Safety Analysis in Broadcast Networks Defined by Graph Grammars

Christoffer Lind Andersen[1], Radu Iosif[1], and Arnaud Sangnier[2]

[1] Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG, France
[2] DIBRIS, Università di Genova, Italy

Abstract. We consider families of networks where processes communicate by synchronous broadcast (i.e., a sent message is received by all the neighbours of the emitter) and we study the following safety problem: is there a network in the given family, such that some process can reach an error location? Specifically, we focus on families of network topologies defined by graph grammars, where each node of the produced graph can be either a clique or a cloud (anti-clique) of processes of unbounded sizes. We show that, in general, the considered safety problem is undecidable, when the communication is reliable, and becomes decidable with unreliable communication (i.e., broadcast messages can be lost) or whenever the protocols executed by the different processes cannot send and receive messages from the same control state (also known as the wait-only syntactic restriction).

1 Introduction

Present-day computer systems are increasingly distributed, with myriads of devices that communicate over a network. The design of such networks often requires structuring the network into subnets that have specific communication topologies, which determines their functionalities. The physical layers may also vary inside the same network, from wired, i.e., obeying a point-to-point communication policy, to wireless, i.e., relying on a broadcast communication model.

The huge scale of networks, such as datacenter networks with $\geq 10^4$ routers for a regional hub and continuously growing, requires *parametric* verification techniques, that work no matter how many servers in a rack, switches in a cluster, clusters in a layer, etc. Despite decades of theoretical research and an impressive body of results (see [4] for a nice survey), these techniques consider a small number of fixed communication topologies, such as cliques that model broadcast [14], or rings that model point-to-point communication [6].

One of the reasons for focusing on particular hard-coded architectures (i.e., cliques and rings) is that the parameterized verification of safety properties (i.e., reachability of a given error state by some process in the network) is undecidable even for ring topologies that circulate sufficiently many message types [6]. For clique (or equivalently complete) topologies instead, the problem has been shown to be decidable when the communication is performed by synchronized pairwise rendez-vous [14] (with a polynomial time algorithm) or by broadcast [12].

Some of the limitations imposed by such hard-coded architectures have been overcome in recent years, with the development of parametric verification techniques for systems described using logic [3] or graph grammars [18,5,16]. Logic-based specification languages are particularly suitable for describing graph properties, such as planarity, k-colorability or connectivity. On the other hand, graph grammars are appealing for their constructive aspect, i.e., the ability of specifying how large graphs are built from smaller subgraphs. Both logics and graph grammars are at the core of the theory of formal languages, due to seminal result of Courcelle and Engelfriet, which shows the decidability of Monadic Second Order (MSO) logic over the classes of context-free languages produced by hyperedge-replacement (HR) and vertex-replacement (VR) grammars (see [7] for a survey).

Contributions. In this paper, we consider the parameterized safety problem for families of topologies described by VR-grammars, with a broadcast communication model, where each process sends messages to all its neighbors in the graph. Since the parametric safety problem is known to be undecidable even for line topologies [9], we consider families of underlying *clique-and-cloud* topologies, in which each vertex represents either a clique (i.e., a complete graph) or a cloud (i.e., a completely disconnected graph). The vertices of the expanded cliques/clouds are pairwise linked as indicated by the underlying graph. In other words, our topology specification language does not allow to describe a line, ring or tree of processes, but instead lines, rings or trees of cliques/clouds of arbitrary sizes, respectively. This style of specification is inspired by the layered architectures of modern datacenters, such as Azure [17], where each layer consists of a cloud of pairwise disconnected switches, that are connected with all the switches from the upper and lower layers.

When considering such families of structured cliques/clouds, the parametric safety problem becomes undecidable under the reliable communication model, even for the simplest clique-and-cloud topologies, such as a line. This is not surprising, because, for instance, reliable broadcast in a clique can be used to determine a leader (i.e., send all non-leader processes into an inactive sink state), hence a line of cliques can become a line of leaders, able to simulate a 2-counter Minsky machine [9].

To recover decidability, we consider two natural restrictions: *unreliable communication*, i.e., where messages can be lost, and *reliable communication* with the wait-only restriction on the local control of a process, i.e., no states from which messages can be both sent and received [15]. All decidability results are obtained by an encoding of the safety verification problem in an MSO formula, that is checked against the VR-grammar that describes the family of clique-and-cloud topologies. The latter problem (i.e., the existence of a model of an MSO formula in the language of a VR-grammar) is decidable, by a classical result of Courcelle and Engelfriet [7].

Related Works. Parametric systems with clique topologies, that communicate using reliable broadcast have been among the first to be studied in the liter-

ature [12]. The technique for establishing decidability of the parametric safety problem relies on an algorithm based on well-structured transition systems [1]. Unfortunately, these results strongly depend on the hypothesis that the topology of each system is a single clique. In contrast, our work goes beyond single cliques and allows to reason about parametric systems whose topologies consist of arbitrary graph-like structures, with vertices representing cliques and/or clouds.

In [9,8], Delzanno et al. study the parametric safety problem for reliable and unreliable broadcast seeking for a topology which breaks the safety property. They show that this problem is undecidable for reliable broadcast [9] and is decidable in polynomial time when the communication is unreliable [8]. Another way to regain decidability with reliable broadcast is to restrict the set of topologies by considering graphs where the length of the minimal path between two vertices is bounded [10]. In these works, the set of considered topologies is not able to describe precise architectures, it is either unrestricted or the restriction regards only the length of paths in the graphs.

Graph grammars and MSO have been used to specify families of topologies using the point-to-point (rendez-vous) communication model. In this model, each network of processes is simulated by an equivalent 1-safe Petri net [13], meaning that the parametric verification problem amounts to reasoning about infinite families of 1-safe Petri nets, that share a certain structural invariant. On one hand, parametric systems specified using HR-grammars can be translated into a finite set of Petri nets, that over approximate their behavior [5]. On the other hand, parametric systems specified by VR-grammars can be converted into HR-systems that preserve the safety properties [16]. For token-passing systems, where the rendez-vous communication is restricted to the passing of a single token among processes, the decidability of the safety verification problems is established using the same decidability result of Courcelle and Engelfriet [7] used in this paper [2]. Unfortunately, such decidability results for rendez-vous systems cannot be easily transferred to broadcast systems, mainly because, with broadcast, the number of participants in an interaction is unbounded.

Notation. The set of natural numbers is denoted by $\mathbb{N}$. Given numbers $i, j \in \mathbb{N}$, we write $[i, j] \stackrel{\text{def}}{=} \{i, i+1, \ldots, j\}$, assumed to be empty if $i > j$. The cardinality of a finite set A is denoted by $\|A\|$. The disjoint union $A \uplus B$ is defined as the union of A and B, if $A \cap B = \emptyset$, and undefined, otherwise. For a relation $R \subseteq A \times A$, we denote by R^* its reflexive and transitive closure. We write $[x \to y]$ for the function $f : A \to A$ that sends x into y and behaves as the identity over A everywhere else.

Due to lack of space, omitted proofs can be found in Appendix.

2 Network Model

This section defines the network model. Intuitively, a network is a binary graph, whose vertices run identical copies of one or more finite-state machines, called

process types. Instances of process types, also called processes, placed on adjacent vertices in the network graph may engage in broadcast communication along the edges: a process sends a message and several or all of its neighbours receive it. As usual, sending or receiving a message is modelled by a local state change within a sender or receiver process, respectively.

2.1 Process Types and Network Topologies

Let Σ be a finite alphabet of messages, considered fixed in the rest of this paper.

Definition 1 (Process Type). *A* process type P *is a tuple* (Q, q_{in}, δ) *where:*

- Q *is a finite set of control states,*
- $q_{in} \in Q$ *is the initial state,*
- $\delta \subseteq Q \times \{?m, !m \mid m \in \Sigma\} \times Q$ *is the action relation; as usual, we write* $q \xrightarrow{\alpha}_{\delta} q'$ *instead of* $(q, \alpha, q') \in \delta$.

In the rest of this paper, we fix a set of process types $\mathcal{P} = \{P_1, \ldots, P_\ell\}$, for which we assume that $P_i \stackrel{\text{def}}{=} (Q_i, q_{in,i}, \delta_i)$ for all $i \in [1, \ell]$ and that $Q_i \cap Q_j = \emptyset$ for all $i \neq j \in [1, \ell]$, i.e., each process type has a set of local states disjoint from the other. Let $Q \stackrel{\text{def}}{=} \biguplus_{1 \leq i \leq \ell} Q_i$ and $\delta \stackrel{\text{def}}{=} \biguplus_{1 \leq i \leq \ell} \delta_i$ denote the overall set of states and transition relation, respectively.

