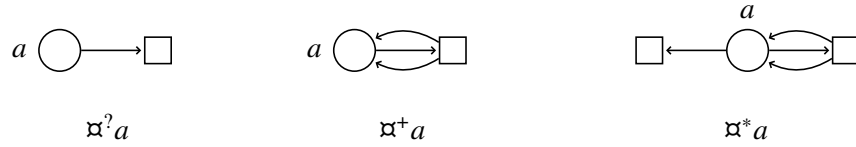


Figure 2: Explicit communication net representations of unreliability constructs

bisimilarities are implicit in the communication net notation:

$$\mathbf{0} \parallel P \sim P \quad (6)$$

$$P \parallel \mathbf{0} \sim P \quad (7)$$

$$(P \parallel Q) \parallel R \sim P \parallel (Q \parallel R) \quad (8)$$

$$P \parallel Q \sim Q \parallel P \quad (9)$$

$$P \parallel \nu a. Q \sim \nu a. (P \parallel Q) \quad \text{if } a \text{ does not occur freely in } P \quad (10)$$

$$\nu a. \nu b. P \sim \nu b. \nu a. P \quad (11)$$

$$\nu a. P \sim P \quad \text{if } a \text{ does not occur freely in } P \quad (12)$$

The reason for (10) to (12) being implicit is that the scopes of local channels are not reflected by communication nets, which just show locality as a property of channels. Actually, (12) is not implicit in general, only when the unused channel a is not depicted, which should be the norm, but is not required.

Two complete communication nets are shown in Fig. 3. They characterize the two networks whose behavioral equivalence we show in Sect. 4. In both communication nets, channels s_i and r_i form the interface of a network node i , with s_i accepting packets for sending and r_i providing received packets. The local channel m in the left communication net represents the broadcast medium, and each local channel l_{ij} in the right communication net represents a multicast link from node i to node j . Note that we assume any communication to be unreliable, which is reflected by the channel m and all channels l_{ij} having duplosers attached. The duplicator part of the duploser attached to m has the additional purpose of allowing packets to reach all nodes instead of only one node each.

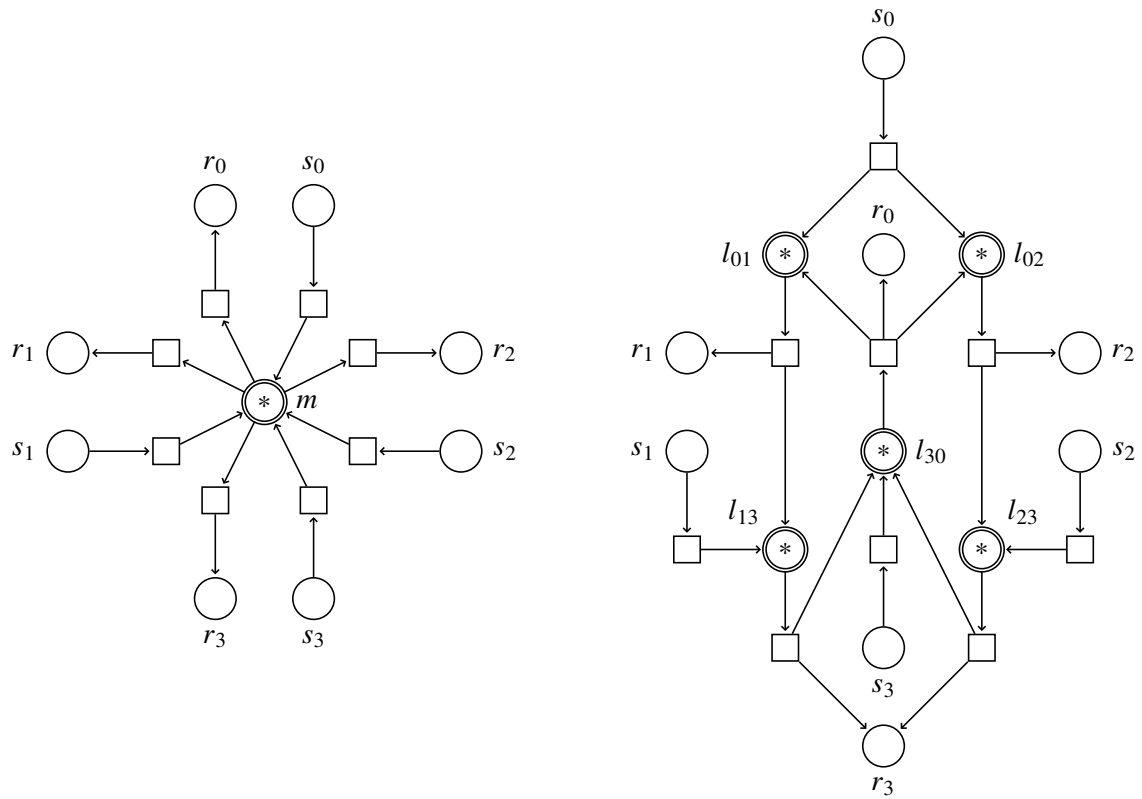


Figure 3: Example of direct broadcast (left) and broadcast via multicast (right)

