

Adding data registers to parameterized networks with broadcast

Giorgio Delzanno

DIBRIS, University of Genova

Italy

Arnaud Sangnier

LIAFA, Univ Paris Diderot, Paris Cité Sorbonne, CNRS

France

Riccardo Traverso

DIBRIS, University of Genova

Italy

Abstract. We study parameterized verification problems for networks of interacting register automata. The network is represented through a graph, and processes may exchange broadcast messages containing data with their neighbours. Upon reception a process can either ignore a sent value, test for equality with a value stored in a register, or simply store the value in a register. We consider safety properties expressed in terms of reachability, from arbitrarily large initial configurations, of a configuration exposing some given control states and patterns. We investigate, in this context, the impact on decidability and complexity of the number of local registers, the number of values carried by a single message, and dynamic reconfigurations of the underlying network.

1. Introduction

Distribution is at the core of modern computer applications. They usually involve partially synchronized entities, use different communication means, and manipulate data like identifiers and time-stamps. For all these reasons, distributed algorithms are a challenging test-case for automated verification methods [24]. Several examples of distributed algorithms are based on the assumption that individual processes follow the same protocol. Methods like model checking are not always directly applicable to this class of algorithms. Indeed, they normally require to fix the initial system configuration or the maximum number of components.

One way to validate distributed algorithms parametric in the number of involved agents consists in searching for finite model properties, e.g., cut-off values for the parameters. In their seminal paper [22],

German and Sistla propose a model where each entity in the system executes the same finite state protocol and where the communication is achieved via *rendez-vous* communication. They exhibit cut-off properties for finding a minimal number of entities that expose a violation to a given property. They also rely on the idea that in such systems, one does not need to know precisely the state of each process, but that it is enough to count the number of processes in each state, this idea is known as the *counting abstraction*. These ideas have then been extended to other parameterized systems with different characteristics: for instance, in [17] and [19] Emerson and Namjoshi and Esparza et al. propose the model of broadcast protocols which extends the model in [22] by allowing the entities to communicate either via *rendez-vous* or via broadcast. In the broadcast operation all the entities that can react to a message have to react. To decide coverability, the authors apply the theory of well-structured transition systems [1, 21] formulated on the counting abstraction of the considered model. Constraint-based methods for the analysis of broadcast protocols have been considered in [10]. In a recent paper [20], Esparza and Ganty introduce a parameterized model in which communication is achieved via a finite set of shared variables storing finite domain values. In [18], Esparza presents a survey of some of the main results for the above mentioned parameterized models. Parameterized verification of systems composed by repeated components have also been proposed by considering different means of communication as token-passing [5, 9], message passing [7], or *rendez-vous* over an infinite domain of data in [11].

introduced in [19, 22], Delzanno et al. propose in [15] a model of ad-hoc networks based on the following consideration: communication in ad-hoc networks is often based on broadcast, but only the entities in the transmission range can receive emitted messages. For this reason, they propose a simple parameterized model, that we will refer to as AHN (for Ad Hoc Network), where each system node executes the same protocol define by a finite state automaton labeled with broadcast and reception actions. Configuration are equipped with a communication topology (defined as a graph). In this model broadcast messages belong to a finite alphabet. The verification problems consists then in asking whether there exists a number of entities and a communication topology such that an execution of the protocol exhibits some anomalies. For this model, they prove that coverability (or reachability of a configuration exhibiting a bad control state) is undecidable [15]. Decidability can be regained by restricting the class of allowed topologies, e.g., by considering bounded path communication topologies (in which the length of the longest simple path is bounded) [15], or clique graphs [16] where broadcast messages are received by all the nodes in the networks, or a mix of these two notions [16]. Decidability results can also be obtained with mobility or non-deterministic reconfiguration of the communication topology at any moment [15]. In reconfigurable AHNs a node may disconnect from its neighbors and connect to other ones at any time during a computation. This behavior models in a natural way unexpected power-off and dynamic movement of devices. For the latter restriction, it has been proved that checking the reachability of a configuration where some control states are present can be done in polynomial time [14]. Furthermore, testing for the absence of some control state in a configuration to be reached renders the problem NP-complete [14]. We point out the fact that the model of AHN with fully connected topologies (or clique communication topologies) is equivalent to the model of broadcast protocols introduced in [19] without *rendez-vous* communication. The model of AHN has also been extended in different ways: considering finite protocols equipped with independent clocks *à la timed automata* [4] evolving at the same rate [2], or finite protocols with probabilities [6].

In this paper we study an extension of the model of AHN with reconfiguration originally introduce in [15] and studied more deeply in [14] where we consider that the messages that are broadcast belong now to an infinite alphabet. We assume furthermore that each node in the network is equipped with a

finite set of registers. The resulting model called Broadcast Networks of Register Automata (BNRA) is aimed at modeling both the local knowledge of distributed nodes as well as their interaction via broadcast communication. As in AHN, a network is modeled via a finite graph where each node runs an instance of a common protocol. A protocol is specified via a register automaton, an automaton equipped with a finite set of registers [23], where each register assumes values taken from the set of natural numbers. Node interaction is specified via broadcast communication where messages are allowed to carry data, that can be assigned to or tested against the local registers of receivers. The resulting model can be used to reason about core parts of client-server protocols as well as of routing protocols, e.g. route maintenance as in Link Reversal Routing. We focus our attention on the decidability and complexity of parameterized verification of safety properties, i.e., the problem of finding a sufficient number of nodes and an initial topology that may lead to a configuration exposing a bad pattern. The considered class of verification problems is parametric in four dimensions: the number of nodes, the topology of the initial configuration to be discovered, and the amounts of data contained in local registers and exchanged messages. The peculiarity of our model is that messages are now data from an infinite domain and that interaction is restricted according to an underlying communication graph. Distributed algorithms often manipulate data belonging to an infinite domain such as identifiers of the agents of the network.

In our analysis we study the decidability status of some coverability problem for this model taking into account the number of registers of individual nodes and the number of fields in the messages. For messages with no data field (and hence no register as well in the nodes), our model boils down to AHN, and we know that the coverability problem is undecidable for arbitrary topologies without reconfiguration, while decidability is regained for fully connected and bounded-path topologies or by taking into account reconfiguration [15, 16]. We study here whether these last decidability results still hold when extending the protocols with registers over infinite data value and fields in the messages. We draw the following decidability frontier.

- When reconfiguration is allowed, we show that:
 - The coverability problem is undecidable if nodes have at least two registers and messages have at least two fields.
 - If we restrict the number of data fields in the messages to be less than or equal to one, we regain decidability (without any bound on the number of allowed registers). The decision algorithm is based on a saturation procedure that operates on a graph-based symbolic representation of sets of configurations in which the data are abstracted away. This representation uses the relations between data (equality, inequality) and is inspired by similar techniques used in the case of classical register automata [23]. We prove that in this case the problem is PSPACE-complete.
- For fully connected topologies without reconfiguration, we have that:
 - The coverability problem is undecidable when nodes are equipped with at least two registers and messages with at least one field;
 - On the other hand if we restrict the number of register to be less than or equal to one and the number of data field per message to be also less than or equal to one, then the coverability problem becomes decidable. The decidability proof exploits the theory of well-structured

transition systems [3, 21]. We obtain as well a non-elementary lower bound which follows from a reduction from coverability in reset nets [28].

This paper corresponds to a completed version of [12].

2. Broadcast Networks of Register Automata

2.1. Syntax and semantics

We model a distributed network using a graph in which the behavior of each node is described via an automaton with operations over a finite set of registers. A node can transmit part of its current data to adjacent nodes using broadcast messages. A message carries both a type and a finite tuple of data. Receivers can test/store/ignore the data contained inside a message. We assume that broadcasts and receptions are executed without delays (i.e. we simultaneously update the state of sender and receiver nodes).

Actions Let us first describe the set of actions. We use $r \geq 0$ to denote the number of registers in each node. We use $f \geq 0$ to denote the number of data fields available in each message and we consider a finite alphabet Σ of message types. We often use $[i..j]$ to denote the set $\{k \in \mathbb{N} \mid i \leq k \leq j\}$. We also assume that if $r = 0$ then $f = 0$ (no registers, no information to transmit). The set of broadcast actions parameterized by r , f and Σ is defined follows:

$$Send_{\Sigma}^{r,f} = \{\mathbf{b}(m, p_1, \dots, p_f) \mid m \in \Sigma \text{ and } p_i \in [1..r] \text{ for } i \in [1..f]\}$$

The action $\mathbf{b}(a, p_1, \dots, p_f)$ corresponds to a broadcast message of type a whose i -th field contains the value of the p_i -th register of the sending node. For instance, for $r = 2$ and $f = 4$, $\mathbf{b}(req, 1, 1, 2, 1)$ corresponds to a message of type req in which the current value of the register 1 of the sender is copied in the first two fields and in the last field, and the current value of register 2 of the sender is copied into the third field.

A receiver node can then either compare the value of a message field against the current value of a register, store the value of a message field in a register, or simply ignore a message field. Reception actions parameterized by r , f and Σ are defined as follows:

$$Rec_{\Sigma}^{r,f} = \left\{ \mathbf{r}(m, \alpha_1, \dots, \alpha_f) \left| \begin{array}{l} m \in \Sigma, \alpha_i \in Act^r \text{ for } i \in [1..f] \\ \text{and if } \alpha_i = \alpha_j = \downarrow k \text{ then } i = j \end{array} \right. \right\}$$

where the set of field actions Act^r is: $\{?k, ?\bar{k}, \downarrow k, * \mid k \in [1..r]\}$. When used in a given position of a reception action, $?k$ [resp. $?\bar{k}$] tests whether the content of the k -th register is equal [resp. different] to the corresponding value of the message, $\downarrow k$ is used to store the corresponding value of the message into the k -th register, and $*$ is used to denote that the corresponding value is ignored.

As an example, for $r = 2$ and $f = 4$, $\mathbf{r}(req, ?\bar{2}, ?1, *, \downarrow 1)$ specifies the reception of a message of type req in which the first field is tested for inequality against the current value of the second register, the second field is tested for equality against the first register, the third field is ignored, and the fourth field is assigned to the first register. We now provide the definition of a protocol that models the behavior of an individual node.

Definition 2.1. A (r, f) -protocol over Σ is a tuple $\mathcal{P} = \langle Q, R, q_0 \rangle$ where: Q is a finite set of control states, $q_0 \in Q$ is an initial control state, and $R \subseteq Q \times (\text{Send}_{\Sigma}^{r,f} \cup \text{Rec}_{\Sigma}^{r,f}) \times Q$ is a set of broadcasting and reception rules.

In the rest of the paper we call a (r, f) -protocol over Σ simply a (r, f) -protocol when the alphabet is clear from the context.

A configuration is a graph in which nodes represent the current state of the corresponding protocol instance running on it (control state and current value of registers) and edges denote communication links. In this paper we assume that the value of registers are naturals. Therefore, a valuation of registers is defined as a map from register positions to naturals. More formally, a configuration γ of a (r, f) -protocol $\mathcal{P} = \langle Q, R, q_0 \rangle$ is an undirected graph $\langle V, E, L \rangle$ such that V is a finite set of nodes, $E \subseteq V \times V \setminus \{(v, v) \mid v \in V\}$ is a set of edges, and $L : V \rightarrow Q \times \mathbb{N}^r$ is a labeling function (current valuation of registers).

Before we give the semantics of our model, we introduce some auxiliary notations. Let $\gamma = \langle V, E, L \rangle$ be a configuration. For a node $v \in V$, we denote by $L_Q(v)$ and $L_M(v)$ the first and second projection of $L(v)$. For $u, v \in V$, we write $u \sim_{\gamma} v$ – or simply $u \sim v$ when γ is clear from the context – the fact that $(u, v) \in E$, i.e. the two nodes are neighbors. Finally, the configuration γ is said to be initial if $L_Q(v) = q_0$ for all $v \in V$ and, for all $u, v \in V$ and all $i, j \in [1..r]$, if $u \neq v$ or $i \neq j$ then $L_M(v)[i] \neq L_M(v)[j]$. Consequently in an initial configuration, all the registers of the nodes contain different values. Note that we could have consider a different semantics with no restriction on the contents of the registers in the initial configurations. We comment this point in the conclusion section.

We write Γ [resp. Γ_0] for the set of all [resp. initial] configurations, and Γ^{fc} [resp. Γ_0^{fc}] for the set of configurations [resp. initial configurations] $\langle V, E, L \rangle$ that are fully connected, i.e. such that $E = V \times V \setminus \{(v, v) \mid v \in V\}$. Note that for a given (r, f) -protocol the sets Γ , Γ_0 , Γ^{fc} , and Γ_0^{fc} are infinite since we do not impose any restriction on the number of processes present in the graph.

Furthermore, from two nodes u and v of a configuration $\gamma = \langle V, E, L \rangle$ and a broadcast action of the form $\mathbf{b}(m, p_1, \dots, p_f)$, let $\mathcal{R}(v, u, \mathbf{b}(m, p_1, \dots, p_f)) \subseteq Q \times \mathbb{N}^r$ be the set of the possible labels that can take u on reception of the corresponding message sent by v , i.e. we have $(q'_r, M) \in \mathcal{R}(v, u, \mathbf{b}(m, p_1, \dots, p_f))$ if and only if there exists a receive action of the form $\langle L_Q(u), \mathbf{r}(m, \alpha_1, \dots, \alpha_f), q'_r \rangle \in R$ verifying the two following conditions:

- (1) For all $i \in [1..f]$, if there exists $j \in [1..r]$ s.t. $\alpha_i = ?j$ [resp. $\alpha_i = ?\bar{j}$], then $L_M(u)[j] = L_M(v)[p_i]$ [resp. $L_M(u)[j] \neq L_M(v)[p_i]$];
- (2) For all $j \in [1..r]$, if there exists $i \in [1..f]$ such that $\alpha_i = \downarrow j$ then $M[j] = L_M(v)[p_i]$ otherwise $M[j] = L_M(u)[j]$.