We represent a network by a simple undirected binary graph, with no self-loops, such that each vertex of the network graph is labelled by a process type.

Definition 2 (Network Topology). *A* network topology *is a* $\mathcal{P}$*-labeled graph* $G = (V, E, lab)$ *such that:*

- V *is a finite set of vertices,*
- $E \subseteq (V \times V) \setminus \{(v, v) \mid v \in V\}$ *is an irreflexive and symmetric edge relation, i.e.,* $(u, v) \in E \iff (v, u) \in E$ *for all* $u, v \in V$,
- $lab : V \mapsto \mathcal{P}$ *labels each vertex with a process type.*

2.2 Execution Semantics

The execution of a network is formally described by a transition relation over configurations. A configuration is a snapshot of the network, that gives the local state of each process. Formally, a configuration of a network topology $G = (V, E, lab)$ is a function $c : V \mapsto Q$, where Q is the set of states, such that $c(v) \in Q_i$ if $lab(v) = P_i$, for each vertex $v \in V$. The *initial* configuration c^G_{in} is given by $c^G_{in}(v) = q_{in,i}$, for each vertex $v \in V$ such that $lab(v) = P_i$. For a given configuration c, we denote by $\#_c(q) \stackrel{\text{def}}{=} \|\{v \in V \mid c(v) = q\}\|$, whenever the network topology in question is understood. We denote by $\mathcal{C}_G$ the set of configurations for a graph G.

For a given network topology G, we define a transition relation $\Rightarrow_G \subseteq \mathcal{C}_G \times \mathcal{C}_G$. For a family $\mathcal{G}$ of topologies, we define the relation $\Rightarrow_{\mathcal{G}} \stackrel{\text{def}}{=} \bigcup_{G \in \mathcal{G}} \Rightarrow_G$. We omit

mentioning the network topology of the family thereof, whenever this is understood or not important. In the definition of $\Rightarrow_G$, hence of $\Rightarrow_{\mathcal{G}}$, we distinguish the cases of *reliable* and *unreliable* communication. The communication is reliable if each message that has been sent, via a $!m$ action, will be received, via a matching $?m$ action, by each process able to take that action. Instead, in unreliable communication, some messages may be lost. Let $G = (V, E, lab)$ be a network graph, in the following.

Reliable communication. The execution semantics for reliable communication is the relation $\Rightarrow_r \subseteq \mathcal{C}_G \times \mathcal{C}_G$, such that $c \Rightarrow_r c'$ if and only if, (i) there exists a vertex $u \in V$ and a message $m \in \Sigma$ such that $c(u) \xrightarrow{!m}_\delta c'(u)$ and (ii) for each vertex $v \in V \setminus \{u\}$, exactly one of the following holds:

- $(u, v) \in E$ and $c(v) \xrightarrow{?m}_\delta c'(v)$, i.e., v is a neighbour of u that can receive m,
- $(u, v) \in E$, $c(v) \xrightarrow{?m}_\delta q'$ for no $q' \in Q$ and $c'(v) = c(v)$, i.e., v is a neighbour of u, but it cannot receive m, or
- $(u, v) \notin E$ and $c'(v) = c(v)$, i.e., v is not a neighbour of u.

In this case, we also say that u sends the message m. Note that a send action is always non-blocking, meaning that it will be executed even if no neighbouring process can execute a matching reception.

Unreliable communication. The execution semantics for unreliable communication is the relation $\Rightarrow_u \subseteq \mathcal{C}_G \times \mathcal{C}_G$, such that $c \Rightarrow_u c'$ if and only if, (i) there exists a vertex $u \in V$ and a message $m \in \Sigma$ such that $c(u) \xrightarrow{!m}_\delta c'(u)$ and (ii) for each vertex $v \in V \setminus \{u\}$ one of the following holds:

- $(u, v) \in E$ and $c(v) \xrightarrow{?m}_\delta c'(v)$, i.e., v is a neighbour of u that receives m,
- $c'(v) = c(v)$, i.e., v does not receive the message.

In this case, even if a neighbour of the sender can receive a message m, via a $q \xrightarrow{?m} q'$ action, it may not take that action.

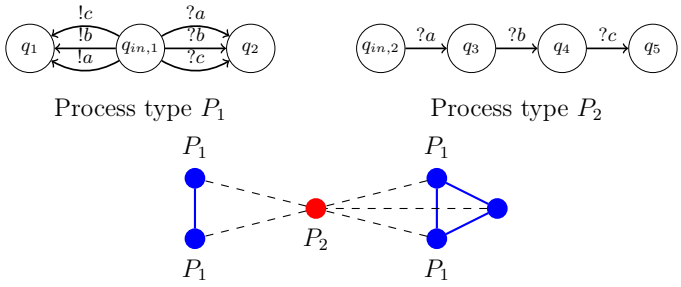

Fig. 1: Two process types and a network topology G

Example 1. We draw an example of two process types P_1 and P_2 on Figure 1 with respective initial states $q_{in,1}$ and $q_{in,2}$ together with a possible network topology G (all the lines are considered to be edges, even if they are dashed). We wonder whether in this latter topology, there is an execution from the initial configuration which leads to a configuration where the central vertex is in state q_5. Note that to reach such a configuration, the central vertex needs to receive sequentially the messages a, b and c. These messages can be produced by the vertices with process type P_1. However, if we consider the reliable semantics, it is not possible to reach this configuration, because in the two cliques of vertices with process type P_1 (on the left and on the right), as soon as a process broadcasts a message, it sends its neighbours in state q_2 from which they cannot broadcast messages any more. It is hence impossible that the three messages are sent. As a matter of fact, it is possible to put the central node in state q_4 but not in q_5. If we consider the unreliable semantics, then it is possible to reach a configuration with the central node in q_5, the reason being that when a process in one of the clique of process types P_1 has a neighbour which broadcasts a message, it does not have to receive it. We can hence have three vertices, where the first one broadcasts a, then the second one broadcasts b and finally the last one broadcasts c.

3 Families of Topologies

We introduce a specification method for describing infinite families of network topologies. The definition is given in two stages. First, we consider graphs whose vertices can be replaced either by a clique (i.e., complete graphs) or a cloud (i.e., a set of disconnected vertices), depending on a special label. These graphs are called *clique-and-cloud* topologies. Second, we introduce *graph grammars* that describe infinite sets of clique-and-cloud topologies. A family of topologies described in this way consists of the set of concrete expansions of the clique-and-cloud topologies from the language of the graph grammar.

3.1 Clique-and-Cloud Topologies

A *clique-and-cloud topology* $\mathcal{G}$ is a tuple $(\mathcal{V}, \mathcal{E}, Lab, \mathcal{V}_{\Delta}, \mathcal{V}_{\cdot\cdot})$ where $(\mathcal{V}, \mathcal{E}, Lab)$ is a network topology and $\mathcal{V} = \mathcal{V}_{\Delta} \uplus \mathcal{V}_{\cdot\cdot}$ is a partition of the vertices. Each vertex in $\mathcal{V}_{\Delta}$ represents a clique (i.e., a complete graph of unbounded size), whereas each vertex in $\mathcal{V}_{\cdot\cdot}$ represents a cloud (i.e., an unbounded number of disconnected vertices). Each undirected edge $(u, v) \in \mathcal{E}$ represents a set of undirected edges, between each vertex represented by u and each vertex represented by v.

Definition 3 (Expansion). *A network topology $G = (V, E, lab)$ is an* expansion *of the clique-and-cloud topology $\mathcal{G} = (\mathcal{V}, \mathcal{E}, Lab, \mathcal{V}_{\Delta}, \mathcal{V}_{\cdot\cdot})$ via a surjective function $f : V \mapsto \mathcal{V}$ if and only if the following hold:*

- *$lab(v) = Lab(f(v))$ for all $v \in V$, i.e., all vertices in the expansion inherit the process type of the originating vertex from the clique-and-cloud topology,*

– $(u, v) \in E$ *and (i)* $(f(u), f(v)) \in \mathcal{E}$ *or (ii)* $f(u) = f(v)$ *and* $f(u) \in \mathcal{V}_\Delta$*, i.e., an edge in the expansion corresponds to an edge from the clique-and-cloud topology or from one of the expanded cliques.*

We denote by $[\![\mathcal{G}]\!]$ *the set of pairs* $\{(G, f) \mid G$ *expands* $\mathcal{G}$ *via* $f\}$.