The two communication nets in Fig. 3 correspond to processes D (direct broadcast) and M (broadcast via multicast) defined as follows:

$$D = \nu m. (\alpha^* m \parallel s_0 \rightarrow m \parallel s_1 \rightarrow m \parallel s_2 \rightarrow m \parallel s_3 \rightarrow m \parallel m \rightarrow r_0 \parallel m \rightarrow r_1 \parallel m \rightarrow r_2 \parallel m \rightarrow r_3) \quad (13)$$

$$M = \nu l_{01}. \nu l_{02}. \nu l_{13}. \nu l_{23}. \nu l_{30}. (M_* \parallel M_i \parallel M_o) \quad (14)$$

$$M_* = \alpha^* l_{01} \parallel \alpha^* l_{02} \parallel \alpha^* l_{13} \parallel \alpha^* l_{23} \parallel \alpha^* l_{30} \quad (15)$$

$$M_i = s_0 \Rightarrow [l_{01}, l_{02}] \parallel s_1 \Rightarrow [l_{13}] \parallel s_2 \Rightarrow [l_{23}] \parallel s_3 \Rightarrow [l_{30}] \quad (16)$$

$$M_o = l_{01} \Rightarrow [r_1, l_{13}] \parallel l_{02} \Rightarrow [r_2, l_{23}] \parallel l_{13} \Rightarrow [r_3, l_{30}] \parallel l_{23} \Rightarrow [r_3, l_{30}] \parallel l_{30} \Rightarrow [r_0, l_{01}, l_{02}] \quad (17)$$

3 Loss-Agnostic Behavioral Equivalence

It is self-suggesting to consider weak bisimilarity as the kind of equivalence that should hold between the two network processes introduced in the previous section. Weak bisimilarity is a well-established notion of behavioral equivalence that provides a fine-grained distinction of observable behavior. Usually, weak bisimilarity is a congruence with respect to most, if not all, process constructors, allowing for modular reasoning. In the case of the communication language, it is a congruence with respect to both parallel composition and restriction.

The notion of behavior behind weak bisimilarity captures possible future actions that are observable. The behavior of a process is essentially a rooted graph where nodes are states, with the root node being the current state, edges are state transitions, which are marked with the observable actions that cause these transitions, and branching characterizes non-determinism. In the case of the \mathcal{P} -calculus and thus the communication language, observable actions are send and receive actions that involve global channels. With the expressivity of our communication language, which is limited compared to that of the \mathcal{P} -calculus, behavior characterizes essentially how handing over packets to global channels may result in packets appearing on possibly other global channels.

Unfortunately, weak bisimilarity turns out to be too strict for our situation, as it is able to distinguish between broadcast via multicast and direct broadcast. To see why, assume in each of the networks shown in Fig. 3 a packet is sent by node 0 and this packet makes it to node 3. With direct broadcast, it is possible that neither node 1 nor node 2 receives the packet as well. With broadcast via multicast, however, node 1 or node 2 must receive it and must do so before node 3 receives it.

This mismatch is an example of a more general issue, which shows up with other sender–receiver pairs and also with other pairs of networks: With direct broadcast, any arrival pattern is possible. With broadcast via multicast, only arrival patterns that correspond in some sense to the network topology are possible; more precisely, when a packet makes it to some node, a path from the sender to the receiver must exist whose intermediate nodes all receive the packet in the order they appear on this path.

To remove this constraint on arrival patterns, we make the receive channels lossy. This way, intermediate nodes are no longer guaranteed to receive packets. It is not sufficient, however, to introduce this lossiness for broadcast via multicast only; we need to introduce it also for direct broadcast. This is because packet loss in the receive channels is observable and consequently unilateral introduction of such loss would create another behavioral mismatch between the two networks.