Given a (r, f) -protocol $\mathcal{P} = \langle Q, R, q_0 \rangle$, we define a Broadcast Network of Register Automata (BNRA) as the transition system $BNRA(\mathcal{P}) = \langle \Gamma, \Rightarrow, \Gamma_0 \rangle$ where Γ [resp. Γ_0] is the set of all [resp. initial] configurations and $\Rightarrow \subseteq \Gamma \times \Gamma$ is the transition relation defined as follows: for $\gamma = \langle V, E, L \rangle$ and $\gamma' = \langle V', E', L' \rangle \in \Gamma$, we have $\gamma \Rightarrow \gamma'$ if and only if $V = V'$ and one of the following conditions holds:

- (Broadcast)** $E = E'$ and there exist $v \in V$ and $\langle q, \mathbf{b}(m, p_1, \dots, p_f), q' \rangle \in R$ such that $L_Q(v) = q$, $L'_Q(v) = q'$ and for all $u \in V \setminus \{v\}$:
- if $u \sim v$ then $L'(u) \in \mathcal{R}(v, u, \mathbf{b}(m, p_1, \dots, p_f))$, or, $\mathcal{R}(v, u, \mathbf{b}(m, p_1, \dots, p_f)) = \emptyset$ and $L(u) = L'(u)$;

- if $u \approx v$, then $L(u) = L'(u)$.

(Reconfiguration) $L = L'$ (no constraint on new edges E').

Reconfiguration steps model dynamic changes of the connection topology, e.g., loss of links and messages or node movement. An internal transition τ can be defined using a broadcast of a special message such that there are no reception rules associated to it. A register $j \in [1..r]$ is said to be read-only if and only if there is no $\langle q, \mathbf{r}(m, \alpha_1, \dots, \alpha_f), q' \rangle \in R$ and $i \in [1..f]$ such that $\alpha_i = \downarrow j$. Read-only registers can be used as identifiers of the associated nodes.

Given $BNRA(\mathcal{P}) = \langle \Gamma, \Rightarrow, \Gamma_0 \rangle$, we use \Rightarrow_b to denote the restriction of \Rightarrow to broadcast steps only, and \Rightarrow^* [resp. \Rightarrow_b^*] to denote the reflexive and transitive closure of \Rightarrow [resp. \Rightarrow_b]. Now we define the set of reachable configurations as: $Reach(\mathcal{P}) = \{\gamma' \in \Gamma \mid \exists \gamma \in \Gamma_0 \text{ s.t. } \gamma \Rightarrow^* \gamma'\}$, $Reach^b(\mathcal{P}) = \{\gamma' \in \Gamma \mid \exists \gamma \in \Gamma_0 \text{ s.t. } \gamma \Rightarrow_b^* \gamma'\}$, and $Reach^{fc}(\mathcal{P}) = Reach^b(\mathcal{P}) \cap \Gamma^{fc}$.

2.2. Coverability Problem

Our goal is to decide whether there exists an initial configuration (of any size and topology) from which it is possible to reach a configuration exposing (covered by with respect to graph inclusion) a bad pattern. We express bad patterns using reachability queries defined as follows. Let $\mathcal{P} = \langle Q, R, q_0 \rangle$ be a (r, f) -protocol and Z a denumerable set of variables. A reachability query φ for \mathcal{P} is a formula generated by the following grammar:

$$\varphi ::= q(\mathbf{z}) \mid M_i(\mathbf{z}) = M_j(\mathbf{z}') \mid M_i(\mathbf{z}) \neq M_j(\mathbf{z}') \mid \varphi \wedge \varphi$$

where $\mathbf{z}, \mathbf{z}' \in Z$, $q \in Q$ and $i, j \in [1..r]$. We now define the satisfiability relation for such queries. Given a configuration $\gamma = \langle V, E, L \rangle \in \Gamma$, a valuation is a function $g : Z \mapsto V$. The satisfaction relation \models is parameterized by a valuation and is defined inductively as follows:

- $\gamma \models_g q(\mathbf{z})$ if and only if $L_Q(g(\mathbf{z})) = q$,
- $\gamma \models_g M_i(\mathbf{z}) = M_j(\mathbf{z}')$ if and only if $L_M(g(\mathbf{z}))[i] = L_M(g(\mathbf{z}'))[j]$,
- $\gamma \models_g M_i(\mathbf{z}) \neq M_j(\mathbf{z}')$ if and only if $L_M(g(\mathbf{z}))[i] \neq L_M(g(\mathbf{z}'))[j]$,
- $\gamma \models_g \varphi \wedge \varphi'$ if and only if $\gamma \models_g \varphi$ and $\gamma \models_g \varphi'$.

We say that a configuration γ satisfies a reachability query φ , denoted by $\gamma \models \varphi$ if and only if there exists a valuation g such that $\gamma \models_g \varphi$. Furthermore we assume that our queries do not contain contradictions with respect to $=$ and \neq . This query language mediates between expressiveness and simplicity, enabling us to search for graph patterns involving both control states and register values. We can now provide the definition of the parameterized coverability problem, which consists in finding an initial configuration that leads to a configuration containing in which the query can be matched.

Definition 2.2. The problem $Cov(r, f)$ is defined as follows: given a (r, f) -protocol \mathcal{P} and a reachability query φ , does there exist $\gamma \in Reach(\mathcal{P})$ such that $\gamma \models \varphi$?

The problem $Cov^b(r, f)$ [resp. $Cov^{fc}(r, f)$] is obtained by replacing the reachability set with $Reach^b(\mathcal{P})$ [resp. $Reach^{fc}(\mathcal{P})$]. Finally, $Cov(*, f)$ denotes the disjunction of the problems $Cov(r, f)$ varying on $r \geq 0$ (i.e. for any (finite) number of registers).

Note that these problems belong to the class of coverability problem since we seek a configuration which ‘‘covers’’ the query, in other words a configuration which contains a subpart respecting a reachability query. Furthermore in our context, strict reachability problems, where one asks whether a configuration is reachable, are easier to solve, since when we fix a final configuration we know the number of nodes present in all the previous configurations (since during an execution this number does not change) and hence the problem boils down to the verification of a finite state system.

For the cases with no register ($r = 0$) and hence no information to transmit ($f = 0$), the problems have already been studied previously. More precisely it has been shown that $Cov^b(0, 0)$ is undecidable [15] and that $Cov^{fc}(0, 0)$ is decidable [16, 19] and Ackermann-complete [27] and that $Cov(0, 0)$ [14] can be solved in polynomial time.

3. An Example: Route Discovery Protocol

We describe here the behavior of our model on an example Consider the problem of building a route from nodes of type *sender* to nodes of type *dest*. We assume that nodes are equipped with two registers, called *id* and *next*, used to store a pointer to the next node in the route to *dest*. The protocol that collects such information is defined in Figure 1.

Initially nodes have type *sender*, *idle*, and *dest*. Request messages like *rreq* are used to query

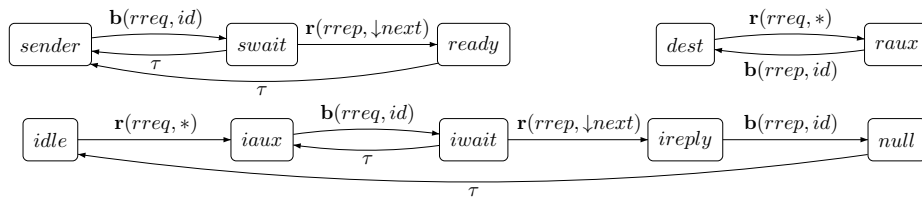


Figure 1. Route discovery example

adjacent nodes in search for a valid neighbor. Back edges are used to restart the protocol in case of loss of intermediate messages or no reply at all. An instance of the protocol starts with a node in state *sender* broadcasts a route request *rreq*, attaching his identifier to the message, and waits. Intermediate nodes in state *idle* react to it by forwarding another *rreq* with their identifier, and then they wait too for a reply. The protocol goes on until an *rreq* message finally reaches a destination, a node in state *dest*, which replies by providing its identifier with an *rrep* message to its vicinity. All of the intermediate nodes involved save in the local register *next* the value from the *rrep* message, and send another *rrep* message with their identifier to the neighbors, to notify them that they are on the route to reach the destination. When an *rrep* message arrives to the sender, it saves the identifier of the *next* hop and the route is established.

In this example an undesired state is, e.g., any configuration in which two adjacent nodes n and n' point to each other. Bad patterns like this one can be specified using a query like $ready(z_1) \wedge ready(z_2) \wedge M_{id}(z_1) = M_{next}(z_2) \wedge M_{next}(z_1) = M_{id}(z_2)$.

Note that in this work we are mainly interested in safety properties, or more precisely, properties that can be checked thanks to reachability queries. On this example, another interesting property could be to check whether the protocol builds eventually a route from *sender* to *dest*. Such a property would involve a more complex reasoning than the one we currently propose and it will be hence more difficult to tackle.

4. Reconfiguration in Arbitrary Graphs

4.1. Undecidability of $Cov(2, 2)$

Our first result is the undecidability of coverability for nodes with two registers (one read-only) and messages with two data fields. The proof is based on a reduction from reachability in two counter machines. The reduction builds upon an election protocol that can be applied to select a linked list (of arbitrary length) of nodes in the network. The existence of such a list-builder protocol is at the core of the proof. The simulation of a two counter machine becomes easy once a list has been constructed. In this section, we assume that protocols have at least one read-only register $id \in [1..r]$. We formalize next the notion of list and list-builder that we use in the undecidability proofs presented across the paper.

We first say that a node v points to a node v' via x if the register x of v contains the same value as register id of v' . We consider a configuration $\gamma = \langle V, E, L \rangle \in \Gamma$ with $L_Q(v) \subseteq Q$ for all $v \in V$. For a set of states Q and pairwise disjoint sets $Q_a, Q_b, Q_c \subset Q$, we say that γ contains a $\langle Q_a, Q_b, Q_c \rangle$ -list (linked via x), or simply *list*, starting at v if there exists a set of nodes $\{v_1, \dots, v_k\} \subseteq V$ such that $L_Q(v_1) \in Q_a$, $L_Q(v_k) \in Q_c$, and $L_Q(v_i) \in Q_b$ for $i \in [2..k-1]$, and furthermore v_j is the unique node in V that points to v_{j+1} via x and has label in $Q_a \cup Q_b$ for $j \in [0..k-1]$. In other words sets Q_a and Q_c are sentinels for a list made of nodes with label in Q_b . A backward list is defined as before but with reversed pointers, i.e., v_{j+1} points to v_j and we say that the list ends at v .

We often write $\langle q_a, q_b, q_c \rangle$ -list as a shorthand for a $\langle \{q_a\}, \{q_b\}, \{q_c\} \rangle$ -list. For a transition relation $\rightsquigarrow \in \{\Rightarrow, \Rightarrow_b\}$, $\Gamma' \subseteq \Gamma$ and $\gamma \in \Gamma$, $\Gamma' \rightsquigarrow^* \gamma$ is true iff there exists $\gamma' \in \Gamma'$ s.t. $\gamma' \rightsquigarrow^* \gamma$. We now state the definition of list builders.

Definition 4.1. A protocol $\mathcal{P} = \langle Q, R, q_0 \rangle$ is a forward [resp. backward] $\langle q_a, q_b, q_c \rangle$ -list builder for a transition relation $\rightsquigarrow \in \{\Rightarrow, \Rightarrow_b\}$ and $\Gamma'_0 \subseteq \Gamma_0$ on $x \in [1..r]$ if, for any $\gamma = \langle V, E, L \rangle \in \Gamma$ and every $v \in V$ such that $\Gamma'_0 \rightsquigarrow^* \gamma$ and $L_Q(v) = q_a$, we have that γ contains a $\langle q_a, q_b, q_c \rangle$ -list [resp. $\langle q_c, q_b, q_a \rangle$ -list] linked via x starting at v [resp. ending at v]. Furthermore, if \rightsquigarrow is \Rightarrow_b , then $v' \sim v''$ for all successive nodes v' and v'' in the list.

We will now see how we can exploit the list (of arbitrary length) generated by a list-builder protocol to build a simulation of a two counter machine. Indeed, notice that if node v is the only one pointing to node v' then the pair of actions $\mathbf{b}(m, x)$ and $\mathbf{r}(m, ?id)$ can be used to send a message from v to v' (v' is the only node that can receive m from v). Furthermore, the pair of actions $\mathbf{b}(m, id)$ and $\mathbf{r}(m, ?x)$ can be used to send a message from v' to v (v is the only node that can receive m from v'). This property can be exploited to simulate counters by using intermediate nodes as single units (the value of the counter is the sum of unit nodes in the list). One of the sentinels is used as program location, and the links in the list are used to send messages (in two directions) to adjacent nodes to increment or decrement (update of labels) the counters. Test for zero is encoded by a double traversal of the list in order to check that each intermediate node represents zero units.