We sometimes write $G \in [\![\mathcal{G}]\!]$, meaning that there exists a surjective function f, such that $(G, f) \in [\![\mathcal{G}]\!]$. Note that each clique-and-cloud topology represents an infinite number of network topologies.

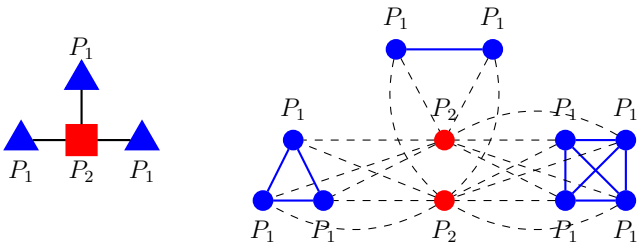

Fig. 2: A clique-and-cloud topology and one of its expansion

Example 2. On Figure 2, we depicted on the left a clique-and-cloud topology consisting of three cliques ▲ labeled with P_1 and one cloud node ■ labeled with P_2. The right side of the figure shows an expansion of this topology (we have drawn in dashed lines the edges which connect the cliques and clouds together, and in plain lines the edges of the cliques).

3.2 Graph Grammars

We consider graph grammars written using the vertex-replacement (VR) algebra, see [11,7] for surveys. This algebra manipulates graphs having a set of designated vertices, called *ports*. In our setting, ports are labelled (in addition to process types and clique/cloud labels) with elements from a countably infinite set Π of *port labels*. An *open* clique-and-cloud topology is a tuple $(\mathcal{V}, \mathcal{E}, Lab, \mathcal{V}_\Delta, \mathcal{V}_\therefore, Port)$, where $(\mathcal{V}, \mathcal{E}, Lab, \mathcal{V}_\Delta, \mathcal{V}_\therefore)$ is a clique-and-cloud topology as before and $Port : \mathcal{V} \rightharpoonup \Pi$ is a partial mapping associating several vertices with port labels. Note that the same port label can be used to label zero or more vertices. In the following, we do not distinguish isomorphic graphs, i.e, that differ only by a renaming of vertices.

We consider the VR algebra, whose domain is the set of open clique-and-cloud topologies, and whose operations are described below:

– *Unit graph*: for a port label $\pi \in \Pi$, $P \in \mathcal{P}$ and $\kappa \in \{\Delta, \therefore\}$, the constant (i.e., nullary operation) $\bullet\pi(P, \kappa)$ denotes the open topology with a single vertex,

labelled with the process type P and port π, that belongs to $\mathcal{V}_\kappa$, and no edges.
- *Edge creation*: $\mathtt{edg}_{\pi_1,\pi_2}(\mathcal{G})$ adds an undirected edge between each vertex labelled π_1 and each vertex labelled π_2 from $\mathcal{G}$, see Figure 3 (a).
- *Port redefinition*: $\mathtt{relab}_\alpha(\mathcal{G})$ renames each port label π of $\mathcal{G}$ as $\alpha(\pi)$, for a mapping $\alpha : \Pi \mapsto \Pi$, see Figure 3 (c).
- *Disjoint union*: $\mathcal{G}_1 \oplus \mathcal{G}_2$ takes the component-wise disjoint union of $\mathcal{G}_1$ and $\mathcal{G}_2$ (if the graphs are not disjoint, we take an isomorphic copy of one of them), see Figure 3 (b).

Given a set of variables $\mathcal{X}$ (of arity 0), a term of the VR-algebra over Π and $\mathcal{X}$ is built as usual from the operations of VR and the variables from $\mathcal{X}$. A *ground term* is a term with no variable. We write $\theta^{\mathtt{VR}}$ for the clique-and-cloud topology obtained by interpreting the function symbols in the ground term θ as the corresponding VR-operations and by removing the port labels.

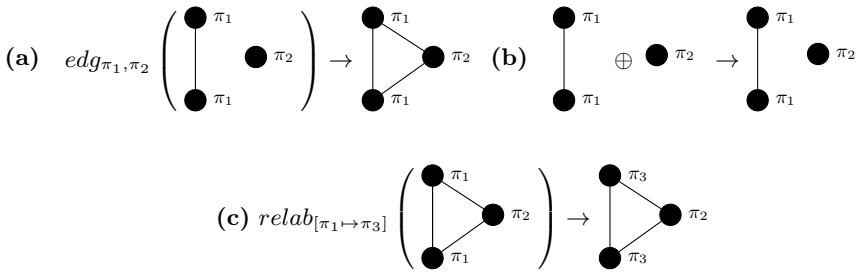

Fig. 3: The operations of the vertex replacement algebra; (**a**) edge creation, (**b**) disjoint union and (**c**) port redefinition

We use grammars (i.e., finite sets of recursive rules) over the signature of the VR-algebra to describe infinite sets of clique-and-cloud topologies:

Definition 4 (VR-graph grammar). *A* VR-graph grammar Γ *is a tuple* $(\mathcal{X}, \Pi, \mathcal{R}, X_0)$ *where* $\mathcal{X}$ *is a set of* non-terminals *(i.e., variables),* Π *is a set of vertex ports,* X_0 *is an initial non-terminal and* $\mathcal{R}$ *is a set of rules of the form* $X \hookrightarrow \theta$ *where* $X \in \mathcal{X}$ *and* θ *is a VR-term with variables from* $\mathcal{X}$*.*

Given two terms ξ and η, a derivation step $\xi \rightsquigarrow_\Gamma \eta$ means that η is obtained from ξ by replacing an occurrence of a non-terminal X with the term θ for some rule $X \hookrightarrow \theta$ from Γ. A X-derivation is then a sequence of steps starting with the non-terminal X. The derivation is complete if it ends with a ground term (a term is ground if it has no variable occurrences). This allows us to define the language of a grammar $\mathcal{L}(\Gamma)$ which is the set of clique-and-cloud topologies defined by $\{\Theta^{\mathtt{VR}} \mid X_0 \rightsquigarrow_\Gamma^* \Theta \text{ is a complete } X_0\text{-derivation}\}$.

Example 3. We present a grammar which generates clique-and-cloud topologies in form of a star with a central cloud vertex of process type P_2 and adjacent

clique vertices around it of process type P_1. The grammar is defined as $\Gamma_{Star} = (\{X_0, S, C, O\}, \{\pi_1, \pi_2\}, \mathcal{R}, X_0)$ and the rules of the grammar are given by:

$$\begin{array}{ll} X_0 \hookrightarrow edg_{\pi_1,\pi_2}(S) & C \hookrightarrow \bullet\pi_2(P_2, \therefore) \\ S \hookrightarrow (S \oplus O) & O \hookrightarrow \bullet\pi_1(P_1, \Delta) \\ S \hookrightarrow (C \oplus O) & \end{array}$$

The clique-and-cloud topology depicted on the right of Figure 2 belongs to $\mathcal{L}(\Gamma_{Star})$ and it corresponds to the term: $edg_{\pi_1,\pi_2}(((\bullet\pi_2(P_2, \therefore) \oplus \bullet\pi_1(P_1, \Delta)) \oplus \bullet\pi_1(P_1, \Delta)) \oplus \bullet\pi_1(P_1, \Delta))$

3.3 Safety Problem for Topologies Defined by Grammars

We can now present the safety verification problem we are interested in. It consists in checking, given a VR-graph grammar and a control state q, whether there is a network topology G belonging the the expansion of a clique-and-cloud topology in the language of the grammar such that there exists an execution in G where a process leads the control state q. This problems is connected to a safety question, because if q is an error state and the answer to the problem is negative, it means that in all the clique-and-cloud topologies obtained from the grammar, the error state can never be reached. We move to the formal definition of the control state reachability problem which is parameterized by a network transition relation $\Rightarrow$ for the broadcast (reliable vs unreliable):

`CSReach`($\Rightarrow$)

Input: A VR-graph grammar Γ and a control state q

Question: Does there exist $\mathcal{G} \in L(\Gamma)$ and $G \in [\![\mathcal{G}]\!]$ and $c \in \mathcal{C}_G$ such that: $\#_c(q) > 0$ and $c_{in}^G \Rightarrow^* c$?