The approach of making receive channels lossy leads to a notion of *weak bisimilarity up to loss*, which is derived from weak bisimilarity (written \approx in the following):

Definition 1 (Weak bisimilarity up to loss). Two processes P and Q are weakly bisimilar up to loss in channels r_1 to r_n exactly if

$$\alpha^? r_1 \parallel \dots \parallel \alpha^? r_n \parallel P \approx \alpha^? r_1 \parallel \dots \parallel \alpha^? r_n \parallel Q .$$

Note that describing the equivalence of the two broadcast networks using weak bisimilarity up to loss does not result in a trivial statement. In particular, weak bisimilarity up to loss does not just consider the situation where all packets are lost. Instead, it considers also non-trivial arrival patterns, because the losers that are attached to the receive channels compete with the environment for packets, so that packets may escape the grip of the losers.

4 A Proof of Correctness of Broadcast via Multicast

Broadcast via multicast is expected to behave equivalently to direct broadcast as long as the multicast network is strongly connected. In this work, however, we prove this equivalence only for the particular networks depicted in Fig. 3, which nevertheless capture important phenomena that show up in other cases:

- The multicast network has a node with several outgoing and a node with several incoming links.
- In the multicast network, some nodes are reachable from certain other nodes only via more than one hop.

Nothing in our proof is fundamentally tied to these particular networks, though, and generalizing this proof to arbitrary pairs of a broadcast-via-multicast network and a corresponding direct-broadcast network should be straightforward. We merely chose a concrete example for this early work on broadcast network equivalence to make the proof easier to conduct.

Our concrete goal is to prove that M , defined in (14), and D , defined in (13), are weakly bisimilar up to loss in the receive channels, which is expressed by the statement

$$\alpha^?r_0 \parallel \alpha^?r_1 \parallel \alpha^?r_2 \parallel \alpha^?r_3 \parallel M \approx \alpha^?r_0 \parallel \alpha^?r_1 \parallel \alpha^?r_2 \parallel \alpha^?r_3 \parallel D .$$

We prove this statement by turning its left-hand side into its right-hand side through a series of transformation steps, each of which replaces subprocesses with bisimilar ones. These replacements are justified by several basic lemmas about the communication language. Due to time constraints, we have not yet proved all of these lemmas, but we have reasonable confidence that they are correct.

4.1 Correctness of Broadcast via Multicast: Graphically

To outline our proof, we present the individual transformation steps and the key lemmas used by them graphically using communication nets. The lemmas are all bisimilarities, which are depicted in Fig. 4. The transformation steps are the following ones:

1. Untangling of receiving and relaying, which turns the broadcast-via-multicast process depicted in Fig. 3 with the receive channels made lossy into the process depicted in Fig. 5, using the *distributor-split* lemma
2. Transforming the core, which turns the result of the previous step into the process depicted in Fig. 6, using the *bridge-shortcut-redundancy* lemma
3. Collapsing the sending part, which turns the result of the previous step into the process depicted in Fig. 7, using the *distributor-target-fusion* lemma
4. Collapsing the receiving part, which turns the result of the previous step into the process depicted in Fig. 8, using the *bridge-source-switch* lemma
5. Collapsing the core, which turns the result of the previous step into the direct-broadcast process depicted in Fig. 3 with the receive channels made lossy, using the *duploss-detour-collapse* lemma