Let Q_l be the set $\{q_a, q_b, q_c\}$. We say that a forward or backward $\langle q_a, q_b, q_c \rangle$ -list builder protocol $\mathcal{P}_{lb} = \langle Q_{lb}, R_{lb}, q_0 \rangle$ is *extended* with new states Q' and rules R' when the resulting protocol $\mathcal{P} = \langle Q, R, q_0 \rangle$ first executes \mathcal{P}_{lb} reaching a state in Q_l , and then continues only in states in Q' by firing only rules in R' which preserve lists and cannot interfere with the \mathcal{P}_{lb} sub-protocol. More formally, we require that $Q = Q_{lb} \cup Q'$, $Q_{lb} \cap Q' = Q_l$, $R = R_{lb} \cup R'$, and each rule in R' cannot involve: messages $m \in \Sigma$ that appear in some rule of R_{lb} ; states in $Q_{lb} \setminus Q_l$; or store operations overwriting register x . Furthermore, there is a partitioning Q_a, Q_b , and Q_c of Q' such that $q_a \in Q_a$, $q_b \in Q_b$, $q_c \in Q_c$, and every rule in R' does not involve states belonging to different partitions. Thanks to all these conditions, while executing \mathcal{P} , $\langle q_a, q_b, q_c \rangle$ -lists may evolve into $\langle Q_a, Q_b, Q_c \rangle$ -lists while maintaining the original underlying structure. Then (e.g., with $f = 1$ and forward list builders), a message $m \in \Sigma$ can be propagated from a node v_i of the list $v_1 \cdots v_k \in V$ to the next v_{i+1} by broadcasting $\mathbf{b}(m, x)$ and receiving $\mathbf{r}(m, ?id)$. Indeed, because of the property of lists, v_i is the only node in the network which can possibly execute the broadcast from some state in Q' and, at the same time, having its register x set to v_{i+1} . At the same time, v_{i+1} is the only node in the network in some state in Q' with the reception rule possibly enabled, because the read-only register id uniquely identifies it. For the same reasons, m can be propagated backward from v_{i+1} to v_i by broadcasting $\mathbf{b}(m, id)$ and receiving $\mathbf{r}(m, ?x)$.

Lemma 4.1. For $r \geq 2$ and $f \geq 1$, $Cov(r, f)$ [resp. $Cov^b(r, f)$] restricted to initial configurations $\Gamma'_0 \subseteq \Gamma_0$ is undecidable if there exists a forward or backward list builder (r, f) -protocol for \Rightarrow [resp. \Rightarrow_b] and $\Gamma'_0 \subseteq \Gamma_0$ on $x \in [1..r]$ that can generate lists of arbitrary finite length.

Proof:

We show that, under the assumptions of the Lemma, the following reduction from the halting problem for two-counter machines to $Cov^b(r, f)$ is correct. Then, to prove the $Cov(r, f)$ case, we will show that the reduction also works with \Rightarrow . We provide the reduction only for the case of forward list builders: in case of backward ones it is sufficient to swap the patterns to communicate back and forth in the linked list. Indeed, the only change to be dealt with would be the direction of the links kept in register x of each node.

First we recall the definition of a two-counter machine; it is machine $\mathcal{M} = \langle Loc, Inst, \ell_0 \rangle$ where Loc is a finite set of location, $\ell_0 \in Loc$ is an initial location and $Inst$ is a finite set of instructions manipulating two variables c_1 and c_2 which take their value in the natural numbers (aka counters), each rule being of the following form: **increment of counter** c_i (ℓ, c_i++, ℓ'), **decrement of counter** c_i (ℓ, c_i--, ℓ') and **zero-test of counter** c_i ($\ell, c_i == 0, \ell'$) with $\ell, \ell' \in Loc$. In such a machine, the counters can never take negative values. We do not recall the semantics of such machine which is quite natural. The reachability problem for a two counter machine \mathcal{M} and a location $\ell \in Loc$ consists in determining whether such a machine starting in ℓ_0 with 0 as counter values can reach the location ℓ by executing the instructions. This problem is known to be undecidable [26].

Let $\mathcal{P}_{lb} = \langle Q_{lb}, R_{lb}, q_0 \rangle$ be a forward $\langle q_h, q_z, q_t \rangle$ -list builder for \Rightarrow_b and $\Gamma'_0 \subseteq \Gamma_0$ on $x \in [1..r]$ and with a read-only register $id \in [1..r]$, and let $\mathcal{M} = \langle Loc, Inst, \ell_0 \rangle$ be a two-counter machine. We extend \mathcal{P}_{lb} to obtain protocol $\mathcal{P} = \langle Q, R, q_0 \rangle$ as an encoding of \mathcal{M} . Each location $\ell \in Loc \setminus \{\ell_0\}$ and each instruction $i \in Inst$ are mapped respectively to a state $\mathcal{P}(\ell) \in Q \setminus Q_{lb}$ and to a set of new auxiliary states $q \in Q \setminus Q_{lb}$ and rules $r \in R \setminus R_{lb}$. The initial location ℓ_0 is mapped into $\mathcal{P}(\ell_0) = q_h$, because, by Definition 4.1, as soon as a node labelled q_h appears its corresponding list is ready. Counters are encoded in unary through individual processes: each process in state q_z represents a zero and it may change state

in order to represent a unit of one counter (q_{c_1}) or another (q_{c_2}). The encoded instructions work by propagating appropriate messages back and forth through the list, with the the tail node q_t serving as a terminator.

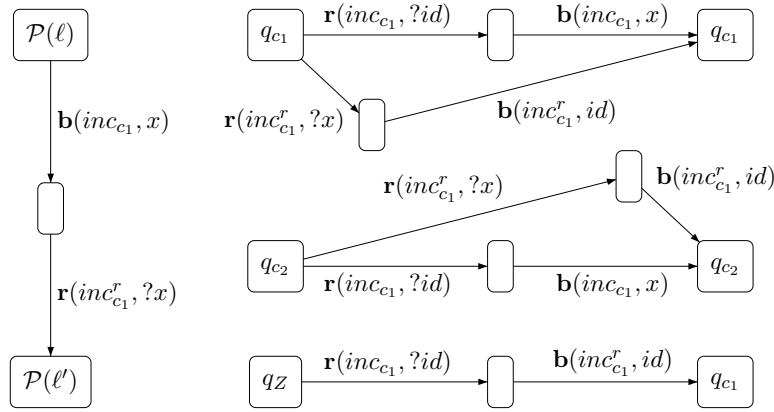


Figure 2. Increment of counter c_1

It is worth noting that, since \mathcal{P}_{lb} may build more than one list, at a given point we may have several ongoing simulations of \mathcal{M} . However, by following the point to point communication patterns previously described we ensure they are independent of each other. Figure 2 shows how increments $(\ell, c++, \ell') \in Inst$ are encoded. The head node, in state $\mathcal{P}(\ell)$, sends an increment order inc_c and waits for an acknowledgement reply inc_c^r before moving to the encoding of the next state, $\mathcal{P}(\ell')$. The message is propagated through the list, until it reaches the first process in state q_z , which goes to state q_c and replies back, or it reaches the tail q_t , which ignores it leading the head node to a deadlock (meaning the processes in the list were not enough to keep count of c_1 and c_2). A decrement instruction can be encoded by following the same pattern as for increments. Tests $(\ell, c == 0, \ell') \in Inst$ are encoded in a similar way, but in this case the reply with the acknowledgement tz_c^r can be sent only by the tail node q_t . The nodes in state q_c representing units of the currently tested counter do not propagate the message, therefore the message tz_c travels through the whole list and reaches the tail if and only if there are no q_c nodes, i.e. when $c = 0$.

When considering reconfigurations, i.e. when the transition relation is \Rightarrow , all of the previous assumptions still hold, except for the fact that, when sending a message from a node in the list to its successor, we no longer know if the two of them are neighbors. Otherwise said, with \Rightarrow we may lose messages, and the computation would block as soon as this happens. This is not a problem for the reduction however, because we know that an execution with reconfigurations such that no messages are lost still exists. Indeed, when encoding the reachability problem for \mathcal{M} and $\ell \in Loc$ with $Cov(r, f)$ for \mathcal{P} and $\mathcal{P}(\ell)$, blocked executions do not represent an obstacle, since the parameterized coverability problem is satisfied by the existence of an execution that leads to the target state. \square

The previous lemma tells us that to prove undecidability of the parameterized coverability problem we just have to exhibit a list-builder protocol. In the case of $Cov(2, 2)$, we apply Lemma 4.1, by showing that protocol \mathcal{P}_{lb} of Figure 4 is a backward list-builder for q_h, q_z , and q_t on $x \in [1..r]$. The rationale is as follows. Lists $\{v_1, \dots, v_k\}$ are built one node at a time, starting from the tail v_k , in state q_t . The links

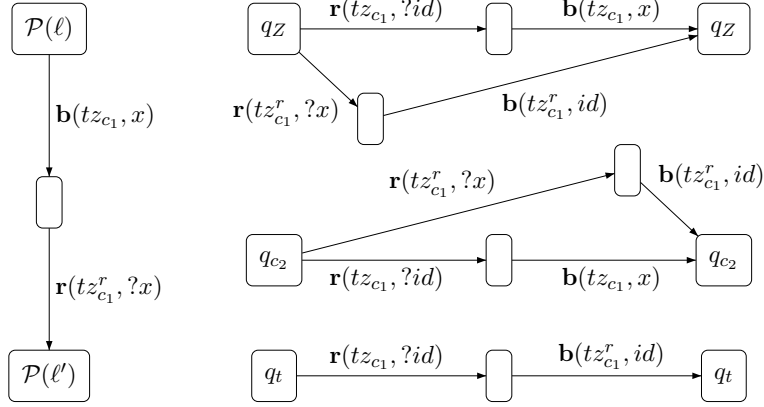


Figure 3. Testing for zero counter c_1

point from each node to the previous one, up to the head v_1 , in state q_h . Any node in the initial state q_0 (e.g., v_1) may decide to become a tail by starting to build its own list. Every such construction activity, however, is guaranteed not to interfere in any way with the others, thanks to point to point communication between nodes simulated on top of network reconfigurations and broadcast by exploiting the two payload fields. This is achieved via a three-way handshake where the first and second fields respectively identify the sender and the recipient. When the sub-protocol is done, v_1 moves to state q_t , v_2 moves to the intermediate state q_i , and one points to the other. Node v_2 decides whether to stop building the list by becoming the head q_h , or to continue by executing another handshake to elect node v_3 . The process continues until some v_k finally ends the construction by moving to state q_h . The following theorem then holds.

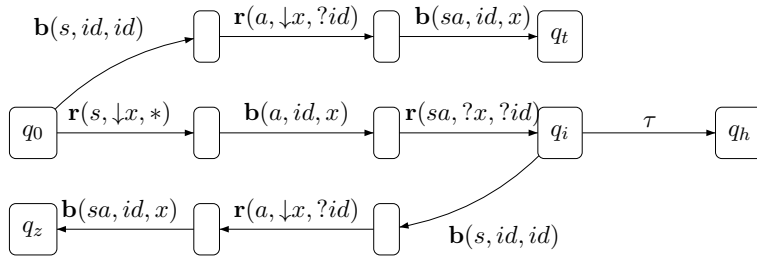


Figure 4. \mathcal{P}_{lb} : backward list-builder for q_h, q_z, q_t , and Γ_0 on x

Theorem 4.1. $Cov(2, 2)$ is undecidable even when restricting one register to be read-only.

Proof:

Let us consider protocol \mathcal{P}_{lb} of Figure 4. We now prove that \mathcal{P}_{lb} is a backward list builder for q_h, q_z, q_t and Γ_0 on $x \in [1..r]$.

Let $\gamma = \langle V, E, L \rangle \in \Gamma$ be a configuration. It is not fundamental that $\gamma \in \Gamma_0$, because the protocol may elect multiple lists. The only requirement is to have at least some nodes still in the initial state

q_0 , and this is trivially satisfied by every $\gamma_0 \in \Gamma_0$. A node $v_i \in V$ wishing to establish a connection with $v_{i+1} \in V$ broadcasts its identifier with a request $\mathbf{b}(s, id, id)$, either from q_0 or q_i (the paths from those two states to respectively q_t and q_z are labelled by the same actions). Its current neighbors in state q_0 store the identifier of v_i by firing $\mathbf{r}(s, \downarrow x, *)$. The first v_{i+1} of them that answers $\mathbf{b}(a, id, x)$ gets its own identifier stored by v_i with the reception rule $\mathbf{r}(a, \downarrow x, ?id)$ (provided reconfigurations did not disconnect it, otherwise the message is lost and the protocol stops). The winner, v_{i+1} , is notified by v_i with a confirmation message $\mathbf{b}(sa, id, x)$. Only v_{i+1} will be able to react to such a message, because it is the only node in the network for which the guard $?id$ in $\mathbf{r}(sa, ?x, ?id)$ is satisfied. At this point, node v_i which started the communication from q_0 or q_i is respectively in q_t or q_z . Node v_{i+1} is necessarily in the intermediate state q_i instead, as the (temporarily) latest elected node of the list. Its role is to choose whether to stop the construction via an internal transition to q_h , which would make it the head of the list, or to continue as previously described by choosing the path toward q_z . In the latter case, v_{i+1} becomes an intermediate node q_z and loses the pointer to v_i , which is overwritten because of the handshake with the next v_{i+2} . Nevertheless v_i will continue to point to v_{i+1} : the pointers of a completed list, therefore, go from q_t to q_h . With appropriate reconfigurations to keep only two nodes connected at a time, the protocol may build lists of arbitrarily length by involving all nodes in the network.

According to Definition 4.1, is indeed a backward list builder for q_h, q_z, q_t and Γ_0 on $x \in [1..r]$. By applying Lemma 4.1, we can finally conclude that $Cov(2, 2)$ is undecidable. \square

4.2. Decidability of $Cov(*, 1)$

In this section, we will prove that $Cov(*, 1)$, i.e. the restriction of our coverability problem to processes with only one field in the message, is PSPACE-complete.

4.2.1. Lower bound for $Cov(*, 1)$

We obtain PSPACE-hardness through a reduction from the reachability problem for 1-safe Petri nets.

Proposition 4.1. $Cov(*, 1)$ is PSPACE-hard.