4 Undecidability for Reliable Communication

We prove that the parametric safety problem is undecidable for clique-and-cloud systems with reliable communication. In [9], this problem was proven to be undecidable, without any restriction on the considered topologies, i.e. given a single process type and a state, does there exist a network topology where a process can reach the given state? However, if one considers the networks to be cliques, the problem is known to be decidable [12]. This is what motivates us to look at network topologies generated by clique-and-cloud topologies. The undecidability proof presented in [9] relies on two ingredients, first the authors show it is possible to 'extract' a line from a network topology (here extracting means that the other vertices are sent in an error state from which they cannot perform any actions any more) and then they explain how to simulate the behavior of a Minsky machine with two counters (programs that manipulate two natural variables, which can be only incremented, decremented and compared to 0) on lines of unbounded size. Our undecidability proof uses similar techniques.

4.1 Leader Election in Clique-and-Cloud Topologies

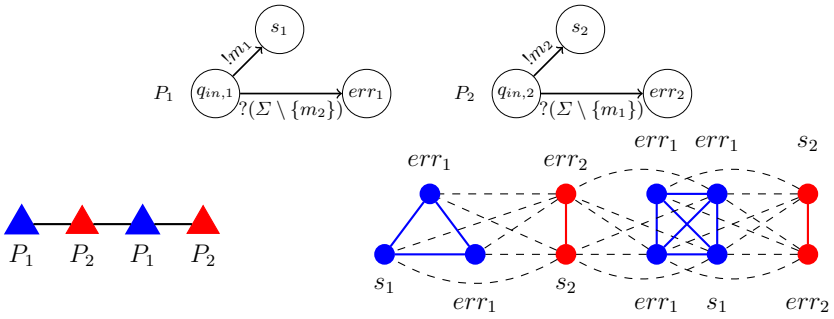

Fig. 4: Two process-types, a cloud-and-clique toplogy in $\mathcal{L}(\Gamma_L)$ and one of its expansion

We first give the intuition on how one can extract a line of single vertices with different process types, from a line of cliques. We consider the two process types given in Figure 4 and a grammar which produces lines of clique vertices, where the process associated to each vertex alternates between P_1 and P_2. The grammar in question is $\Gamma_L = (\{L_0\}, \{\pi_1, \pi_2, \pi_3, \pi_4, \pi_5\}, \mathcal{R}, L_0)$, having the following rules:

$$L_0 \hookrightarrow edg_{\pi_1,\pi_2}(\bullet\pi_1(P_1, \Delta) \oplus \bullet\pi_2(P_2, \Delta))$$
$$L_0 \hookrightarrow relab_{[\pi_3 \mapsto \pi_5, \pi_4 \mapsto \pi_5]}(edg_{\pi_3,\pi_4}(relab_{[\pi_2 \mapsto \pi_3]}(L_0) \oplus relab_{[\pi_1 \mapsto \pi_4]}(L_0)))$$

One can see that the language $\mathcal{L}(\Gamma_L)$ is composed by clique-and-cloud topologies of the following form for some $m \geq 1$: $\mathcal{G} = (\{v_i\}_{1 \leq i \leq 2m}, \mathcal{E}, Lab, \mathcal{V}_\Delta, \mathcal{V}_\cdot)$ with $\mathcal{E} = \{(v_i, v_{i+1}), (v_{i+1}, v_i) \mid 1 \leq i \leq 2m-1\}$, $Lab(v_i)$ is equal to P_1 if i is odd and to P_2 otherise, and $\mathcal{V}_\Delta = \{v_i\}_{1 \leq i \leq 2m}$. The left-hand side of Figure 4 provides an example of such a topology with $m = 2$ and on the right is depicted one of its expansion. On this latter network topology, we have indicated control states to represent a possible configuration reachable from the corresponding initial configuration with reliable broadcast. One specificity of the reachable configurations (with the reliable semantics) is that for each clique component in the clique-and-cloud topology, there is at any moment at most one vertex in state s_1 or s_2 and if such a vertex is present, the other vertices of the same clique are in the state err_1 or err_2. Indeed, to reach such a state the corresponding process needs to broadcast the message m_1 or m_2 sending all its neighbors with the same process type into the error state err_1 or err_2. Note that if we consider the unreliable semantics, this property will not hold anymore, and we could find two vertices in the same clique labeled for instance with s_1. If we consider that the vertices in states err_1 and err_2 will not participate in any further communication, and we extend the process types starting from s_1 or s_2, this means that we are able,

in a certain sense, to extract a single line from any clique-and-cloud topology where process types of type P_1 and P_2 alternate. While this method of leader selection is only viable for clique topologies, we present a second method in the full proof in the Appendix which deals with clique-and-cloud topologies.

4.2 Simulating a Minsky Machine

As said before, our undecidability result comes from the possibility to simulate the behavior of a (two-counter) Minsky machine with line topologies of unbounded length. A (two-counter) Minsky machine is a program which manipulates two natural variables (or counters) c_1 and c_2, and which is composed of a finite set of instructions. Each of the instruction is either of the form (1) $L : c_i := c_i + 1$; `goto` L' or (2) L : `if` $c_i = 0$ `then goto` L' `else` $c_i := c_i - 1$; `goto` L'' where $i \in \{1, 2\}$ and L, L', L'' are labels preceding each instruction. Furthermore, there is a special label L_F from which nothing can be done. The behavior of each instruction is the expected one. The *halting problem* of deterministic Minsky machines, which consists in deciding whether the unique execution that starts from L_0 with counters equal to 0 reaches L_F, is undecidable [19].

Fig. 5: Network topology to simulate a run of a 2-counter Minsky machine

Our proof follows the same line as the one presented in [9]. We use a grammar which builds lines of unbounded length as the one presented in Figure 5. Using a technique similar to the one presented before we extract a single leader process in each clique vertex, and then we simulate the counter machine. The process type Ctl, called the controller, simulates the instructions of the Minsky machines and the processes of type K_i encode the value of the counter c_i thanks to specific states Z and NZ. Intuitively, the number of processes in states NZ represent the value of the counter and all the processes in the 'extracted line' from the controller to the first node in state NZ are in state Z. To increase a counter, a message is transmitted to the last vertex in Z who changes its state to NZ and sends back an acknowledgment to the controller. To decrement the counter, we proceed the same way but this time, it is the last node to be in state NZ (which has a neighbor in state Z) which moves to Z. Finally, to test if the counter c_i is 0, the controller checks (thanks to an exchange of messages) whether its first neighbor of process type K_i is in state Z. One key idea of this reduction is that if there are not enough processes in the line to simulate the counter value, the simulation is blocked. However we have that the Minsky machine halts iff there exists a line (big enough) where the controller ends in the state L_f. The detailed construction is provided in Appendix.

Theorem 1. `CSReach`$(\Rightarrow_r)$ *is undecidable.*

5 Describing Reachable States in MSO

The previous negative result motivates the search for sufficient conditions that guarantee decidability of the safety problem, without jeopardizing the expressivity of the model we consider. In this section, we show that, under certain assumptions, we are able to build a formula of the Monadic Second Order logic (MSO) that characterizes the local control states which can be reached by an execution of some network topology obtained by the expansion of a given clique-and-cloud topology.

5.1 Monadic Second Order Logic for Clique-and-Cloud Topologies

We begin by giving the definition of MSO, adapted to our context. Monadic Second Order (MSO) logic is the set of formulae built inductively, from a set var of first-order variables and a set Var of set variables, according to the syntax:

$$\varphi ::= \varphi \wedge \varphi \mid \neg\varphi \mid \exists x.\varphi \mid \exists X.\varphi \mid X(y) \mid E(x,y) \mid P_i(x) \mid \Delta(x) \mid \therefore(x)$$

where $x, y \in \mathit{var}$, $X \in \mathit{Var}$, $\wedge$ and $\neg$ are the classical boolean operations for conjunction and complement, $X(y)$ is the unary predicate for membership $E(x,y)$ is the binary predicate for edges and, $P_i(x)$ (with $P_i \in \mathcal{P}$) is the labeling predicate for process type and $\Delta(x)$ and $\therefore(x)$ are the labeling predicates for expansion types. Additionally, we assume classical shorthand notations for $\vee$ (disjunction), $\Longrightarrow$ (implication), $\forall v.\varphi$ and $\forall V.\varphi$ (universal quantification). As usual, we call *sentence* a formula with no free occurrence of first order or second order variables (all variables are in the scope of some quantifiers).