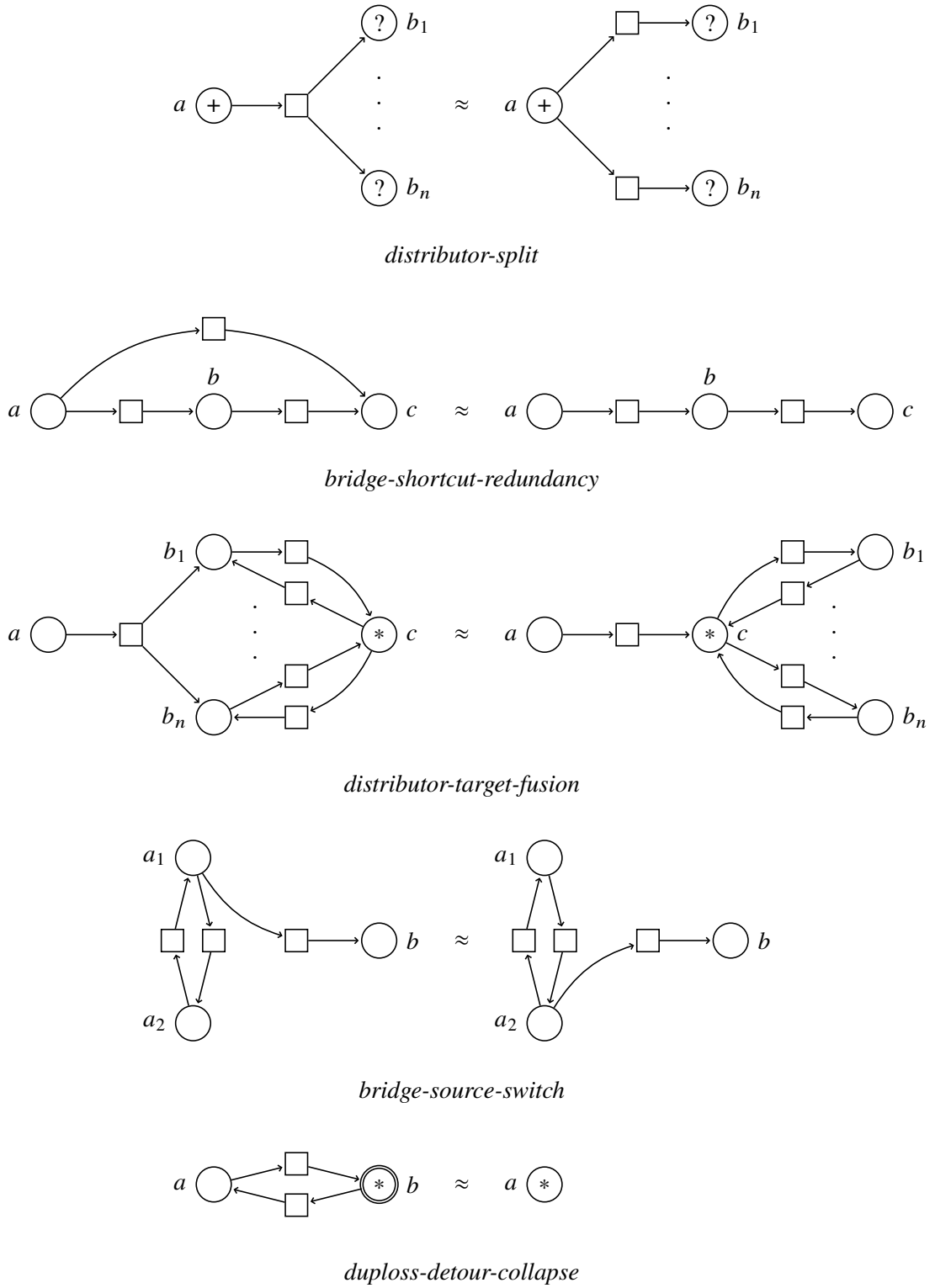


Figure 4: The key bisimilarities used by the proof

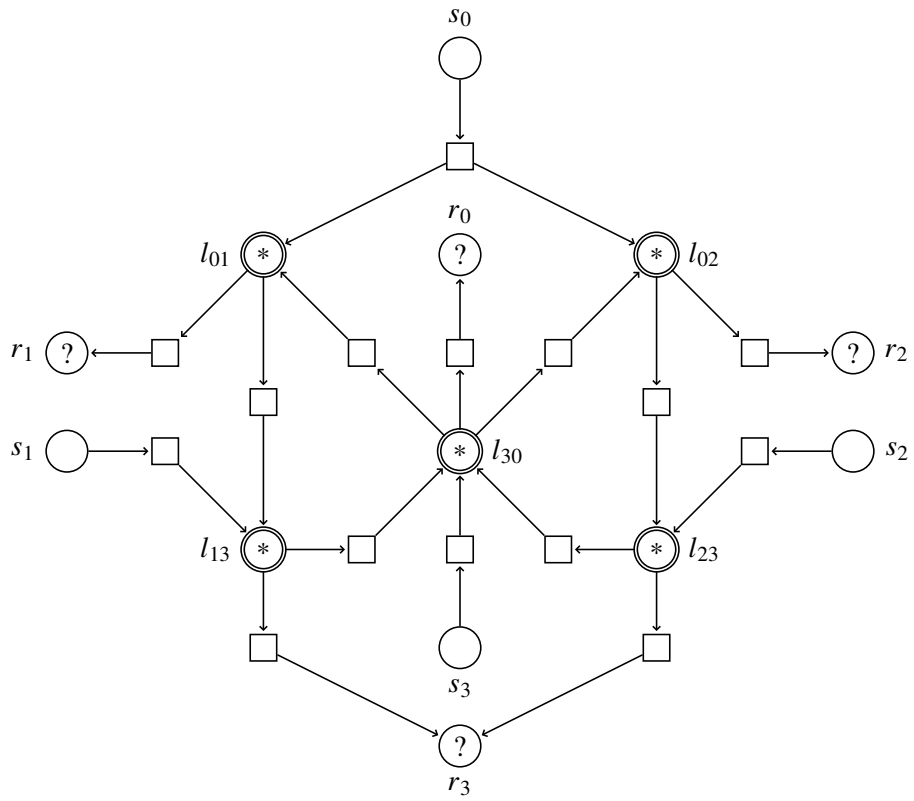


Figure 5: The process after untangling of receiving and relaying

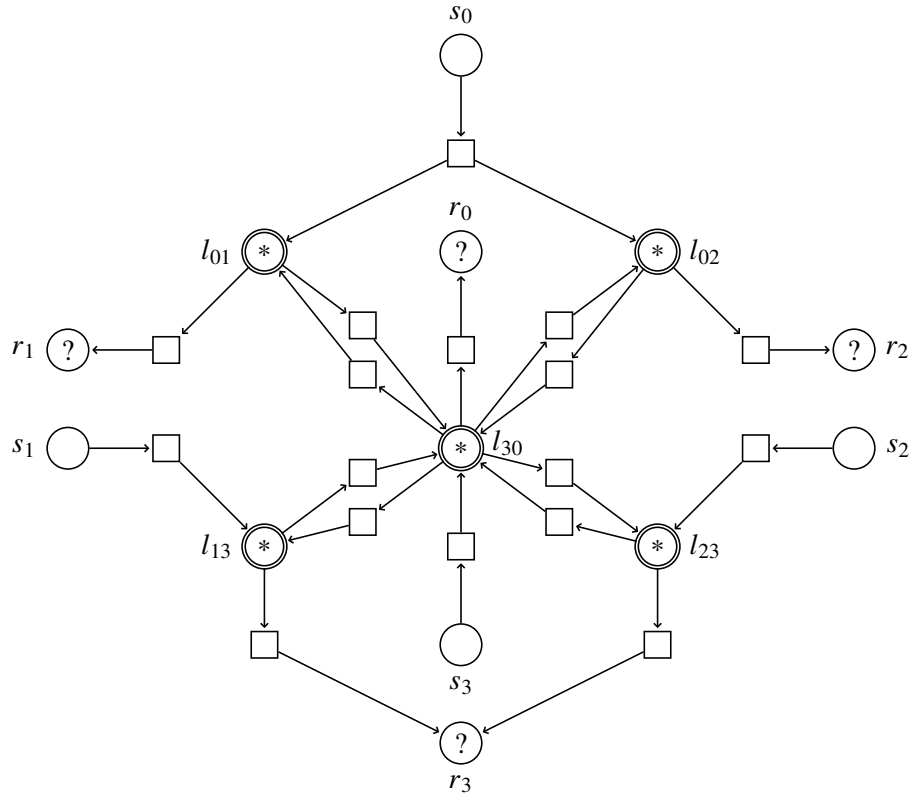


Figure 6: The process after transforming the core

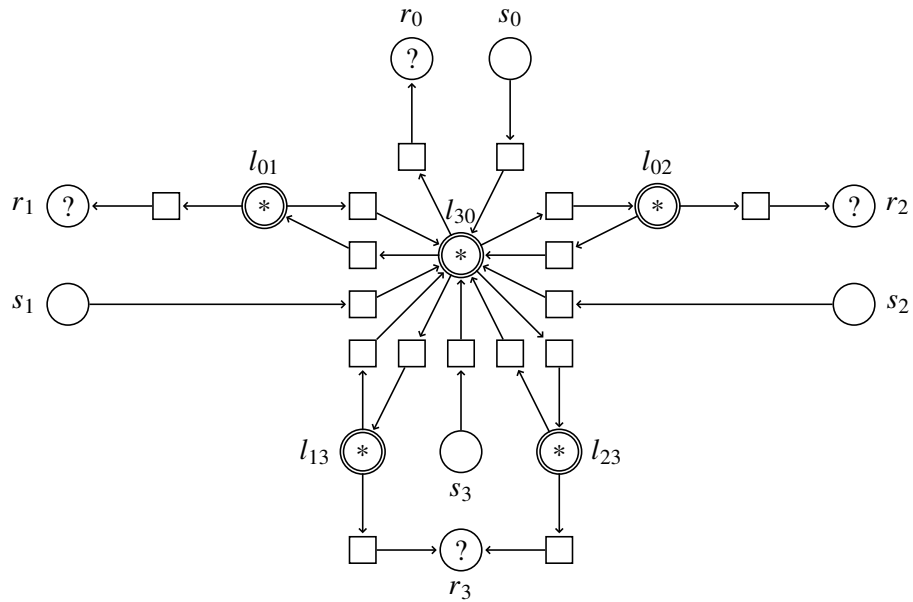


Figure 7: The process after collapsing the sending part

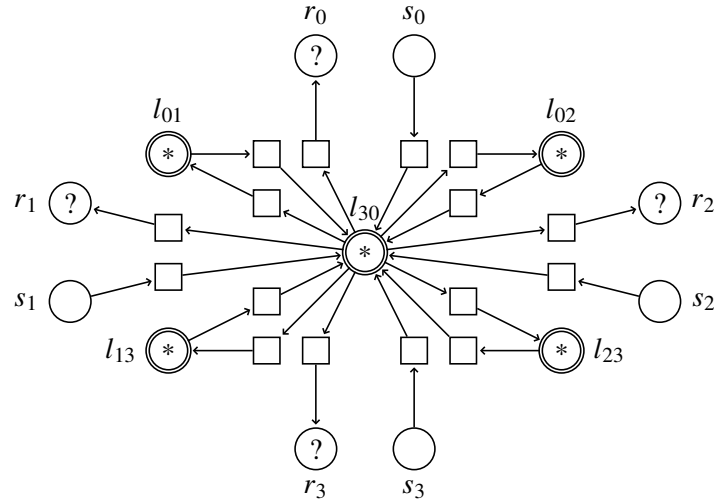


Figure 8: The process after collapsing the receiving part

4.2 Correctness of Broadcast via Multicast: Formally

We have developed our proof formally using Isabelle/HOL.² The proof is phrased in a style similar to equational reasoning, with the difference that we use weak bisimilarity instead of equality. Since Isabelle does not come with support for automated rewriting based on equivalence relations other than equality, we have developed a corresponding extension,³ which we use in our proof.