Proof:

A Petri net N is a tuple $N = \langle P, T, \vec{m}_0 \rangle$, where: P is a finite set of places, T is a finite set of transitions t such that $\bullet t$ and $t \bullet$ are multisets of places (pre- and post-conditions of t), and \vec{m}_0 is a multiset of places that indicates how many tokens are located in each place in the initial net marking. Given a marking \vec{m} , the firing of a transition t such that $\bullet t \subseteq \vec{m}$ leads to a new marking \vec{m}' obtained as $\vec{m}' = \vec{m} \setminus \bullet t \cup t \bullet$. A Petri net P is 1-safe if in every reachable marking every place has at most one token. Reachability of a marking \vec{m}_1 from the initial marking \vec{m}_0 is decidable for Petri nets, and PSPACE-complete for 1-safe nets [8].

Given a 1-safe net $N = \langle P, T, \vec{m}_0 \rangle$ and a marking \vec{m}_1 , we encode the reachability problem into $Cov(|N|, 1)$. We will assume that $P = \{p_1, \dots, p_r\}$ and that p_1 is the unique place such that $\vec{m}_0(p_1) = 1$ and p_r the only place such that $\vec{m}_1(p_r) = 1$ (without loss of generality we can in fact reduce the reachability problem of 1-safe net into such a simple case). We now explain how to simulate the behavior of N with a $(r, 1)$ -protocol $\mathcal{P} = \langle Q, R, q_0 \rangle$. The protocol \mathcal{P} contains two control states *full* and *empty* from which the only possible action is the broadcast of a message containing the value stored in the first register. This is depicted in Figure 5. We see that nodes in state *empty* will always broadcast messages

of type α and nodes of type *full* will always broadcast messages of type β , and each of these messages contains the value of the first register which will never be overwritten (and hence will correspond to the identifier of the node). Then for each transition $t \in T$ with $\bullet t = \{p_{i_1}, \dots, p_{i_k}\}$ and $t^\bullet = \{p_{j_1}, \dots, p_{j_l}\}$

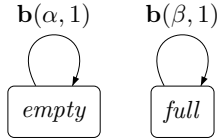


Figure 5. Encoding the nodes of type *full* and *empty*

(with $i_1, \dots, i_k, j_1, \dots, j_l \in [1..r]$), we will have in \mathcal{P} the transitions depicted in Figure 6. Basically, a node in state q_1 will first begin to test whether it has identifiers of nodes of type *full* in its register i_1 to i_k , then it will put in these registers identifiers of nodes of type *empty* to simulate the consumption of tokens in the associated places (by receiving messages of type α) and finally it will store identifiers of nodes of type *full* in its registers j_1 to j_l to simulate the production of tokens in the associated places. Finally, the Figure 7 shows how the simulation begin from the initial state q_0 , first nodes can go in states

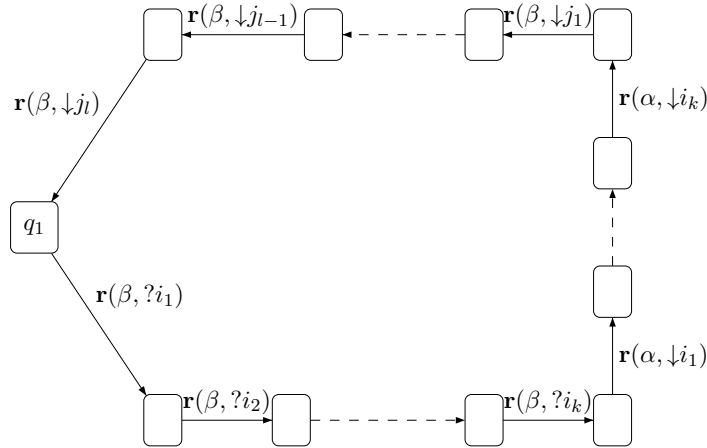


Figure 6. Encoding transition t with $\bullet t = \{p_{i_1}, \dots, p_{i_k}\}$ and $t^\bullet = \{p_{j_1}, \dots, p_{j_l}\}$

full or *empty* by broadcasting a message that no one will receive and then a node can go to state q_1 by receiving a message sent by a node *full* and it will store the identifier in the first register, this to simulate that the initial marking of N is the one with one token in p_1 . Finally, a node will go in state *end* if there is an identifier of a node of type *full* in the f -th register. One can then easily prove that the protocol \mathcal{P} verifies the property that \vec{m}_1 is reachable from \vec{m}_0 in N if and only if there exists $\gamma \in Reach(\mathcal{P})$ such that $\gamma \models end$. \square

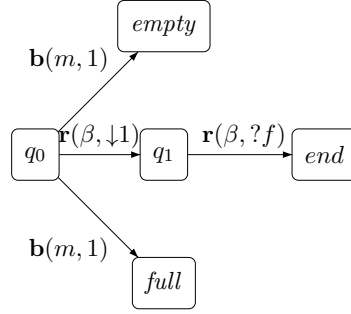


Figure 7. Initialization of the simulation and ending of the simulation of the 1-safe net

4.2.2. Upper bound for $Cov(*, 1)$

We now provide a PSPACE algorithm for solving $Cov(*, 1)$. The algorithm is based on a saturation procedure that computes a symbolic representation of reachable configurations. The representation is built using graphs that keep track of control states that may appear during a protocol execution and of relations between values in their registers. The set of symbolic configurations we consider is finite and each symbolic configuration can be encoded in polynomial space.

Symbolic configurations. Assume a $(r, 1)$ -protocol $\mathcal{P} = \langle Q, R, q_0 \rangle$ over Σ . A symbolic configuration θ for \mathcal{P} is a labelled graph $\langle W, \delta, \lambda \rangle$ where W is a set of nodes, $\delta \subseteq W \times [1..r] \times [1..r] \times W$ is the set of labelled edges and $\lambda : W \mapsto Q \times \{0, 1\}^r$ is a labeling function (as for configurations, we will denote λ_Q [resp. λ_M] the projection of λ to its first [resp. second] component) such that the following rules are respected:

- For $w, w' \in W$, $w \neq w'$ implies $\lambda_Q(w) \neq \lambda_Q(w')$, i.e. there cannot be two nodes with the same control state;
- If $(w, a, b, w') \in \delta$ then $\lambda_M(w)[a] = 1$ or $\lambda_M(w')[b] = 1$ (or both);
- For $w \in W$ and $j \in [1..r]$, if $\lambda_M(w)[j] = 1$ then $(w, j, j, w) \in \delta$.

The labels $\{0, 1\}^r$ are redundant (they can be derived from edges) but simplify some of the constructions needed in the algorithm. We denote by Θ the set of symbolic configurations for \mathcal{P} .

The intuition behind symbolic configuration is the following: a concrete configuration γ belongs to the denotation $\llbracket \theta \rrbracket$ of θ if for any node of γ there is a node in θ labelled with the same control states. Furthermore, for a pair of nodes v_1 and v_2 in γ containing the same value in registers a and b , respectively, θ must contain nodes labelled with the corresponding states and an edge labelled with (a, b) connecting them. Finally, if there are two nodes v in γ labelled with the state q and with the same value in register j , then there must be a node w in θ with state q and $\lambda_M(w)[j] = 1$.

We now formalize this intuition. Let $\theta = \langle W, \delta, \lambda \rangle$ be a symbolic configuration for \mathcal{P} . Then, $\langle V, E, L \rangle \in \llbracket \theta \rrbracket$ iff the following conditions are satisfied:

1. For each $v \in V$, there is a node $w \in W$ such that $L_Q(v) = \lambda_Q(w)$, i.e. v and w have the same control state;

2. For each $v \neq v' \in V$, if there exist registers $j, j' \in [1..r]$ s.t. $L_M(v)[j] = L_M(v')[j']$, i.e., two distinct nodes with the same value in a pair of registers, then there exists an edge $(w, j, j', w') \in \delta$ with $\lambda_Q(w) = L_Q(v)$ and $\lambda_Q(w') = L_Q(v')$, i.e. we store possible relations on data in registers using edges in θ ;
3. For each $v \in V$, if there exist $j \neq j' \in [1..r]$ s.t. $L_M(v)[j] = L_M(v)[j']$, i.e. a node with the same value in two distinct registers, then there exists a self loop $(w, j, j', w) \in \delta$ with $\lambda_Q(w) = L_Q(v)$.

We remark that we do not include any information on the communication links of γ , indeed reconfiguration steps can change the topology in an arbitrary way. We define the initial symbolic configuration $\theta_0 = \langle \{w_0\}, \emptyset, \lambda_0 \rangle$ with $\lambda_0(w_0) = (q_0, \vec{0})$. Clearly, we have $\llbracket \theta_0 \rrbracket = \Gamma_0$, i.e. the set of concrete configurations represented by θ_0 is the set of initial configurations of the protocol \mathcal{P} .

Computing symbolically the successors. In order to perform a symbolic reachability on symbolic configurations, we define a symbolic post operator $\text{POST}_{\mathcal{P}}$ that, by working on a symbolic configuration θ simulates the effect of the application of a broadcast rule on its instances $\llbracket \theta \rrbracket$. We illustrate the key points underlying its definition with the help of an example. Consider the symbolic configurations θ_1 and θ_2 in Figure 8, where we represent edges $(w, a, b, w') \in \delta$ with arrows from w to w' labelled by a, b . Please note that, even though we use directed edges for the graphical representation, the relation between nodes in W symmetrical as $(w, a, b, w') \in \delta$ is equivalent to (w', b, a, w) .

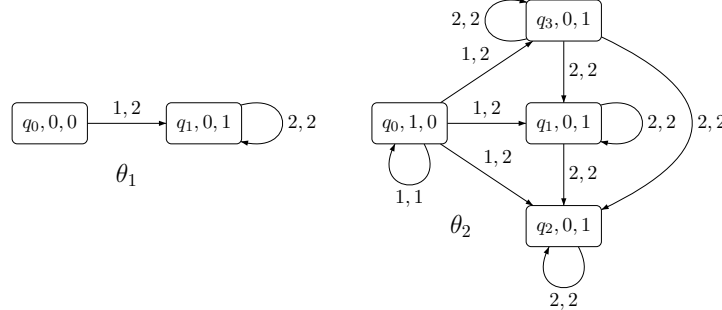


Figure 8. Example of computations of symbolic post

θ_1 denotes configurations with any number of nodes with label q_0 or q_1 . Nodes in state q_0 must have registers containing distinct data (label 0, 0). Nodes in state q_1 may have the same value in their second register (label 0, 1 is equivalent to edge $\langle q_1, 2, 2, q_1 \rangle$), that in turn may be equal to the value of the first register in a node labelled q_0 (edge $\langle q_0, 1, 2, q_1 \rangle$). θ_1 can be obtained from the initial symbolic configuration by applying rules like $\langle q_0, \mathbf{b}(\alpha, 1), q_0 \rangle$ and $\langle q_0, \mathbf{r}(\alpha, \downarrow 2), q_1 \rangle$. Indeed, in q_0 we can send the value of the first register to other nodes in q_0 that can then move to q_1 and store the data in the second register (i.e. we create a potential data relation between the first and second register).

We now give examples of rules that can generate the symbolic configuration θ_2 starting from θ_1 . The pair $\langle q_0, \mathbf{b}(\beta, 1), q_0 \rangle$ and $\langle q_0, \mathbf{r}(\beta, \downarrow 1), q_0 \rangle$ generates a new data relation between nodes in state q_0 modelled by changing from 0 to 1 the value of $\lambda_M(q_0)[1]$. We remark that a label 1 only says that registers in distinct nodes may be (but not necessarily) equal.

Consider now the reception rule $\langle q_1, \mathbf{r}(\beta, ?2), q_2 \rangle$ for the same message β . The data relation between nodes in state q_0 and q_1 in θ_1 tells us that the rule is fireable. To model its effect we need to create a new

node with label q_2 with data relations between registers expressed by the edges between labels q_0, q_1 and q_2 in the figure. Due to possible reconfigurations, not all nodes in q_1 necessarily react, i.e. θ_2 contains the denotations of θ_1 .

A rule like $\langle q_1, \mathbf{r}(\beta, ?\bar{2}), q_3 \rangle$ can also be fireable from instances of θ_1 . Indeed, the message β can be sent by a node in state q_0 that does not satisfy the data relation specified by the edge (1, 2) in θ_1 , i.e., the sending node is not the one having the same value in its first register as the node q_1 reacting to the message, hence the guard $?\bar{2}$ could also be satisfied. This leads to a new node with state q_3 which inherits from q_1 the constraints on the first register, but whose second register can have the same value as the second register of nodes in any state.

We will now provide the formal definition of this symbolic post operator $\text{POST}_{\mathcal{P}} : \Theta \mapsto \Theta$ that takes as input a symbolic configuration and compute a symbolic configuration characterizing the successor of all the configurations represented by the input symbolic configuration following the rules of \mathcal{P} . Before giving the formal definition, we need to introduce another notion over symbolic configurations. Given two symbolic configurations $\theta = \langle W, \delta, \lambda \rangle$ and $\theta' = \langle W', \delta', \lambda' \rangle$, we define the union of symbolic configurations θ and θ' , denoted $\theta \sqcup \theta'$, as follows: $\langle W'', \delta'', \lambda'' \rangle = \theta \sqcup \theta'$ iff the following conditions are respected:

- there exist $w'' \in W''$ with $\lambda_Q(w'') = q$ iff there exists $w \in W$ with $\lambda_Q(w) = q$ or there exists $w' \in W'$ with $\lambda_Q(w) = \lambda_Q''(w'')$ and furthermore for all $i \in [1..r]$, $\lambda_M(w'')[i] = 1$ if and only if $\lambda_M(w)[i] = 1$ or $\lambda_M(w')[i] = 1$.
- there exists $(w''_1, a, b, w''_2) \in \delta''$ with $\lambda_Q(w''_1) = q_1$ and $\lambda_Q(w''_2) = q_2$ iff there exists $(w_1, a, b, w_2) \in \delta$ (with $\lambda_Q(w_1) = q_1$) and $\lambda_Q(w_2) = q_2$ or there exists $(w'_1, a, b, w'_2) \in \delta'$ (with $\lambda_Q'(w'_1) = q_1$ and $\lambda_Q(w'_2) = q_2$)

The idea is that to build $\theta \sqcup \theta'$, we put all the labels present in the symbolic configuration in the result symbolic configuration and each time we encounter a 1 in a label, it is reported in the union and all the edges of the two configurations are reported in the union. We have then the following result which makes the link between the symbolic union and the union on the corresponding concrete configurations.