The MSO formulae we consider in this paper are then interpreted over clique-and-cloud topologies. A variable store for a clique-and-cloud topology $\mathcal{G}$ is a mapping ν from first order (resp. set) variables into vertices (resp. sets of vertices) of $\mathcal{G}$. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathit{Lab}, \mathcal{V}_\Delta, \mathcal{V}_\therefore)$ be a clique-and-cloud topology, we define the satisfaction relation $\models$ parameterized by a store inductively as follows (boolean cases are omitted):

$$\begin{array}{lll} \mathcal{G} \models_\nu E(x,y) & \iff & (\nu(x),\nu(y)) \in \mathcal{E} \\ \mathcal{G} \models_\nu \Delta(x) & \iff & \nu(x) \in \mathcal{V}_\Delta \\ \mathcal{G} \models_\nu \therefore(x) & \iff & \nu(x) \in \mathcal{V}_\therefore \end{array} \qquad \begin{array}{lll} \mathcal{G} \models_\nu X(x) & \iff & \nu(x) \in \nu(X) \\ \mathcal{G} \models_\nu P_i(x) & \iff & \mathit{Lab}(\nu(x)) = P_i \text{ for } P_i \in \mathcal{P} \\ \mathcal{G} \models_\nu \exists x.\varphi & \iff & \mathcal{G} \models_{\nu[x \mapsto u]} \varphi \text{ for some } u \in \mathcal{V} \\ \mathcal{G} \models_\nu \exists X.\varphi & \iff & \mathcal{G} \models_{\nu[X \mapsto \mathcal{U}]} \varphi \text{ for some } \mathcal{U} \subseteq \mathcal{V} \end{array}$$

where $\nu[x \mapsto u]$ (resp. $\nu[X \mapsto \mathcal{U}]$) represents the store that agrees with ν everywhere except for the variable x (resp. X) which is mapped to the vertex u (resp. the set of vertices $\mathcal{U}$). If the MSO formula φ is a sentence, the store is not important, hence we omit it. In this case, we write $[\![\varphi]\!] \stackrel{\text{def}}{=} \{\mathcal{G} \mid \mathcal{G} \models \varphi\}$ for the set of models of a sentence φ. We point out that considering MSO with vertex quantifiers[3] is crucial for the following decidability theorem (Theorem 2).

In our context, MSO formulae appear as a very useful tool because we are able to decide whether there exists a clique-and-cloud topology in the language of

[3] Usually also denoted MSO_1 in the literature [7].

a grammar which satisfies a given formula. This is thanks to a result of Courcelle and Engelfriet, which states that given a VR-graph grammar Γ and an MSO sentence φ, it is possible to build a "refined" grammar Γ_φ that produces the set of graphs from the language of Γ that, moreover, satisfy φ, see, e.g., [7, Theorem 1.22]. Since emptiness of a grammar is decidable in polynomial time (in the size of the input grammar), we obtain the following theorem:

Theorem 2. *The following problem is decidable:* given a graph grammar Γ written in the VR-algebra and an MSO sentence φ, does $\mathcal{L}(\Gamma) \cap [\![\varphi]\!] = \emptyset$?

5.2 Reachable States Labeling of Clique-and-Cloud Topologies

In the rest of the section, let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, Lab, \mathcal{V}_\Delta, \mathcal{V}_{\cdot\cdot})$ be a fixed clique-and-cloud topology and $\Rightarrow$ a network transition relation (i.e., a relation between network configurations, defined by either the reliable or unreliable semantics). We recall that Lab maps each vertex in $\mathcal{V}$ to a process type in $\mathcal{P}$, where $\mathcal{P} = \{P_1, \ldots, P_\ell\}$, $P_i = (Q_i, q_{in,i}, \delta_i)$, for all $i \in [1, \ell]$ and $Q = \biguplus_{1 \leq i \leq \ell} Q_i$ and $\delta = \biguplus_{1 \leq i \leq \ell} \delta_i$ denote the overall set of states and transition relation, respectively.

A *states-labeling* of $\mathcal{G}$ is a function $\gamma : \mathcal{V} \mapsto 2^Q$, that associates each vertex to a subset of Q. In particular, we are interested in a specific states labeling, namely the *reachable states-labeling for* $\mathcal{G}$ $\gamma^{\Rightarrow}_{reach} : \mathcal{V} \mapsto 2^Q$, defined as follows:

Definition 5 (Reachable states-labeling). *For all $v \in \mathcal{V}$, we have $q \in \gamma^{\Rightarrow}_{reach}(v)$ if and only if there exist $(G, f) \in [\![\mathcal{G}]\!]$, $u \in f^{-1}(v)$ and $c \in \mathcal{C}_G$ verifying the following conditions: (1) $c(u) = q$ and (2) $c^G_{in} \Rightarrow^* c$.*

The next lemma, which can be directly proven by applying the definition, shows the connection between the reachable states-labeling and the control state reachability problem for the clique-and-cloud topology $\mathcal{G}$.

Lemma 1. *Let $q \in Q$ be a state. Then, there exists $G \in [\![\mathcal{G}]\!]$ and $c \in \mathcal{C}_G$ such that $\#_c(q) > 0$ and $c^G_{in} \Rightarrow^* c$ iff there exists $v \in \mathcal{V}$ such that $q \in \gamma^{\Rightarrow}_{reach}(v)$*

In the sequel, we will show that, under certain conditions, we can build an MSO formula which characterizes the clique-and-cloud topology whose reachable states labeling exhibit a given control state q. By Theorem 2, this provides a general argument for the decidability of the `CSReach`$(\Rightarrow)$ problem.

5.3 Two Sufficient Criteria

We state here the conditions we rely on to establish our decidability results. We begin by introducing some new notations. Given a clique-and-cloud topology $\mathcal{G} = (\mathcal{V}, \mathcal{E}, Lab, \mathcal{V}_\Delta, \mathcal{V}_{\cdot\cdot})$ and two disjoint network topologies $G_1 = (V_1, E_1, lab_1)$ and $G_2 = (V_2, E_2, lab_2)$ such that $(G_1, f_1), (G_2, f_2) \in [\![\mathcal{G}]\!]$, we denote by $G_1 \uplus G_2$ the network topology (V, E, lab) where: $V \stackrel{\text{def}}{=} V_1 \uplus V_2$ and $E \stackrel{\text{def}}{=} E_1 \uplus E_2 \uplus \{(v, v') \in V_i \times V_{3-i} \mid f_i(v) = f_{3-i}(v'),\ f_i(v) \in \mathcal{V}_\Delta,\ i = 1, 2\}$ and $lab(v) \stackrel{\text{def}}{=} lab_i(v)$ for all $v \in V_i$ and all $i \in \{1, 2\}$. Note that, by construction, we have the following:

Lemma 2. *If $f : V \mapsto \mathcal{V}$ is the surjective function such that $f(v) = f_1(v)$ for all $v \in V_1$ and $f(v) = f_2(v)$ for all $v \in V_2$ then $(G_1 \uplus G_2, f) \in [\![\mathcal{G}]\!]$.*

Finally, given two configurations $c_1 \in \mathcal{C}_{G_1}$ and $c_2 \in \mathcal{C}_{G_2}$, we denote by $c_1 \uplus c_2$ the configuration in $\mathcal{C}_{G_1 \uplus G_2}$ such that $c_1 \uplus c_2(v) = c_1(v)$ for all $v \in V_1$ and $c_1 \uplus c_2(v) = c_2(v)$ for all $v \in V_2$. We say that a state $q \in Q$ is an *emitter* state, whenever there exists $m \in \Sigma$ and $q' \in Q$ such that $(q, !m, q') \in \delta$ and denote by $Q_E \subseteq Q$ the set of emitter states. The first criterion is stated below. We say that the network transition relation $\Rightarrow$ respects the *composition property for emitters* when it satisfies the following statement, for all clique-and-cloud topologies $\mathcal{G}$:

Definition 6 (Composition property for emitters). *For all $G_1, G_2 \in [\![\mathcal{G}]\!]$, if $c_i \in \mathcal{C}_{G_i}$ are configurations, where $c_{in}^{G_i} \Rightarrow^* c_i$, $i = 1, 2$, then there exists a configuration $c_1' \in \mathcal{C}_{G_1}$ verifying $c_{in}^{G_1 \uplus G_2} \Rightarrow^* (c_1' \uplus c_2)$ and $c_1'(v) = c_1(v)$, for all vertices v of G_1, such that $c_1(v) \in Q_E$.*

Intuitively this property expresses that if two configurations are reachable then we can reach another configuration where all the emitters states (resp. all the states) of the first (resp. second) configuration are present. This allows us to compose configurations on demand to compute reachable states. Even if not stated this way, this property is the key ingredient of the polynomial time algorithm for safety under the unreliable semantics proposed in [8] and a similar property has been established for wait-only protocols under the reliable semantics in [15].