While the formal proof closely follows its communication net counterpart, it contains more technical detail. In particular, it includes applications of those bisimilarities that are implicit in the communication net notation, as described in Sect. 2. However, our engine for equivalence-based rewriting is able to automatically find any proof that consists of a chain of rewriting steps involving only these bisimilarities and optionally certain idempotency statements, which are bisimilarities as well. As a result, we can bundle consecutive applications of all these bisimilarities in our formal proof, which therefore is still reasonably concise and readable.

To illustrate the general style of the proof, we show how the first transformation step is conducted. We do not use Isabelle syntax for this presentation, in order to make it slightly more accessible. However, the Isabelle version is almost identical to what we show here, the main difference being that it mentions the repeated use of the equivalence reasoner and the lemmas to use as rewrite rules. In the proof snippet, we use the notation $\prod a \leftarrow [b_1, \dots, b_n]. P$, which stands for $P[b_1/a] \parallel \dots \parallel P[b_n/a]$ and has higher precedence than \parallel .

The first transformation step consists of three substeps, which are presented in Fig. 9. The second substep is where the *distributor-split* lemma is invoked. The first and the third substep apply bisimilarities implicit in the communication net notation and idempotency statements: the first substep does so to enable the invocation of the *distributor-split* lemma; the third substep does so to undo modifications done by the first substep and further streamline the term.

²See <https://github.com/input-output-hk/network-equivalences>.

³See <https://github.com/input-output-hk/equivalence-reasoner>.