Lemma 4.2. Let θ, θ' be two symbolic configurations. We have $\llbracket \theta \rrbracket \cup \llbracket \theta' \rrbracket \subseteq \llbracket \theta \sqcup \theta' \rrbracket$.

Proof:

Assume $\theta = \langle W, \delta, \lambda \rangle$ and $\theta' = \langle W', \delta', \lambda' \rangle$. Let $\gamma = \langle V, E, L \rangle$ be in $\llbracket \theta \rrbracket \cup \llbracket \theta' \rrbracket$. We suppose $\gamma \in \llbracket \theta \rrbracket$ (the case $\gamma \in \llbracket \theta' \rrbracket$ can be treated similarly). Let $\langle W'', \delta'', \lambda'' \rangle = \theta \sqcup \theta'$. We will show that $\gamma \in \llbracket \langle W'', \delta'', \lambda'' \rangle \rrbracket$. We verify each point of the definition of $\llbracket \cdot \rrbracket$.

1. Let $v \in V$, since $\gamma \in \llbracket \theta \rrbracket$, there exists $w \in W$ such that $L_Q(v) = \lambda_Q(w)$, by definition of \sqcup , there exists $w'' \in W''$ such that $\lambda_Q''(w'') = \lambda_Q(w) = L_Q(v)$.
2. Let $v, v' \in V$ with $v \neq v'$ and let $j, j' \in [1..r]$. Assume $L_M(v)[j] = L_M(v')[j']$. Then there exists $(w, j, j', w') \in \delta$ with $\lambda_Q(w) = L_Q(v)$ and $\lambda_Q(w') = L_Q(v')$ and by definition of \sqcup there exists $(w''_1, j, j', w''_2) \in \delta''$ with $\lambda_Q(w''_1) = \lambda_Q(w) = L_Q(v)$ and $\lambda_Q(w''_2) = \lambda_Q(w') = L_Q(v')$.
3. Let $v \in V$ and $j, j' \in [1..r]$ with $j \neq j'$ such that $\lambda_M(v)[j] = \lambda_M(v)[j']$ then there exists $(w, j, j', w) \in \delta$ with $\lambda_Q(w) = L_Q(v)$ and by definition of \sqcup there exists $(w'', j, j', w'') \in \delta''$ with $\lambda_Q(w'') = \lambda_Q(w) = L_Q(v)$.

This allows us to conclude that $\gamma \in [\theta \sqcup \theta']$. □

Algorithm 1 gives the formal definition of the function $\text{POST}_{\mathcal{P}} : (Q \times \text{Send}_{\Sigma}^{r,1} \times Q) \times \Theta \mapsto \Theta$ which take as input a broadcast rule and a symbolic configuration and compute the effect of the broadcast on the configurations by considering the different receptions of the protocol \mathcal{P} . The operator $\text{POST}_{\mathcal{P}} : \Theta \mapsto \Theta$ is then simply the symbolic union of the possible symbolic configurations obtained by applying all the broadcast rules of \mathcal{P} . More formally if $\mathcal{P} = \langle Q, R, q_0 \rangle$, then for all symbolic configurations θ we have $\text{POST}_{\mathcal{P}}(\theta) = \bigsqcup_{\langle q, \mathbf{b}(m,p), q' \rangle \in R} \text{POST}_{\mathcal{P}}(\langle q, \mathbf{b}(m,p), q' \rangle, \theta)$.

Before giving the properties of the $\text{POST}_{\mathcal{P}}$ operator, we introduce some notations. First we introduce an order on symbolic configurations. Given two symbolic configurations $\theta = \langle W, \delta, \lambda \rangle$ and $\theta' = \langle W', \delta', \lambda' \rangle$, we say that $\theta \sqsubseteq \theta'$ if and only if there exists an injective function $h : W \mapsto W'$ such that for all $w, w' \in W$:

- $\lambda_Q(w) = \lambda'_Q(h(w))$;
- for all $j \in [1..r]$, if $\lambda_M(w)[j] = 1$ then $\lambda'_M(h(w))[j] = 1$;
- if $(w, a, b, w') \in \delta$ then $(h(w), a, b, h(w')) \in \delta'$.

In other words, we have $\theta \sqsubseteq \theta'$ if there are more nodes in θ' than in θ and all the labels of θ appears in θ' as well, and for what concerns the symbolic register valuation, the one of θ' should "cover" the one of θ , i.e. there are more 1 in θ' than in θ . In the sequel, we will say that two symbolic configurations are equal if they are equal up to isomorphism. Since the number of symbolic configurations which are pairwise disjoint is finite (because there is at most $|Q|$ nodes in a symbolic configuration), and by definition of the $\llbracket \cdot \rrbracket$ operator and of \sqsubseteq , one can easily prove the following result.

Lemma 4.3. (1) If $\theta \sqsubseteq \theta'$ then $\llbracket \theta \rrbracket \subseteq \llbracket \theta' \rrbracket$. (2) If there exists an infinite increasing sequence $\theta_0 \sqsubseteq \theta_1 \sqsubseteq \theta_2 \dots$ then there exists $i \in \mathbb{N}$ s.t. for all $j \geq i$, $\theta_j = \theta_i$.

Using the definition of \sqsubseteq , we can state our first property saying any symbolic configurations is symbolically included in its symbolic successor.

Lemma 4.4. For all symbolic configurations θ , we have $\theta \sqsubseteq \text{POST}_{\mathcal{P}}(\theta)$.

Sketch of proof. Recall that $\text{POST}_{\mathcal{P}}(\theta) = \bigsqcup_{\langle q, \mathbf{b}(m,p), q' \rangle \in R} \text{POST}_{\mathcal{P}}(\langle q, \mathbf{b}(m,p), q' \rangle, \theta)$. If we look carefully at Algorithm 1, we notice that it only changes 0 to 1 in the label of the nodes of the input symbolic configuration θ or it adds new edges or new nodes. Hence by definition of the relation \sqsubseteq , we have for all rules $\langle q, \mathbf{b}(m,p), q' \rangle \in R$, $\theta \sqsubseteq \text{POST}_{\mathcal{P}}(\langle q, \mathbf{b}(m,p), q' \rangle, \theta)$ and using the definition of the operator \sqcup that also do not delete any edges from the symbolic configurations given in input, we deduce that $\theta \sqsubseteq \text{POST}_{\mathcal{P}}(\theta)$. □

One consequence of these two lemmas is that if we denote $\text{POST}_{\mathcal{P}}^i$ the function which consists in applying i times $\text{POST}_{\mathcal{P}}$, then we now that for all symbolic configurations θ , there exists an integer K such that for all $i \geq K$ we have $\text{POST}_{\mathcal{P}}^i(\theta) = \text{POST}_{\mathcal{P}}^K(\theta)$. We denote in the sequel $\text{POST}_{\mathcal{P}}^*(\theta)$ the symbolic configuration $\text{POST}_{\mathcal{P}}^K(\theta)$. Note that each symbolic configuration of the $(r, 1)$ -protocol \mathcal{P} is a graph with at most $|Q|$ nodes and at most $|Q|^2 * |r|^2$ edges and hence we need only polynomial space in the size of the protocol \mathcal{P} to compute $\text{POST}_{\mathcal{P}}^*(\theta)$ for a symbolic configuration θ .

Algorithm 1 $\theta' = \text{POST}_{\mathcal{P}}(\langle q, \mathbf{b}(m, p), q' \rangle, \theta)$

Require: A broadcast rule $\langle q, \mathbf{b}(m, p), q' \rangle \in R$ and $\theta = \langle W, \delta, \lambda \rangle$ a symbolic configuration of the $(r, 1)$ -protocol $\mathcal{P} = \langle Q, R, q_0 \rangle$

Ensure: $\theta' = \langle W', \delta', \lambda' \rangle$

- 1: $W' := W, \delta' := \delta, \lambda' := \lambda$
 - 2: **if** there exists $w \in W$ such that $\lambda_Q(w) = q$ **then**
 - 3: **if** there does not exist $w' \in W'$ such that $\lambda'_Q(w') = q'$ **then**
 - 4: Create a node $w' \in W'$ with $\lambda'_Q(w') = q'$ and $\lambda'_M(w') := \vec{0}$
 - 5: **end if**
 - 6: Let $w' \in W'$ such that $\lambda'_Q(w') = q'$
 - 7: for all $j \in [1..r]$, if $\lambda_M(w)[j] = 1$ then $\lambda'_M(w')[j] := 1$
 - 8: for all $(w, a, b, w'') \in \delta', \delta' := \delta' \cup \{(w', a, b, w'')\}$
 - 9: for all $(w, a, b, w) \in \delta', \delta' := \delta' \cup \{(w', a, b, w')\}$
 - 10: **for** all $\langle q'', \mathbf{r}(m, \alpha), q''' \rangle \in R$ such that there exists $w'' \in W$ with $\lambda_Q(w'') = q''$ **do**
 - 11: **if** $\alpha = ?\bar{k}$ or $\alpha = *$ **then**
 - 12: **if** there does not exist $w''' \in W'$ such that $\lambda'_Q(w''') = q'''$ **then**
 - 13: Create a node $w''' \in W'$ with $\lambda'_Q(w''') = q'''$ and $\lambda'_M(w''') := \vec{0}$
 - 14: **end if**
 - 15: Let $w''' \in W'$ such that $\lambda'_Q(w''') = q'''$
 - 16: for all $j \in [1..r]$, if $\lambda_M(w'')[j] = 1$ then $\lambda'_M(w''')[j] := 1$
 - 17: for all $(w'', a, b, v) \in \delta', \delta' := \delta' \cup \{(w''', a, b, v)\}$
 - 18: for all $(w'', a, b, w'') \in \delta', \delta' := \delta' \cup \{(w''', a, b, w'')\}$
 - 19: **end if**
 - 20: **if** $\alpha = \downarrow k$ **then**
 - 21: **if** there does not exist $w''' \in W'$ such that $\lambda'_Q(w''') = q'''$ **then**
 - 22: Create a node $w''' \in W'$ with $\lambda'_Q(w''') = q'''$ and $\lambda'_M(w''') := \vec{0}$
 - 23: **end if**
 - 24: Let $w''' \in W'$ such that $\lambda'_Q(w''') = q'''$
 - 25: $\lambda_M(w''')[k] = 1$ and $\delta' := \delta' \cup \{(w', p, k, w'''), (w''', k, k, w''')\}$
 - 26: For all $j \in [1..r]$, if $\lambda_M(w'')[j] = 1$ then $\lambda'_M(w''')[j] := 1$
 - 27: if $\lambda_M(w)[p] = 1$, Then for all $(w, p, b, v) \in \delta, \delta' := \delta' \cup \{(w''', k, b, v)\}$
 - 28: for all $(w'', a, b, v) \in \delta'$ with $a \neq k$, $\delta' := \delta' \cup \{(w''', a, b, v)\}$
 - 29: For all $(w'', a, b, w'') \in \delta'$ with $a \neq k$ and $b \neq k$, $\delta' := \delta' \cup \{(w''', a, b, w'')\}$
 - 30: **end if**
 - 31: **if** $\alpha = ?k$ and $(w, p, k, w'') \in \delta$ or $(w'', k, p, w) \in \delta$ **then**
 - 32: **if** there does not exist $w''' \in W'$ such that $\lambda'_Q(w''') = q'''$ **then**
 - 33: Create a node $w''' \in W'$ with $\lambda'_Q(w''') = q'''$ and $\lambda'_M(w''') := \vec{0}$
 - 34: **end if**
 - 35: Let $w''' \in W'$ such that $\lambda'_Q(w''') = q'''$
 - 36: for all $j \in [1..r]$, if $\lambda_M(w'')[j] = 1$ then $\lambda'_M(w''')[j] := 1$
 - 37: for all $(w'', a, b, v) \in \delta', \delta' := \delta' \cup \{(w''', a, b, v)\}$
 - 38: for all $(w'', a, b, w'') \in \delta', \delta' := \delta' \cup \{(w''', a, b, w'')\}$
 - 39: **end if**
 - 40: **end for**
 - 41: **end if**
-

Lemma 4.5. Given a symbolic configuration, $\text{POST}_{\mathcal{P}}^*(\theta)$ can be computed in polynomial space in the size of \mathcal{P} .

Given a set of configurations $S \subseteq \Gamma$ of the $(r, 1)$ -protocol $\mathcal{P} = \langle Q, R, q_0 \rangle$ (with $\text{BNRA}(\mathcal{P}) = \langle \Gamma, \Rightarrow, \Gamma_0 \rangle$), we define $\text{post}_{\mathcal{P}}(S) = \{\gamma' \in \Gamma \mid \exists \gamma \in S \text{ s.t. } \gamma \Rightarrow \gamma'\}$ and $\text{post}_{\mathcal{P}}^*$ is the reflexive and transitive closure of $\text{post}_{\mathcal{P}}$. We will now see how to relate $\text{POST}_{\mathcal{P}}$ and $\text{post}_{\mathcal{P}}$.

Lemma 4.6. For all symbolic configuration θ , we have $\text{post}_{\mathcal{P}}(\llbracket \theta \rrbracket) \subseteq \llbracket \text{POST}_{\mathcal{P}}(\theta) \rrbracket$.