The second criterion is a condition on when control state of a vertex can change when taking one step of the network transition relation $\Rightarrow$. We say that the network transition relation $\Rightarrow$ respects the *local successor property* whenever if satisfies the following statements, for all clique-and-cloud topologies $\mathcal{G}$:

Definition 7 (Local successor property). *For all $G = (V, E, lab) \in [\![\mathcal{G}]\!]$, $c \in \mathcal{C}_G$ and $v \in V$, there exists $c' \in \mathcal{C}_G$ such that $c \Rightarrow c'$ and $c(v) \neq c'(v)$ if and only if there exists $m \in \Sigma$, such that one of the following properties holds:*
(1) $c(v) \xrightarrow{!m}_\delta c'(v)$, or,
(2) $c(v) \xrightarrow{?m}_\delta c'(v)$ and there exists $u \in V$ and $q \in Q$ such that $(u, v) \in E$ and $c(u) \xrightarrow{!m}_\delta q$.

Relying on the definitions of the reliable and unreliable semantics, we can prove the following lemma:

Lemma 3. *The network transition relations $\Rightarrow_r$ (reliable communication) and $\Rightarrow_u$ (unreliable communication) both meet the local successor property.*

Note that whereas we prove that the unreliable semantics respects the composition property for emitters, this does not hold for the reliable semantics, and we will see that we need another restriction in this context.

5.4 Characterization of the Reachable States-labeling

We now provide a characterization of the reachable states labeling of clique-and-cloud topologies for the network transitions relations respecting both the

composition property for emitters (Definition 6) and the local successor property (Definition 7). Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, Lab, \mathcal{V}_\Delta, \mathcal{V}_{\cdot\cdot})$ be a clique-and-cloud topology and $\Rightarrow$ be a transition relation. We define the following property of states-labelings:

Definition 8 (Inductive reachability property). *A states-labeling $\gamma : \mathcal{V} \mapsto 2^Q$ meets the* inductive reachability property *for $\Rightarrow$ if and only if the following hold, for all $v, v' \in \mathcal{V}$ and $q, q', q'', q''' \in Q$:*

1. *if $Lab(v) = P_i$ then $q_{in,i} \in \gamma(v)$,*
2. *if $q \in \gamma(v)$ and $q \xrightarrow{!m}_\delta q'$ then $q' \in \gamma(v)$,*
3. *if $v \in \mathcal{V}_\Delta$ and $q, q' \in \gamma(v)$, $q \xrightarrow{!m}_\delta q''$ and $q' \xrightarrow{?m}_\delta q'''$ then $q''' \in \gamma(v)$,*
4. *if $q \in \gamma(v)$, $q' \in \gamma(v')$, $(v, v') \in \mathcal{E}$, $q \xrightarrow{!m}_\delta q''$ and $q' \xrightarrow{?m}_\delta q'''$ then $q''' \in \gamma(v')$.*

Furthermore, given states-labelings γ and γ', we write $\gamma \sqsubseteq \gamma'$ for the pointwise inclusion of γ into γ'. We state below an important lemma relating the reachable states-labeling γ_{reach} and the inductive reachability property:

Lemma 4. *If a network transition relation $\Rightarrow$ respects both the composition property for emitters and the local successor property, then $\gamma^{\Rightarrow}_{reach}$ is the least states-labeling (in the sense of $\sqsubseteq$) having the inductive reachability property.*

Proof. Let $\Rightarrow$ be a network transition relation respecting both the composition property for emitters and the local successor property and let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, Lab, \mathcal{V}_\Delta, \mathcal{V}_{\cdot\cdot})$ be a clique-and-cloud topology. We recall that we assumed $\mathcal{P} = \{P_1, \ldots, P_\ell\}$ where $P_i = (Q_i, q_{in,i}, \delta_i)$ for all $i \in [1, \ell]$ and that we denote $Q \stackrel{\text{def}}{=} \bigcup_{1 \leq i \leq \ell} Q_i$ and $\delta = \biguplus_{1 \leq i \leq \ell} \delta_i$. We fist show that $\gamma^{\Rightarrow}_{reach}$ verifies each case of the inductive reachability property.

1. Let $v \in \mathcal{V}$ and assume $Lab(v) = P_i$. Consider the network topology $G = (\mathcal{V}, \mathcal{E}, Lab)$. Note that $(G, id) \in [\![\mathcal{G}]\!]$ where id is the identity function (which is surjective). By definition of c^G_{in}, we have $c^G_{in}(v) = q_{in,i}$. Since $id^{-1}(v) = v$ and since $c_{in} \Rightarrow^* c_{in}$, applying the definition of $\gamma^{\Rightarrow}_{reach}$, we deduce $q_{in,i} \in \gamma^{\Rightarrow}_{reach}(v)$.
2. Let $v \in \mathcal{V}$ and assume that $q \in \gamma^{\Rightarrow}_{reach}(v)$ and that there exists $q \xrightarrow{!m}_\delta q'$. We assume that $q \neq q'$, otherwise the property is obviously satisfied. By definition of $\gamma^{\Rightarrow}_{reach}$, there exist $(G, f) \in [\![\mathcal{G}]\!]$ and a vertex $v' \in f^{-1}(v)$ and a configuration $c \in \mathcal{C}_G$ such that $c(v') = q$ and $c^G_{in} \Rightarrow c$. Since we have $(q \xrightarrow{!m}_\delta q')$, using the fact that $\Rightarrow$ respects the local successor property, we deduce there exists a configuration c' such that $c \Rightarrow c'$ and $c'(v') = q'$. Hence, we have as well $c^G_{in} \Rightarrow c'$. We conclude that $q' \in \gamma^{\Rightarrow}_{reach}(v)$.
3. Let $v \in \mathcal{V}$ and assume that there exist $q, q' \in \gamma^{\Rightarrow}_{reach}(v)$ and $v \in \mathcal{V}_\Delta$ and $q \xrightarrow{!m}_\delta q''$ and $q' \xrightarrow{?m}_\delta q'''$. By definition of $\gamma^{\Rightarrow}_{reach}$, there exist $(G_1, f_1) \in [\![\mathcal{G}]\!]$ and a vertex $v_1 \in f_1^{-1}(v)$ and a configuration $c_1 \in \mathcal{C}_{G_1}$ such that $c_1(v_1) = q$ and $c^{G_1}_{in} \Rightarrow^* c_1$. Similarly, there exist $(G_2, f_2) \in [\![\mathcal{G}]\!]$ and a vertex $v_2 \in f_2^{-1}(v)$ and a configuration $c_2 \in \mathcal{C}_{G_2}$ such that $c_2(v_2) = q'$ and $c^{G_2}_{in} \Rightarrow^* c_2$. We now show that $q''' \in \gamma^{\Rightarrow}_{reach}(v)$ (assuming $q''' \neq q'$ otherwise, the result is obvious).

We consider the network topology $G = G_1 \uplus G_2$ with $G = (V, E, lab)$. Thanks to Lemma 2, we have $(G, f) \in [\![\mathcal{G}]\!]$ (where f is the function given in the lemma). Since $\Rightarrow$ respects the composition property for emitters and since q is an emitter state, we deduce that there exists a configuration $c_1' \in \mathcal{C}_{G_1}$ verifying $c_{in}^G \Rightarrow^* (c_1' \uplus c_2)$ and $c_1'(v_1) = c_1(v_1) = q$. We denote by c the $(G_1 \uplus G_2)$-configuration $c_1' \uplus c_2$. By definition of $\uplus$, we have hence $c(v_1) = q$ and $c(v_2) = q'$. Furthermore, by definition of (G, f), since $f(v_1) = f_1(v_1) = f_2(v_2) = f(v_2) = v$ and $v \in \mathcal{V}_\Delta$, we have $(v_1, v_2) \in E$. Using the fact that $\Rightarrow$ respects the local successor property, we deduce there exists a configuration c' such that $c \Rightarrow c'$ and $c'(v_2) = q'''$. Hence, we have as well $c_{in}^G \Rightarrow^* c'$. We conclude that $q''' \in \gamma^{\Rightarrow}_{reach}(v)$.