$$\begin{aligned}
\mathfrak{A}^2 r_0 \parallel \mathfrak{A}^2 r_1 \parallel \mathfrak{A}^2 r_2 \parallel \mathfrak{A}^2 r_3 \parallel M_* \parallel M_o &\approx (\mathfrak{A}^+ l_{01} \parallel \prod a \leftarrow [r_1, l_{13}]. \mathfrak{A}^2 a \parallel l_{01} \Rightarrow [r_1, l_{13}]) \parallel \\
&(\mathfrak{A}^+ l_{02} \parallel \prod a \leftarrow [r_2, l_{23}]. \mathfrak{A}^2 a \parallel l_{02} \Rightarrow [r_2, l_{23}]) \parallel \\
&(\mathfrak{A}^+ l_{13} \parallel \prod a \leftarrow [r_3, l_{30}]. \mathfrak{A}^2 a \parallel l_{13} \Rightarrow [r_3, l_{30}]) \parallel \\
&(\mathfrak{A}^+ l_{23} \parallel \prod a \leftarrow [r_3, l_{30}]. \mathfrak{A}^2 a \parallel l_{23} \Rightarrow [r_3, l_{30}]) \parallel \\
&(\mathfrak{A}^+ l_{30} \parallel \prod a \leftarrow [r_0, l_{01}, l_{02}]. \mathfrak{A}^2 a \parallel l_{30} \Rightarrow [r_0, l_{01}, l_{02}]) \\
&\approx (\mathfrak{A}^+ l_{01} \parallel \prod a \leftarrow [r_1, l_{13}]. \mathfrak{A}^2 a \parallel \prod a \leftarrow [r_1, l_{13}]. l_{01} \rightarrow a) \parallel \\
&(\mathfrak{A}^+ l_{02} \parallel \prod a \leftarrow [r_2, l_{23}]. \mathfrak{A}^2 a \parallel \prod a \leftarrow [r_2, l_{23}]. l_{02} \rightarrow a) \parallel \\
&(\mathfrak{A}^+ l_{13} \parallel \prod a \leftarrow [r_3, l_{30}]. \mathfrak{A}^2 a \parallel \prod a \leftarrow [r_3, l_{30}]. l_{13} \rightarrow a) \parallel \\
&(\mathfrak{A}^+ l_{23} \parallel \prod a \leftarrow [r_3, l_{30}]. \mathfrak{A}^2 a \parallel \prod a \leftarrow [r_3, l_{30}]. l_{23} \rightarrow a) \parallel \\
&(\mathfrak{A}^+ l_{30} \parallel \prod a \leftarrow [r_0, l_{01}, l_{02}]. \mathfrak{A}^2 a \parallel \prod a \leftarrow [r_0, l_{01}, l_{02}]. l_{30} \rightarrow a) \\
&\approx \mathfrak{A}^2 r_0 \parallel \mathfrak{A}^2 r_1 \parallel \mathfrak{A}^2 r_2 \parallel \mathfrak{A}^2 r_3 \parallel M_* \parallel \\
&l_{01} \rightarrow r_1 \parallel l_{02} \rightarrow r_2 \parallel l_{13} \rightarrow r_3 \parallel l_{23} \rightarrow r_3 \parallel l_{30} \rightarrow r_0 \parallel \\
&l_{01} \rightarrow l_{13} \parallel l_{02} \rightarrow l_{23} \parallel l_{13} \rightarrow l_{30} \parallel l_{23} \rightarrow l_{30} \parallel l_{30} \rightarrow l_{01} \parallel l_{30} \rightarrow l_{02}
\end{aligned}$$

Figure 9: The first transformation step in detail

5 Related Work

The correctness of broadcast via multicast, commonly referred to as *network flooding*, and similar techniques has been studied to some extent in the literature. In the following, we discuss two characteristic examples:

- Bani-Abdelrahman [2] formally specifies synchronous and bounded asynchronous flooding algorithms using LTL and verifies them using the model checker NuSMV. His results are limited to small network sizes and fixed delays, though.
- Bar-Yehuda et al. [3] emulate a single-hop (direct-broadcast) network with a multi-hop network using a synchronous gossiping algorithm. With a gossiping algorithm, a node does not relay an incoming packet to all neighboring nodes, but only to a randomly chosen one.

The existing literature, however, seems to lack the study of an *equivalence* of the two broadcasting approaches, which we provide in this work. The reason for this lack may be related to the complications that arise due to the behavioral mismatches explained in Sect. 3.

There is a vast amount of literature on the relationship between process calculi and Petri nets. In particular, a long line of research has been developed with the theme of giving Petri net semantics to process calculi [5, 7, 11], providing process calculi with operational semantics expressing true concurrency as opposed to the traditional interleaving semantics. Another line of research approaches the reverse problem, that is, finding process calculi that are suitable for modeling Petri nets of certain classes [4, 8]. Our tandem of the communication language and its communication net notation, described in Sect. 2, fits right into all this research: the communication language can be regarded as a restricted process calculus that comes with a Petri net semantics and models the class of hierarchical Petri nets with exactly one input place per transition.

6 Conclusion

We have defined a language for describing communication networks, which is embedded in a process calculus and corresponds to a class of Petri nets. Based on the connection to Petri nets, we have devised a graphical notation for our language. Building on this foundation, we have proved behavioral equivalence of broadcast via multicast and direct broadcast for a typical pair of networks. The graphical notation has allowed us to reason in an intuitive way on an abstract level, while the embedding in a process calculus has permitted us to develop a fully machine-checked proof. For specifying the equivalence of the two realizations of broadcast, we have devised the notion of weak bisimilarity up to loss.

7 Ongoing and Future Work

At the moment, we are completing the proofs of the fundamental lemmas that the correctness proof shown in Sect. 4 uses. Furthermore, we are working on a variant of our correctness proof that deals with broadcast integrated with packet filtering according to a fixed predicate.

An important task for the future is the generalization of our correctness proofs to apply to arbitrary strongly connected multicast networks and their direct-broadcast counterparts. Furthermore, we may prove a modified correctness statement where the receive channels of direct broadcast are not lossy. Doing so would clarify that broadcast via multicast has the more constrained behavior, but would require adjustments to the specification of network behavior. Finally, we consider generalizing the proof about broadcast integrated with filtering to work with state-dependent filters.