Sketch of proof. We consider a symbolic configuration θ . We recall that by definition $\text{POST}_{\mathcal{P}}(\theta) = \bigsqcup_{\langle q, \mathbf{b}(m, p), q' \rangle \in R} \text{POST}_{\mathcal{P}}(\langle q, \mathbf{b}(m, p), q' \rangle, \theta)$. We consider a broadcast rule $\langle q, \mathbf{b}(m, p), q' \rangle$ and we denote by $\text{post}_{\mathcal{P}}(\langle q, \mathbf{b}(m, p), q' \rangle, \llbracket \theta \rrbracket)$ the set of configurations γ' that can be obtained by performing a **Broadcast** step in $\text{BNRA}(\mathcal{P})$ from a configuration γ in $\llbracket \theta \rrbracket$ using the broadcast rule $\langle q, \mathbf{b}(m, p), q' \rangle$. By performing a case analysis on the different reception rules of \mathcal{P} , one can show that $\text{post}_{\mathcal{P}}(\langle q, \mathbf{b}(m, p), q' \rangle, \llbracket \theta \rrbracket) \subseteq \llbracket \text{POST}_{\mathcal{P}}(\langle q, \mathbf{b}(m, p), q' \rangle, \theta) \rrbracket$. We have hence:

$$\bigcup_{\langle q, \mathbf{b}(m, p), q' \rangle \in R} \text{post}_{\mathcal{P}}(\langle q, \mathbf{b}(m, p), q' \rangle, \llbracket \theta \rrbracket) \subseteq \bigcup_{\langle q, \mathbf{b}(m, p), q' \rangle \in R} \llbracket \text{POST}_{\mathcal{P}}(\langle q, \mathbf{b}(m, p), q' \rangle, \theta) \rrbracket$$

Thanks to Lemma 4.2, we can deduce that

$$\bigcup_{\langle q, \mathbf{b}(m, p), q' \rangle \in R} \llbracket \text{POST}_{\mathcal{P}}(\langle q, \mathbf{b}(m, p), q' \rangle, \theta) \rrbracket \subseteq \llbracket \bigsqcup_{\langle q, \mathbf{b}(m, p), q' \rangle \in R} \text{POST}_{\mathcal{P}}(\langle q, \mathbf{b}(m, p), q' \rangle, \theta) \rrbracket$$

and hence we have:

$$\bigcup_{\langle q, \mathbf{b}(m, p), q' \rangle \in R} \text{post}_{\mathcal{P}}(\langle q, \mathbf{b}(m, p), q' \rangle, \llbracket \theta \rrbracket) \subseteq \llbracket \text{POST}_{\mathcal{P}}(\theta) \rrbracket$$

Furthermore note that for all $\gamma \in \llbracket \theta \rrbracket$ and all configurations γ' , if $\gamma \Rightarrow \gamma'$ and the applied rule is a **Reconfiguration**, then we have $\gamma' \in \llbracket \theta \rrbracket$, hence using Lemma 4.4 and the first item of Lemma 4.3, we deduce that $\gamma' \in \llbracket \text{POST}_{\mathcal{P}}(\theta) \rrbracket$. Consequently we can conclude that $\text{post}_{\mathcal{P}}(\llbracket \theta \rrbracket) \subseteq \llbracket \text{POST}_{\mathcal{P}}(\theta) \rrbracket$. \square

From this lemma, we can deduce by an easy induction the following lemma.

Corollary 4.1. For all symbolic configurations θ , we have $\text{post}_{\mathcal{P}}^*(\llbracket \theta \rrbracket) \subseteq \llbracket \text{POST}_{\mathcal{P}}^*(\theta) \rrbracket$.

Ideally, we would like to have that for any symbolic configuration the set $\llbracket \text{POST}_{\mathcal{P}}(\theta) \rrbracket$ is included into $\text{post}_{\mathcal{P}}^*(\llbracket \theta \rrbracket)$ because we perform many broadcast in a symbolic step during the computation of the symbolic post. Unfortunately this is not the case. In fact, consider the symbolic configuration θ_1 depicted in Figure 8. As we explained this symbolic configuration could be equal to $\text{POST}_{\mathcal{P}}(\theta_0)$ (where θ_0 is the initial symbolic configuration containing a single node labelled by $\langle q_0, \vec{0} \rangle$) by considering the rules in \mathcal{P} of the form: $\langle q_0, \mathbf{b}(\alpha, 1), q_0 \rangle$ and $\langle q_0, \mathbf{r}(\alpha, \downarrow 2), q_1 \rangle$. Note that the configuration γ containing two nodes u and v such that $L_Q(u) = q_0$, $L_M(u) = \langle 0, 1 \rangle$, $L_Q(v) = q_1$ and $L_M(v) = \langle 2, 3 \rangle$ belongs to $\llbracket \theta_1 \rrbracket$ but is not reachable thanks to the according rules, in fact when one node goes from q_0 to q_1 it should share in its second register a value with a node labelled by q_0 . Hence we have $\gamma \in \llbracket \text{POST}_{\mathcal{P}}(\theta_0) \rrbracket$ and $\gamma \notin \text{post}_{\mathcal{P}}^*(\llbracket \theta_0 \rrbracket)$. However we can "increase" the configuration γ to obtain a bigger configuration γ'

reachable from an initial configuration and belonging as well to $\llbracket \theta_1 \rrbracket$, for instance by adding a node u' to γ such $L_Q(u') = q_0$ and $L_M(u') = \langle 3, 4 \rangle$. We now formalize this idea.

We introduce an ordering relation $\trianglelefteq \subseteq \Gamma \times \Gamma$ on concrete configurations defined as follows: given two configurations $\gamma = \langle V, E, L \rangle$ and $\gamma' = \langle V', E', L' \rangle$, we have $\gamma \trianglelefteq \gamma'$ iff there exists an injective function $h : V \mapsto V'$ such that:

- for all $v \in V$, $L_Q(v) = L_Q(h(v))$;
- for all $v, v' \in V$ and all $i, j \in [1..r]$, $L_M(v)[i] = L_M(v')[j]$ if and only if $L_M(h(v))[i] = L_M(h(v'))[j]$.

Note that in the previous example, we have effectively $\gamma \trianglelefteq \gamma'$. By using the definition of the transition relation \Rightarrow and of the satisfaction relation \models for reachability query we have the following lemma.

Lemma 4.7. Let $\gamma_1, \gamma_2 \in \Gamma$ such that $\gamma_1 \trianglelefteq \gamma_2$. We have then:

1. For all $\gamma'_1 \in \Gamma$ such that $\gamma_1 \Rightarrow \gamma'_1$, there exists $\gamma'_2 \in \Gamma$ such that $\gamma_2 \Rightarrow \gamma'_2$ and $\gamma'_1 \trianglelefteq \gamma'_2$.
2. For all reachability queries φ , if $\gamma_1 \models \varphi$ then $\gamma_2 \models \varphi$.

Furthermore, as shown with the previous example, by using the definition of $\llbracket \cdot \rrbracket$ over symbolic configurations and the definition of the symbolic post operator $\text{POST}_{\mathcal{P}}$, we can deduce the following result.

Lemma 4.8. Let θ be a symbolic configuration. For all $\gamma \in \llbracket \text{POST}_{\mathcal{P}}(\theta) \rrbracket$, there exists $\gamma' \in \text{post}_{\mathcal{P}}^*(\llbracket \theta \rrbracket) \cap \llbracket \text{POST}_{\mathcal{P}}(\theta) \rrbracket$ such that $\gamma \trianglelefteq \gamma'$.

These two last lemmas allow us to obtain the following corollary.

Corollary 4.2. Let θ be a symbolic configuration. For all $\gamma \in \llbracket \text{POST}_{\mathcal{P}}^*(\theta) \rrbracket$, there exists $\gamma' \in \text{post}_{\mathcal{P}}^*(\llbracket \theta \rrbracket)$ such that $\gamma \trianglelefteq \gamma'$.

Proof:

We will prove that for all $n \in \mathbb{N}$, for all $\gamma \in \llbracket \text{POST}_{\mathcal{P}}^n(\theta) \rrbracket$, there exists $\gamma' \in \text{post}_{\mathcal{P}}^*(\llbracket \theta \rrbracket)$ such that $\gamma \trianglelefteq \gamma'$. We reason by induction on n . First in the case $n = 0$, the property holds trivially (we recall that $\text{POST}_{\mathcal{P}}^0(\theta) = \theta$). Now assume the property holds for $n \in \mathbb{N}$ and we will show it is still true for $n + 1$. Let $\gamma \in \llbracket \text{POST}_{\mathcal{P}}^{n+1}(\theta) \rrbracket$. From Lemma 4.8, we deduce that there exists $\gamma' \in \text{post}_{\mathcal{P}}^*(\llbracket \text{POST}_{\mathcal{P}}^n(\theta) \rrbracket) \cap \llbracket \text{POST}_{\mathcal{P}}^{n+1}(\theta) \rrbracket$ such that $\gamma \trianglelefteq \gamma'$. Hence there exists $\gamma_1 \in \llbracket \text{POST}_{\mathcal{P}}^n(\theta) \rrbracket$ such that $\gamma_1 \Rightarrow \dots \Rightarrow \gamma'$. By induction hypothesis, there exist $\gamma_2 \in \text{post}_{\mathcal{P}}^*(\llbracket \theta \rrbracket)$ such that $\gamma_1 \trianglelefteq \gamma_2$. But then thanks the first item of Lemma 4.7, since $\gamma_1 \trianglelefteq \gamma_2$ and $\gamma_1 \Rightarrow \dots \Rightarrow \gamma'$ we deduce that there exists γ'_2 such that $\gamma_2 \Rightarrow \dots \Rightarrow \gamma'_2$ and $\gamma' \trianglelefteq \gamma'_2$. Note that since $\gamma_2 \in \text{post}_{\mathcal{P}}^*(\llbracket \theta \rrbracket)$, then $\gamma'_2 \in \text{post}_{\mathcal{P}}^*(\llbracket \theta \rrbracket)$ and since $\gamma \trianglelefteq \gamma'$ and $\gamma' \trianglelefteq \gamma'_2$, we also have $\gamma \trianglelefteq \gamma'_2$. \square

Evaluating a reachability query symbolically. We now define how to evaluate a reachability query over a symbolic configuration. Let $\theta = \langle W, \delta, \lambda \rangle$ be a symbolic configuration and φ be a reachability query. We denote by $\text{Vars}(\varphi)$ the subset of variables used in the query φ and we assume that $\varphi = \bigwedge_{k \in [1..m]} \varphi_k$ where for each $k \in [1..m]$, φ_k is of the form $q(\mathbf{z})$ or $M_i(\mathbf{z}) = M_j(\mathbf{z}')$ or $M_i(\mathbf{z}) \neq M_j(\mathbf{z}')$. We will then say that $\theta \models \varphi$ if there exists a function $g : \text{Vars}(\varphi) \mapsto W$ such that for all $k \in [1..m]$ we have the following properties: if $\varphi_k = q(\mathbf{z})$, then $\lambda_Q(g(\mathbf{z})) = q$; if $\varphi_k = (M_i(\mathbf{z}) = M_j(\mathbf{z}'))$ with $\mathbf{z} \neq \mathbf{z}'$ or $i \neq j$, then $(g(\mathbf{z}), i, j, g(\mathbf{z}')) \in \delta$. We have then the following lemma.

Lemma 4.9. Given a symbolic configuration θ and a reachability query φ , we have $\theta \models \varphi$ if and only if there exists $\gamma \in \llbracket \theta \rrbracket$ such that $\gamma \models \varphi$.

We can now state the main result about the symbolic post operator.

Lemma 4.10. Let θ be a symbolic configuration of the protocol \mathcal{P} . Then we have for all reachability query φ , there exists $\gamma \in \text{post}_{\mathcal{P}}^*(\llbracket \theta \rrbracket)$ such that $\gamma \models \varphi$ iff $\text{POST}_{\mathcal{P}}^*(\theta) \models \varphi$.

Proof:

Let θ be a symbolic configuration of the protocol \mathcal{P} . Let φ be a reachability query. First assume that there exists $\gamma \in \text{post}_{\mathcal{P}}^*(\llbracket \theta \rrbracket)$ such that $\gamma \models \varphi$, then by Corollary 4.1 we know that $\gamma \in \text{POST}_{\mathcal{P}}^*(\theta)$. By Lemma 4.9, we deduce that $\text{POST}_{\mathcal{P}}^*(\theta) \models \varphi$. Assume now that $\text{POST}_{\mathcal{P}}^*(\theta) \models \varphi$. By Lemma 4.9, there exists $\gamma \in \llbracket \text{POST}_{\mathcal{P}}^*(\theta) \rrbracket$ such that $\gamma \models \varphi$. By Corollary 4.2, there exists $\gamma' \in \text{post}_{\mathcal{P}}^*(\llbracket \theta \rrbracket)$ such that $\gamma \preceq \gamma'$. Using the second item of Lemma 4.7, we obtain that $\gamma' \models \varphi$. \square

We have consequently an algorithm to solve whether there exists $\gamma \in \text{Reach}(\mathcal{P}) = \text{post}_{\mathcal{P}}^*(\Gamma_0)$. In fact it is enough to compute $\text{POST}_{\mathcal{P}}^*(\theta_0)$ and to check whether $\text{POST}_{\mathcal{P}}^*(\theta) \models \varphi$. This computation is feasible in polynomial space thanks to Lemma 4.5. Finally we can check in non-deterministic linear time whether $\text{POST}_{\mathcal{P}}^*(\theta_0) \models \varphi$ (it is enough to guess the function g from $\text{Vars}(\varphi)$ to the nodes of $\text{POST}_{\mathcal{P}}^*(\theta_0)$). Using Lemma 4.10, this gives us a polynomial space procedure to check whether there exists $\gamma \in \text{Reach}(\mathcal{P})$ such that $\gamma \models \varphi$. Furthermore, thanks to the lower bound given by Proposition 4.1, we can deduce the exact complexity of coverability for protocols using a single field in their messages.

Theorem 4.2. $\text{Cov}(*, 1)$ is PSPACE-complete.