4. Let $v, v' \in \mathcal{V}$ such that $(v, v') \in \mathcal{E}$ and assume that there exist $q \in \gamma^{\Rightarrow}_{reach}(v)$ and $q' \in \gamma^{\Rightarrow}_{reach}(v')$ and $q \xrightarrow{!m}_\delta q''$ and $q' \xrightarrow{?m}_\delta q'''$. By definition of $\gamma^{\Rightarrow}_{reach}$, there exist $(G_1, f_1) \in [\![\mathcal{G}]\!]$ and a vertex $v_1 \in f_1^{-1}(v)$ and a configuration $c_1 \in \mathcal{C}_{G_1}$ such that $c_1(v_1) = q$ and $c_{in}^{G_1} \Rightarrow^* c_1$. Similarly, there exist $(G_2, f_2) \in [\![\mathcal{G}]\!]$ and a vertex $v_2 \in f_2^{-1}(v)$ and a configuration $c_2 \in \mathcal{C}_{G_2}$ such that $c_2(v_2) = q'$ and $c_{in}^{G_2} \Rightarrow^* c_2$. To prove that $q''' \in \gamma^{\Rightarrow}_{reach}(v')$, we apply the same reasoning as the previous case, the main difference being that we have here $(v_1, v_2) \in E$ because $(f(v_1), f(v_2)) = (v, v') \in \mathcal{E}$.

Consequently, $\gamma^{\Rightarrow}_{reach}$ respects the inductive reachability property.

We consider now a states labeling γ respecting the inductive reachability property, and we show that $\gamma^{\Rightarrow}_{reach} \sqsubseteq \gamma$. For this matter, we define the family $(\gamma_n)_{n \in \mathbb{N}}$ of states labelings as follows: for all $n \in \mathbb{N}$, for all $v \in \mathcal{V}$, we have $q \in \gamma_n(v)$ iff there exists $G = (V, E, lab)$ such that $(G, f) \in [\![\mathcal{G}]\!]$ and there exists $v' \in f^{-1}(v)$ and $c \in \mathcal{C}_G$ verifying the following conditions:

1. $c(v') = q$, and,
2. $c_{in} \Rightarrow^m c$ for $0 \leq m \leq n$ (where $\Rightarrow^0$ is the identity relation and $\Rightarrow^k$ is the relation $\Rightarrow \cdot \Rightarrow^{k-1}$ for all $k > 1$)

We show by induction that $\gamma_n \sqsubseteq \gamma$.

- We first show that $\gamma_0 \sqsubseteq \gamma$. Let $v \in \mathcal{V}$ and q be a control state in $\gamma_0(v)$. We prove that $q \in \gamma(v)$. By definition of γ_0, there exists $(G, f) \in [\![\mathcal{G}]\!]$ with $G = (V, E, lab)$ and a vertex $v' \in f^{-1}(v)$ such that $c_{in}^G(v') = q$. Assume now that $lab(v') = P_i$. In that case we have $lab(v') = P_i$ and $q = q_{in,i}$. By definition of $[\![\mathcal{G}]\!]$, we have $Lab(v) = Lab(f(v')) = lab(v') = P_i$. Since γ respects the inductive reachability property (in particular statement 1), we deduce that $q = q_{in,i}$ belongs to $\gamma(v)$. Hence, we have $\gamma_0 \sqsubseteq \gamma$.
- We now move to the inductive step. Let $n \in \mathbb{N}$ and assume that $\gamma_n \sqsubseteq \gamma$. We show that $\gamma_{n+1} \sqsubseteq \gamma$. Let $v \in \mathcal{V}$ and q be a control state in $\gamma_{n+1}(v)$. First note that if $q \in \gamma_n(v)$, then using the induction hypothesis we obtain directly that $q \in \gamma(v)$, we assume hence that $q \notin \gamma_n(v)$. By definition of γ_{n+1}, there exists $(G, f) \in [\![\mathcal{G}]\!]$ with $G = (V, E, lab)$ and a vertex $v' \in f^{-1}(v)$ and $c \in \mathcal{C}$ and $m \leq (n+1)$ such that $c_m(v') = q$ and $c_{in}^G \Rightarrow^m c_m$. Note that since

$q \notin \gamma_n(v)$, we know that $m = n + 1$ and there must exist a configuration $c' \in \mathcal{C}$ such that $c^G_{in} \Rightarrow^n c' \Rightarrow c$ with $c'(v') = q'$ (where $q \neq q'$). By definition we have $q' \in \gamma_n(v)$ and thanks to induction hypothesis, we deduce $q' \in \gamma(v)$. Now since $c' \Rightarrow c$, and since $\Rightarrow$ respects the local successor property, we can perform the following case analysis:

1. Either, $q' \xrightarrow{!m}_\delta q$, in that case since $q' \in \gamma(v)$ and since γ respects the inductive reachability property (in particular statement 2.), we deduce that $q \in \gamma(v)$.
2. Or $q' \xrightarrow{?m}_\delta q$ and there exists $u \in V$ and $q'' \in Q$ such that $f(u) = f(v')$ and $(u, v') \in E$ and $c(u) \xrightarrow{!m}_\delta q''$. First note that since $f(u) = f(v') = v$ and $(u, v') \in E$, by definition of $(G, f) \in [\![\mathcal{G}]\!]$, we have $v \in \mathcal{V}_\Delta$. Then the same way, we show that $q' = c'(v') \in \gamma(v)$, we can show that $c'(u) \in \gamma^n(v) \subseteq \gamma(v)$. Since γ respects the inductive reachability property (in particular statement 3.), we deduce that $q \in \gamma(v)$.
3. Or $q' \xrightarrow{?m}_\delta q$ and there exists $u \in V$ and $q'' \in Q$ such that $f(u) \neq f(v')$ and $(u, v') \in E$ and $c(u) \xrightarrow{!m}_\delta q''$. First note that since $f(u) \neq f(v')$ and $(u, v') \in E$, by definition of $(G, f) \in [\![\mathcal{G}]\!]$, we have $(f(u), v) \in \mathcal{E}$. Then the same way we show that $q' = c'(v') \in \gamma(v)$, we can show that $c'(u) \in \gamma^n(f(u)) \subseteq \gamma(f(u))$. Since γ respects the inductive reachability property (in particular statement 4.), we deduce that $q \in \gamma(v)$.

Hence, we obtain $\gamma_{n+1} \sqsubseteq \gamma$.

We have then proved that $\gamma_n \sqsubseteq \gamma$ for all $n \in \mathbb{N}$. By applying the definition, we have that $\gamma^{\Rightarrow}_{reach}(v) = \bigcup_{n\in\mathbb{N}} \gamma_n(v)$ for all $v \in \mathcal{V}$. Hence, we have $\gamma_{reach}(v) \subseteq \gamma(v)$ for all $v \in \mathcal{V}$, and consequently $\gamma_{reach} \sqsubseteq \gamma$. □

5.5 Defining the Reachable States-Labeling in MSO

In this section, we consider a network transition relation $\Rightarrow$ meeting both the composition property for emitters (Definition 6) and the local successor property (Definition 7). We define the equivalent property of the reachable states-labeling stated in Lemma 4 using an MSO formula. More specifically, assuming w.l.o.g. an enumeration $Q = \{q_1, \ldots, q_n\}$ of the set of states, we build an MSO formula $\Phi_{reach}(X_1, \ldots, X_n)$ where each X_i is a free set variable, such that $\mathcal{G} \models_\nu \Phi_{reach}(X_1, \ldots, X_n)$ if and only if $\nu(X_i)$ is the set of vertices v verifying $q_i \in \gamma^{\Rightarrow}_{reach}(v)$ for all $i \in [1, n]$.