Acknowledgements

This work was funded by Input Output. We are thankful to Input Output for giving us the opportunity to work on numerous interesting topics, including the one described in this paper. Furthermore, we want to thank James Chapman, Duncan Coutts, Kevin Hammond, and Philipp Kant, who supported us in our work on network equivalences and the development of all the theory and the vast amount of Isabelle formalizations that underlie it.

References

- [1] Christian Badertscher, Peter Gaži, Aggelos Kiayias, Alexander Russell & Vassilis Zikas (2018): *Ouroboros Genesis: Composable Proof-of-Stake Blockchains with Dynamic Availability*. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ACM, New York, pp. 913–930, doi:10.1145/3243734.3243848. Available at <https://iohk.io/en/research/library/papers/ouroboros-genesiscomposable-proof-of-stake-blockchains-with-dynamic-availability/>.
- [2] Ra'Ed Bani-Abdelrahman (2018): *Specification and Verification of Network Algorithms using Temporal Logic*. Ph.D. thesis, Loughborough University, Loughborough, England, doi:10.26174/thesis.lboro.8138030.v1.
- [3] Reuven Bar-Yehuda, Oded Goldreich & Alon Itai (1989): *Efficient Emulation of Single-Hop Radio Network with Collision Detection on Multi-Hop Radio Network with No Collision Detection*. In Jean-Claude Bermond & Michel Raynal, editors: *Distributed Algorithms, Lecture Notes in Computer Science 392*, Springer, Berlin/Heidelberg, Germany, pp. 24–32, doi:10.1007/3-540-51687-5_29.
- [4] Twan Basten & Marc Voorhoeve (1995): *An Algebraic Semantics for Hierarchical P/T Nets*. In Giorgio De Michelis & Michel Diaz, editors: *Application and Theory of Petri Nets 1995, Lecture Notes in Computer Science 935*, Springer, Berlin/Heidelberg, Germany, pp. 45–65, doi:10.1007/3-540-60029-9_33.

- [5] Eike Best, Raymond Devillers & Maciej Koutny (2001): *A Unified Model for Nets and Process Algebras*. In J. A. Bergstra, A. Ponse & S. A. Smolka, editors: *Handbook of Process Algebra*, chapter 14, Elsevier, Amsterdam, The Netherlands, pp. 873–944, doi:10.1016/B978-044482830-9/50032-1.
- [6] Bernardo David, Peter Gaži, Aggelos Kiayias & Alexander Russell (2018): *Ouroboros Praos: An Adaptively-Secure, Semi-Synchronous Proof-of-Stake Blockchain*. In Jesper Buus Nielsen & Vincent Rijmen, editors: *Advances in Cryptology – EUROCRYPT 2018, Lecture Notes in Computer Science 10821*, Springer, Berlin/Heidelberg, Germany, pp. 66–98, doi:10.1007/978-3-319-78375-8_3. Available at <https://iohk.io/en/research/library/papers/ouroboros-praosan-adaptively-securesemi-synchronous-proof-of-stake-protocol/>.
- [7] Pierpaolo Degano, Rocco De Nicola & Ugo Montanari (1988): *A Distributed Operational Semantics for CCS Based on Condition/Event Systems*. *Acta Informatica* 26(1–2), pp. 59–91, doi:10.1007/BF02915446.
- [8] Roberto Gorrieri (2017): *Process Algebras for Petri Nets*. Monographs in Theoretical Computer Science. An EATCS Series, Springer, Berlin/Heidelberg, Germany, doi:10.1007/978-3-319-55559-1.
- [9] Kohei Honda & Mario Tokoro (1991): *An Object Calculus for Asynchronous Communication*. In Pierre America, editor: *ECOOP '91 European Conference on Object-Oriented Programming, Lecture Notes in Computer Science 512*, Springer, Berlin/Heidelberg, Germany, pp. 133–147, doi:10.1007/BFb0057019.
- [10] Aggelos Kiayias, Alexander Russell, Bernardo David & Roman Oliynykov (2017): *Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol*. In Jonathan Katz & Hovav Shacham, editors: *Advances in Cryptology – CRYPTO 2017, Lecture Notes in Computer Science 10401*, Springer, Berlin/Heidelberg, Germany, pp. 357–388, doi:10.1007/978-3-319-63688-7_12. Available at <https://iohk.io/en/research/library/papers/ouroborosa-provably-secure-proof-of-stake-blockchain-protocol/>.
- [11] Ernst-Rüdiger Olderog (1991): *Nets, Terms and Formulas*. *Cambridge Tracts in Theoretical Computer Science* 23, Cambridge University Press, Cambridge, England, doi:10.1017/CBO9780511526589.