5. Fully Connected Topologies and No Reconfiguration

5.1. Undecidability of $\text{Cov}^{fc}(2, 1)$

We now move to coverability in fully connected topologies. In contrast with the results obtained without identifiers in [15] it turns out that, without reconfiguration, coverability is undecidable already in the case of nodes with two registers and one payload field. We define a (forward) list-builder protocol, which builds lists backwards from the tail q_t . At each step, a node v among the ones which are not part of the list broadcasts its identifier to the others (which store the value, thus pointing to v), and moves to q_z (or q_t , if it is the first step) electing itself as the next node in the list. The construction ends when such a node will instead move to q_h and force everyone else to stop. By applying Lemma 4.1, the following theorem then holds.

Theorem 5.1. $\text{Cov}^{fc}(2, 1)$ is undecidable even when one register is read-only.

Proof:

We now show that the protocol \mathcal{P}_{lb}^{fc} in Figure 9 is a list builder for \Rightarrow_b and Γ_0^{fc} on x and states q_h, q_z, q_t (where the lists are built backwards from q_t). Let $\gamma_0 = \langle V, E, L \rangle \in \Gamma_0^{fc}$ be an initial configuration. As soon as a node $v \in V$ decides to start the construction of the list, it broadcasts its identifier to every other node with a message $\mathbf{b}(\text{tail}, id)$. Since the network is fully connected, every process has to react to the

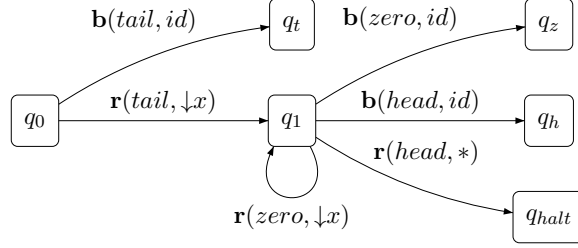


Figure 9. \mathcal{P}_{lb}^{fc} : list builder for $\langle q_h, q_z, q_t \rangle$ and fully connected configurations on x

message: after the transition we get a configuration $\gamma_1 \in \Gamma$ such that v is the only node in state q_t while any other node $u \in V \setminus \{v\}$ is in state q_1 and points to v via x .

The processes labelled by q_1 may build a list of arbitrary length by electing one q_z node at a time. The communication pattern for handling message $\mathbf{b}(zero, id)$ is the same as before, therefore the same reasoning applies: at each step, all of the nodes in state q_1 have their local register x pointing to the newly elected node $z \in V$ in state q_z (or to v if it is the first q_z), and z is excluded from the list construction from now on. As soon as a node $h \in V$ switches to q_h , every remaining process moves to q_{halt} : exactly one list has been built, and the protocol has to stop. According to Definition 4.1, \mathcal{P}_{lb}^{fc} is therefore a forward $\langle q_h, q_z, q_t \rangle$ -list builder for \Rightarrow_b and Γ_0^{fc} on x . Since it also has a read-only register id , we can conclude that $Cov^{fc}(2, 1)$ is undecidable thanks to Lemma 4.1. □

5.2. Decidability of $Cov^{fc}(1, 1)$

We now consider the problem $Cov^{fc}(1, 1)$, where configurations are fully connected and do not change dynamically, processes have a single register, and each message has a single data field. To show decidability, we employ the theory of well-structured transition systems [1, 21] to define an algorithm for backward reachability based on a symbolic representation of infinite set of configurations, namely multisets of multisets of states in Q . In the following we use $[a_1, \dots, a_k]$ to denote a multiset containing (possibly repeated) occurrences a_1, \dots, a_k of elements from some fixed domain. For a multiset m , we use $m(q)$ to denote the number of occurrences of q in m .

In the sequel we consider a $(1, 1)$ -protocol $\mathcal{P} = \langle Q, R, q_0 \rangle$. The set Ξ of symbolic configurations contains, for every $k \in \mathbb{N}$, all multisets of the form $\xi = [m_1, \dots, m_k]$, where m_i for $i \in [1..k]$ is in turn a multiset over Q . Given $\xi = [m_1, \dots, m_k] \in \Xi$, $\langle V, E, L \rangle \in \llbracket \xi \rrbracket$ iff there is a function $f : V \rightarrow [1..k]$ such that (1) for every $v, v' \in V$, if $L_M(v) = L_M(v')$ then $f(v) = f(v')$ and (2) for all $i \in [1..k]$ and $q \in Q$, $m_i(q)$ is equal to the number of nodes $v \in V$ s.t. $f(v) = i$ and $L_Q(v) = q$. Intuitively, each m_i is associated to one of the k distinct values of the register (the actual values do not matter), and $m_i(q)$ counts how many nodes in state q have the corresponding value. We now define an ordering over Ξ .

Definition 5.1. Given $\xi = [m_1, \dots, m_k] \in \Xi$ and $\xi' = [m'_1, \dots, m'_p] \in \Xi$, $\xi \prec \xi'$ iff $k \leq p$ and there exists an injection $h : [1..k] \rightarrow [1..p]$ such that for all $i \in [1..k]$ and all $q \in Q$, $m_i(q) \leq m_{h(i)}(q)$, i.e. m_i is included in $m_{h(i)}$.

The following properties then hold.

Proposition 5.1. The ordering (ξ, \prec) over symbolic configurations is a well-quasi ordering (wqo), i.e. for any infinite sequence $\xi_1 \xi_2 \dots$ there exist $i < j$ s.t. $\xi_i \prec \xi_j$.

Proof:

By Dickson's Lemma, we know that, for multisets over a finite set Q , multiset inclusion is a wqo. By Higman's Lemma, for multisets built over a wqo domain, multiset inclusion (in which elements are compared using the wqo) is still a wqo. Thus, the juxtaposition of the two orderings yields a well-quasi ordering. \square

We now exhibit an algorithm $\text{PRE}_{\mathcal{P}}$ that works on symbolic representations in Ξ of configurations of networks with one register x in each node and one data field in each message. For a symbolic configuration $\xi = [m_1, \dots, m_k]$, m_i is a multiset over Q for $i \in [1, \dots, k]$. The representation allows us to maintain the minimal information about relations ($=$ and \neq) over data and forget about specific values of the data and minimal constraints on the number of nodes (sharing the same value) in each state in Q . For $S \subseteq \Gamma$, we define $\text{pre}_{\mathcal{P}}(S)$ as the set $\{\gamma \mid \gamma \Rightarrow_b \gamma' \text{ and } \gamma' \in S\}$. The following proposition then holds.

Proposition 5.2. There exists an algorithm $\text{PRE}_{\mathcal{P}}$ that takes in input $I \subseteq \Xi$ and returns a set $I' \subseteq \Xi$ s.t. $\llbracket I' \rrbracket = \text{pre}_{\mathcal{P}}(\llbracket I \rrbracket)$.

To prove the proposition, in the rest of the section we define the algorithm $\text{PRE}_{\mathcal{P}}$. The algorithm that computes $\text{PRE}_{\mathcal{P}}$ computes minimal representations of predecessors by applying backwards broadcast and receive rules to elements of I . Actually,

$$\text{PRE}_{\mathcal{P}}(I) = \bigcup_{b \in B} \text{PRE}_b(I)$$

where $B = (Q \times \text{Send}_{\Sigma}^{1,1} \times Q) \cap R$ is the set of all broadcast actions. Furthermore,

$$\text{PRE}_b(\{\xi_1, \dots, \xi_n\}) = \bigcup_{i \in [1, \dots, n]} \text{PRE}_b(\xi_i)$$

We focus our attention on PRE_b for a given broadcast b and a given symbolic configuration ξ . In the rest of the section we assume that

- $b = \langle q, \mathbf{b}(\text{msg}, p_1), q' \rangle$,
- $\xi = [m_1, \dots, m_k]$ where each multiset m_i of symbols in Q is associated to a distinct value for register x ;

To symbolically compute predecessors, we recall that a configuration $\xi = [m_1, \dots, m_k]$ denotes the infinite set of multisets of the form $\gamma = [m'_1, \dots, m'_k, m_{k+1}, \dots, m_r]$ where, in turn, m_i is a sub-multiset of m'_i for $i \in [1..k]$. These kind of configurations are obtained either by adding either nodes with identifiers equal to those already present in ξ (e.g. when m'_i is strictly larger than m_i) or by adding nodes with fresh identifiers (the additional multisets m_{k+1}, \dots, m_r).

We recall that reception rules in R have four possible types of action $?p_1$, $?\overline{p_1}$, $\downarrow p_1$, and $*$. For a reception rule of the shape $r = \langle q_i, \mathbf{r}(msg, \alpha), q'_i \rangle$ with $\alpha \in \{?p_1, ?\overline{p_1}, \downarrow p_1, *\}$ we call q_i [q'_i] the precondition [resp. postcondition] of the rule r .

To illustrate the rationale behind our construction of PRE_b , we first illustrate the key ideas with the help of an example.

Example 5.1. Consider a symbolic configuration $\xi = [m_1, m_2]$, where $m_1 = [q_2, r_2, u_2]$ and $m_2 = [v_2, v_2]$ represent two groups of nodes s.t. m_1 contains at least three processes with the same value c_1 in the register and m_2 contains at least two processes with the same value c_2 in the register. Consider now the rules: $\langle q_1, \mathbf{b}(a, 1), q_2 \rangle$, $\langle r_1, \mathbf{r}(a, ?1), r_2 \rangle$, $\langle u_1, \mathbf{r}(a, \downarrow 1), u_2 \rangle$, and $\langle v_1, \mathbf{r}(a, ?\overline{1}), v_2 \rangle$.

We assume that the sender is the node in state q_2 in m_1 . Its precondition is then the state q_1 . We now have to consider reactions. We first consider the nodes in m_1 (same identifier as the sender) with state r_2 and u_2 . Each node matches a postcondition of a test or store rule. However, for each of them there are two cases to consider: they either reacted to the current broadcast or they reached their state in a previous step. Thus the precondition for r_2 can be either r_1 or r_2 itself. Both preconditions must remain in the same group. For u_2 we have to be more careful. The precondition can be either u_1 or u_2 . However, since the value of the register before store is unknown, they can either remain in the same group, move to other existing groups in ξ , or to newly created groups (associated to fresh identifiers). Similarly, the preconditions for nodes in state v_2 can be either v_1 or v_2 . These processes remain in the same group. Among the predecessors we have then symbolic configurations like: $[[q_1, r_1, u_1], [v_1, v_1]]$, $[[q_1, r_1, u_1], [v_1, v_2]]$, $[[q_1, r_1, u_1], [v_2, v_2]]$, $[[q_1, r_2, u_2], [v_1, v_1]]$, $[[q_1, r_1], [u_1, v_1, v_1]]$, $[[q_1, r_1], [v_1, v_1], [u_1]]$, etc.

To take into account the upward closure of the denotations of ξ , we also have to consider possible extensions of ξ with additional nodes that match postconditions of send and receive rules. For instance, we may assume that there exists another node in state q_1 in m_2 , and then recompute predecessors starting from $[[q_2, r_2, u_2], [q_2, v_2, v_2]]$ or assume that there exist a node with a fresh value with postcondition q_2 and then compute the predecessors from $[[q_2, r_2, u_2], [v_2, v_2], [q_2]]$, and so on. Similarly we have to consider possible extensions of ξ with matching postconditions of reception rules and computed predecessors for them too. Luckily, we have to consider only finitely many extensions since we are interested in computing minimal configurations only. Specifically, extensions of ξ with more than one occurrence of the same postcondition will lead to non-minimal configurations, and thus they can be avoided.

All the predecessor symbolic configurations are then collected together and only the minimal one w.r.t. \prec form the basis of the symbolic representation of predecessor configurations.

To simplify the presentation, we present a non-deterministic algorithm to compute $\text{PRE}_b(\xi)$ defined via a case analysis on broadcast and receptions. The algorithm can be transformed into a deterministic one by exploring all possible alternatives. Consider the broadcast rule $b = \langle q, \mathbf{b}(msg, p_1), q' \rangle$ and the symbolic configuration $\xi = [m_1, \dots, m_k]$. We recall that ξ denotes all configurations larger than ξ w.r.t. \prec . However, to compute predecessors it is enough to consider extensions of ξ with at most one occurrence of a sender process. Adding explicit representations of receivers is not necessary since the corresponding predecessors would produce non minimal representations. This is due to the fact that update of receiver states do not influence the state of other processes.

In what follows, the operator \oplus denotes the multiset union. We first define the finite set of possible extensions of ξ as follows:

$$\text{Ext}_{q'}(\xi) = \{\xi\} \cup \{\xi \oplus [q']\} \cup \{[m_1, \dots, m_i \oplus [q'], \dots, m_k] \mid i \in [1, k]\}$$

The intuition behinds this extension is that we have to consider the configuration in $\llbracket \xi \rrbracket$ where the state q' appears since these are the configuration we will get after taking the broadcast rule $b = \langle q, \mathbf{b}(msg, p_1), q' \rangle$.

For each $\xi' \in Ext_{q'}(\xi)$, we will show how to compute a set of symbolic predecessor. Let $\xi' \in Ext_{q'}(\xi)$ with $\xi' = [m_1, \dots, m_k]$. We can now assume now that $m_i = [q'] \oplus m$ for some i . We first notice that ξ' has no predecessors if there are states that correspond to preconditions of receptions of msg that could be fired with b , unless receptions preserve the state with a loop on q' . Indeed, since the topology is fully connected all nodes must react to the broadcast b (i.e. a precondition state of a reception cannot remain in the current state unless the reception does not change it). Let us assume now that the previous case does not apply. We will give now the way to obtain set of predecessors of ξ of the shape $[m'_1, \dots, m'_\ell]$ with $\ell \geq k$.

To define m'_i we first non-deterministically decompose m into the multisets w_1, w_2, w_3, w_4 , where w_1 contains target states of receptions for msg with action $?p_1$, w_2 contains target states of receptions for msg with action $*$, w_3 contains target states of receptions for msg with action $\downarrow p_1$, and w_4 contains the remaining states. We then non-deterministically decompose w_i into u_i, v_i . In other words we have that

$$m = \left(\bigoplus_{i=1}^3 (u_i \oplus v_i) \right) \oplus w_4$$

We can now define the effect of b on m as the multiset m' defined as

$$m' = \left(\bigoplus_{i=1}^3 (pre(u_i) \oplus v_i) \right) \oplus w_4$$

To multiset $pre(u_i)$ is defined by case analysis on receptions.

- For rules with test and ignore action $pre(u_i)$, $i \in [1, 2]$, is obtained by replacing each occurrence of a postcondition with the corresponding precondition of a (non deterministically selected) reception rule for msg (i.e $pre(u_i)$ and u_i have the same size).
- For rules with store actions, $pre(u_3)$ is obtained by first replacing each occurrence of a postcondition in u_3 with the corresponding precondition of a (non deterministically selected) reception rule for msg , and then by non-deterministically splitting the resulting multiset into two multisets, namely $pre(u_3)$ and $pre_{\neq}(u_3)$. The latter processes correspond to processes with register values distinct from those in m .

We will then have $m'_i = [q] \oplus m'$.

We now have to generate the multiset m'_j associated to the multisets m_j with $j \in [1..k] \setminus \{i\}$. To compute m'_j we consider reception rules that either have $?p_1$ (i.e. the value in the register is distinct from the sender) or $*$ action. We non-deterministically split m_j in

$$m_j = u_j \oplus v_j$$

so that u_j contains postcondition states of receptions of msg , and compute m'_j by applying reception backwards to u_j , i.e.,

$$m'_j = pre(u_j) \oplus v_j$$

Finally, the multiset $pre_{\neq}(u_3)$ is non-deterministically distributed among the multisets m'_j with $j \neq i$ or used to add to the resulting configuration additional multisets (all possible splittings of sub multisets of $pre_{\neq}(u_3)$). In the former case the processes in $pre_{\neq}(u_3)$ correspond to processes with register values that were already present in ξ' . In the latter case they correspond to processes whose register value is fresh with respect to those in ξ' .

When we have computing sets of symbolic predecessors for each $\xi' \in Ext_{q'}(\xi)$, we take for $PRE_b(\xi)$ the minimal set I of symbolic representations representing the union of all the computed sets and which is obtained by removing redundant representations and representations that are larger than others.

5.3. Decision Procedure

Following [3], the algorithm for $PRE_{\mathcal{P}}$ can be used to effectively compute a finite representation of the set of predecessors $pre_{\mathcal{P}}^*(\llbracket Bad \rrbracket)$ for a set of symbolic configurations Bad . The computation iteratively applies $PRE_{\mathcal{P}}$ until a fixpoint is reached. The termination test is defined using \prec . The wqo \prec ensures termination of the computation [1]. The following theorem then holds.

Theorem 5.2. $Cov^{fc}(1, 1)$ is decidable.

Proof:

We show how to apply the symbolic predecessor computation based on $PRE_{\mathcal{P}}$. Let φ be a query with set of variables Z . The (in)equalities in φ induce a finite set P_1, \dots, P_k of partitions of Z . Each partition $P_i = \{X_1^i, \dots, X_{u_i}^i\}$ is such that X_j^i contains variables that may take the same value (i.e. there are no \neq constraints between them in φ). For a partition X , we define the multiset m_X of symbols in Q for which there exists a predicate $q(z)$ with $z \in X$. Thus $P_i = \{X_1^i, \dots, X_{u_i}^i\}$ can be represented via the multiset of multisets $s_i = [m_{X_1^i}, \dots, m_{X_{u_i}^i}]$. The set $I = \{s_1, \dots, s_k\}$ corresponds to the minimal elements of the set of configurations that satisfy φ . To apply the algorithm we set $Bad = I^\uparrow$, i.e., $I = \min(Bad)$. We compute then the least fixpoint of $PRE_{\mathcal{P}}$, say $PRE_{\mathcal{P}}^*(I)$. To check if the resulting set of symbolic configurations contains an initial state, we need to search for a finite basis $\langle V, L, E \rangle$ (where $E = V \times V \setminus \{(v, v) \mid v \in V\}$) in which all nodes have initial states as labels, and in which there cannot be two nodes with the same value in the register (initially all processes have distinct values in local registers). Using the multiset representation, we need to search for a multiset consisting of multisets of the form $[q_0]$ where q_0 is the initial state of the protocol, i.e. coverability holds if and only if $\llbracket [q_0], \dots, [q_0] \rrbracket \in PRE_{\mathcal{P}}^*(I)$. \square

An alternative proof can be given by resorting to an encoding into coverability in data nets [25]. We present such an encoding in [13]. We did not investigate the reverse translation, i.e. whether data nets can be encoded into our model with fully-connected topology, one register and one-field per message, but due to the expressive power of data nets, it seems that it would be difficult to get such a reduction.

We consider now the complexity. We observe that, without registers and fields our model boils down to the AHNs of [15]. For fully connected topologies, AHN can simulate reset nets as shown in [16] and hence the parameterized coverability problem for such a model is Ackermann-hard. In fact, this can be deduced from the fact that the complexity of coverability in reset nets is Ackermann-hard [28]. Furthermore it has been show later that for fully connected topology, the parameterized coverability problem in AHN is in fact Ackermann-complete [27]. Following these results, we obtain the following theoretical lower bound.

Corollary 5.1. $Cov^{fc}(0, 0)$ and $Cov^{fc}(1, 1)$ are Ackermann-hard.

6. Conclusions

In this paper we investigated decidability and complexity for parameterized verification of a formal model of distributed computation based on register automata communicating via broadcast messages with data. The results we obtained are summarized in Table 1 where we recall that r stands for the number of registers present in each node of the network and f characterizes the number of fields allowed in the messages of the protocol. As already mentioned, for $r = 0$ and $f = 0$ the parameterized coverability problem had already been studied previously. From a technical point of view, our results can be viewed as a fine grained refinement of those obtained for the case without data. For instance, undecidability follows from constructions similar to those adopted in [15]. They are based on special use of data for building synchronization patterns that can be applied even in fully connected networks. We point out the fact that we have characterize exhaustively the decidability status of the parameterized coverability problem for Broadcast Networks of Register Automata since as mentioned before it does not make sense to consider more data fields in the message than number of registers in the nodes. The only problem left open is the precise complexity characterization of $Cov^{fc}(1, 1)$.

Problem	Protocol		Complexity
	r	f	
$Cov(r, f)$	0	0	PTIME[14]
	$r \geq 1$	1	PSPACE-complete [Thm. 4.2]
	$r \geq 2$	$f \geq 2$	Undecidable [Thm 4.1]
$Cov^{fc}(r, f)$	0	0	Ackermann-complete [16, 27]
	1	1	Decidable and Ackermann-hard [Thm 5.2]
	$r \geq 2$	$f \geq 1$	Undecidable [Thm. 5.1]
$Cov^b(r, f)$	$r \geq 0$	$f \geq 0$	Undecidable [15]

Table 1. Decidability and complexity boundaries

In our model, we have assumed that in an initial configuration the same data is not present twice (in any register). One could easily verify that for the cases where we obtain decidability (for $Cov(*, 1)$ and $Cov^{fc}(1, 1)$), the same techniques can be applied if we relax this hypothesis and hence we would obtain the same decidability results with the same complexity. For the cases where we have proved undecidability ($Cov(2, 2)$ and $Cov^{fc}(2, 1)$), we can observe that we need in our undecidability proofs one read-only register containing a different identifier for each node of the network. Hence if we relax too much the hypothesis on the initial configurations (such that nodes cannot be anymore distinguished through this register) it is not clear whether the problems will remain undecidable or not.

Finally, in this work, we have considered only safety properties for our model, but as we mention

with the example provided in Section 3, it would be interesting to investigate liveness property that states that eventually something desired happens. For the cases of fully connected topologies in which we rely on the theory of well-structured transition systems, positive results might be difficult to obtain since backward algorithms, as the one we use for $Cov^{fc}(1, 1)$, will not apply, but for the case with reconfiguration with one register and one field per message it might be easier. We plan to investigate such problems in future works. Another possible direction for future research would be to see what happens when the data are ordered. It would be also interesting to understand how such techniques can be applied to real protocols by analyzing for instance approximate executions (our model being not expressive enough to characterize precisely the behaviors of a concrete protocols). In fact to verify the behaviors of concrete protocols with our method, we would need to abstract away some aspects of the protocols in order to be able to encode them in our model. One negative point is that such methods might lead to false alarms (bugs in the approximation which cannot occur in the real protocols) and an idea left also as possible direction of research could then be to provide a way to refine the abstraction in order to discard such wrong executions.

References

- [1] Abdulla, P. A., Cerans, K., Jonsson, B., Tsay, Y.-K.: General Decidability Theorems for Infinite-State Systems, *LICS'96*, IEEE Computer Society, 1996.
- [2] Abdulla, P. A., Delzanno, G., Rezine, O., Sangnier, A., Traverso, R.: On the Verification of Timed Ad Hoc Networks, *FORMATS'11*, 6604, Springer, 2011.
- [3] Abdulla, P. A., Jonsson, B.: Ensuring completeness of symbolic verification methods for infinite-state systems, *Theor. Comput. Sci.*, **256**(1-2), 2001, 145–167.
- [4] Alur, R., Dill, D. L.: A Theory of Timed Automata, *Theor. Comput. Sci.*, **126**(2), 1994, 183–235.
- [5] Aminof, B., Jacobs, S., Khalimov, A., Rubin, S.: Parameterized Model Checking of Token-Passing Systems, *VMCAI'14*, 8318, 2014.
- [6] Bertrand, N., Fournier, P., Sangnier, A.: Playing with Probabilities in Reconfigurable Broadcast Networks, *FOSSACS'14*, 8412, Springer, 2014.
- [7] Bollig, B., Gastin, P., Schubert, J.: Parameterized Verification of Communicating Automata under Context Bounds, *RP'14*, 8762, 2014.
- [8] Cheng, A., Esparza, J., Palsberg, J.: Complexity Results for 1-Safe Nets, *TCS*, **147**(1&2), 1995, 117–136.
- [9] Clarke, E. M., Talupur, M., Touili, T., Veith, H.: Verification by Network Decomposition, *CONCUR'04*, 3170, 2004.
- [10] Delzanno, G.: Constraint-Based Verification of Parameterized Cache Coherence Protocols, *FMSD*, **23**(3), 2003, 257–301.
- [11] Delzanno, G., Rosa-Velardo, F.: On the coverability and reachability languages of monotonic extensions of Petri nets, *Theor. Comput. Sci.*, **467**, 2013, 12–29.
- [12] Delzanno, G., Sangnier, A., Traverso, R.: Parameterized Verification of Broadcast Networks of Register Automata, *RP'13*, 8169, Springer, 2013.
- [13] Delzanno, G., Sangnier, A., Traverso, R.: *Parameterized Verification of Broadcast Networks of Register Automata (Technical Report)*, Technical report, TR-13-03, DIBRIS, University of Genova, 2013, Available at the URL <http://verify.disi.unige.it/publications/>.

- [14] Delzanno, G., Sangnier, A., Traverso, R., Zavattaro, G.: On the Complexity of Parameterized Reachability in Reconfigurable Broadcast Networks, *FSTTCS'12*, 18, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.
- [15] Delzanno, G., Sangnier, A., Zavattaro, G.: Parameterized Verification of Ad Hoc Networks, *CONCUR'10*, 6269, Springer, 2010.
- [16] Delzanno, G., Sangnier, A., Zavattaro, G.: On the Power of Cliques in the Parameterized Verification of Ad Hoc Networks, *FOSSACS'11*, 6604, Springer, 2011.
- [17] Emerson, E. A., Namjoshi, K. S.: On Model Checking for Non-Deterministic Infinite-State Systems, *LICS'98*, IEEE Computer Society, 1998.
- [18] Esparza, J.: Keeping a Crowd Safe: On the Complexity of Parameterized Verification (Invited Talk), *STACS'14*, 25, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014.
- [19] Esparza, J., Finkel, A., Mayr, R.: On the Verification of Broadcast Protocols, *LICS'99*, IEEE Computer Society, 1999.
- [20] Esparza, J., Ganty, P., Majumdar, R.: Parameterized Verification of Asynchronous Shared-Memory Systems, *CAV'13*, 8044, Springer, 2013.
- [21] Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere!, *Theor. Comput. Sci.*, **256**(1-2), 2001, 63–92.
- [22] German, S. M., Sistla, A. P.: Reasoning about Systems with Many Processes, *J. ACM*, **39**(3), 1992, 675–735.
- [23] Kaminski, M., Francez, N.: Finite-Memory Automata, *Theor. Comput. Sci.*, **134**(2), 1994, 329–363.
- [24] Konnov, I., Veith, H., Widder, J.: Who is afraid of Model Checking Distributed Algorithms?, Unpublished contribution to: CAV Workshop (*EC*)², 2012.
- [25] Lazic, R., Newcomb, T., Ouaknine, J., Roscoe, A. W., Worrell, J.: Nets with Tokens which Carry Data, *Fundam. Inform.*, **88**(3), 2008, 251–274.
- [26] Minsky, M.: *Computation, Finite and Infinite Machines*, Prentice Hall, 1967.
- [27] Schmitz, S., Schnoebelen, P.: The Power of Well-Structured Systems, *CONCUR'13*, 8052, Springer, 2013.
- [28] Schnoebelen, P.: Revisiting Ackermann-Hardness for Lossy Counter Machines and Reset Petri Nets, *MFCS'10*, 6281, Springer, 2010.