First, we build an MSO formula $\Phi_{ind}(X_1, \ldots, X_n)$ to characterize the states-labelings which respect the inductive reachability property (Definition 8). This formula uses four families of formulae: $\varphi_{1,i}(x, X_1, \ldots, X_n)$, $\varphi_{2,i}(x, X_1, \ldots, X_n)$, $\varphi_{3,i}(x, X_1, \ldots, X_n)$ and $\varphi_{4,i}(x, X_1, \ldots, X_n)$, for $i \in [1, n]$, where x is a free first-order variable. Each of this family is related to one of the conditions of Definition 8 and expresses the respective condition regarding the state q_i and the vertex associated to x. Formally, for all $i \in [1, n]$, we define:

$$\varphi_{1,i}(x, X_1, \ldots, X_n) \stackrel{\text{def}}{=} \begin{cases} \text{false if } \nexists j \in [1, \ell].\ q_i = q_{in,j}, \\ P_j(x) \text{ if } q_i = q_{in,j} \end{cases}$$

$$\varphi_{2,i}(x, X_1, \ldots, X_n) \stackrel{\text{def}}{=} \bigvee_{j \in \mathcal{J}} X_j(x), \text{ where } \mathcal{J} = \{j \in [1,n] \mid q_j \xrightarrow{!m}_\delta q_i,\ m \in \Sigma\}$$

$$\varphi_{3,i}(x, X_1, \ldots, X_n) \stackrel{\text{def}}{=} \bigvee_{(k,k') \in \mathcal{K}} X_k(x) \wedge X_{k'}(x) \wedge \Delta(x), \text{ where } \mathcal{K} = \left\{ (k,k') \in [1,n] \times [1,n] \;\middle|\; \begin{array}{c} (q_k \xrightarrow{!m}_\delta q) \\ (q_{k'} \xrightarrow{?m}_\delta q_i) \\ m \in \Sigma \\ q \in Q \end{array} \right\}$$

$$\varphi_{4,i}(x, X_1, \ldots, X_n) \stackrel{\text{def}}{=} \bigvee_{(k,k') \in \mathcal{K}} \exists x'.\, X_k(x') \wedge X_{k'}(x) \wedge E(x', x)$$

$$\Phi_{ind}(X_1, \ldots, X_n) \stackrel{\text{def}}{=} \forall x.\big(\bigwedge_{i=1}^{n} (\bigvee_{s=1}^{4} \varphi_{s,i}(x, X_1, \ldots, X_n)) \implies X_i(x)\big)$$

In order to state and prove the property of this formula below, for a store ν, we define the states-labeling $\gamma_\nu(v) \stackrel{\text{def}}{=} \{q_i \in Q \mid v \in \nu(X_i)\}$, for all $v \in \mathcal{V}$. Thanks to the definition of the inductive reachability property and relying on the semantics of MSO, we deduce the following lemma.

Lemma 5. *For each clique-and-cloud topology $\mathcal{G}$ and store ν, we have $\mathcal{G} \models_\nu \Phi_{ind}(X_1, \ldots, X_n)$ if and only if γ_ν satisfies the inductive reachability property.*

From Lemma 4, we know that since the considered network transition relation respects the composition property for emitters and the local successor property, then the reachable states labeling is the smallest states labeling (with respect to $\sqsubseteq$) having the inductive reachability property. In MSO, we can state that a state-labeling is the smallest, by the following formula: $\Phi_{reach}(X_1, \ldots, X_n) = \Phi_{ind}(X_1, \ldots, X_n) \wedge \forall X'_1. \ldots . \forall X'_n. \forall x. \Phi_{ind}(X'_1, \ldots, X'_n) \wedge \bigwedge_{i=1}^{n} (X_i(x) \implies X'_i(x))$. Thanks to Lemmas 4 and 5, we obtain the following:

Lemma 6. *For each clique-and-cloud topology $\mathcal{G}$ and each store ν, we have $\mathcal{G} \models_\nu \Phi_{reach}(X_1, \ldots, X_n)$ if and only if γ_ν is the reachable states-labeling.*

5.6 General Decidability Result

Lemma 6 gives a characterization of the reachable states-labeling in MSO, when the transition relation of the network meets the criteria of Definitions 6 and 7. By Lemma 1, knowing the reachable states-labeling is key to solving the control state reachability problem. In the following, assume that the network transition relation $\Rightarrow$ respects the two criteria of subsection 5.3, Γ be a VR-graph grammar and a q be a control state (w.l.o.g. we assume $q = q_1$). We consider then the MSO formula $\Phi = \exists x. \exists X_1. \ldots . \exists X_n. \Phi_{reach}(X_1, \ldots, X_n) \wedge X_1(x)$. If we interpret this formula, it means that in the reachable states-labeling described

by $\Phi_{reach}(X_1, \ldots, X_n)$, there is at least one vertex such that q_1 is associated to this vertex. Thanks to Lemma 1, this answers the control state reachability question. Furthermore, Theorem 2 states that we can decide whether there exists a network topology $\mathcal{G}$ in the language of the grammar Γ and which satisfies Φ. This gives us a general decidability framework, which can stated as follows:

Theorem 3. *If the network transition relation $\Rightarrow$ respects the composition property for emitters (Definition 6) and the local successor property (Definition 7), then* `CSReach`$(\Rightarrow)$ *is decidable.*

6 Specific Decidability Results

6.1 The Case of Unreliable Broadcast

We show that the control state reachability for the unreliable network transition relation $\Rightarrow_u$ is decidable. Thanks to Theorem 3, it is sufficient to prove that $\Rightarrow_u$ respects the composition property for emitters and the local successor property. Note that we have already proved with Lemma 3 that this relation respects the local successor property. It remains to show the other property.

Lemma 7. *The networks transition relations $\Rightarrow_u$ respects the composition property for emitters.*

Proof. Let $\mathcal{G}$ be a clique-and-cloud topology and let $G_1, G_2 \in [\![\mathcal{G}]\!]$. Assume we have $c_1 \in \mathcal{C}_{G_1}$ such that $c_{in}^{G_1} \Rightarrow_u^* c_1$ and $c_2 \in \mathcal{C}_{G_2}$ such that $c_{in}^{G_2} \Rightarrow_u^* c_2$. We show that $c_{in}^{G_1 \uplus G_2} \Rightarrow^* (c_1 \uplus c_2)$ (this is actually a bit stronger than the composition property for emitters and this is possible thanks to the unreliable broadcast). Since we have unreliable broadcast, nodes are free to ignore incoming messages and not change state, from this we can let $G_1 \uplus G_2$ perform the sequence $c_{in}^{G_1 \uplus G_2} \Rightarrow_u^* c_1 \uplus c_{in}^{G_2}$ and have all nodes in G_2 do not receive the broadcast messages so that they stay in their initial state. Similarly, for the G_2 transition sequence, we will have all nodes in G_1 not react and then have that $c_1 \uplus c_{in}^{G_2} \Rightarrow_u^* c_1 \uplus c_2$ which concludes the proof, showing $c_{in}^{G_1 \uplus G_2} \Rightarrow^* (c_1 \uplus c_2)$. □

Consequently, we can deduce the result of Theorem 4 by Lemma 3 and Lemma 7 using Theorem 3:

Theorem 4. `CSReach`$(\Rightarrow_u)$ *is decidable.*

6.2 Reliable Broadcast with the Wait-Only Restriction

As we have seen in Section 4, the control state reachability is undecidable for the reliable network transition relation $\Rightarrow_r$. This implies that this relation does not satisfy the composition property for emitters (we know by Lemma 3, that it respects the local successor property). However, the wait-only restriction on process types introduced in [15], allows us to regain decidability in our context. This restriction states that in the considered process types, there should never be

a state from which messages can be both sent and received. Formally a process type $P = (Q, q_{in}, \delta)$ is said to be *wait-only* iff for all states $q \in Q$, there does not exist $m, m' \in \Sigma$ and $q', q'' \in Q$ such that $q \xrightarrow{!m}_\delta q'$ and $q \xrightarrow{?m'}_\delta q''$ and there does not exist $q''' \in Q$ and $m'' \in \Sigma$ such that $q_{in} \xrightarrow{?m''}_\delta q'''$. We call q an *action state* if there exists $m \in \Sigma$ and $q' \in Q$ verifying $q \xrightarrow{!m}_\delta q'$ or from which there is no outgoing transitions. Due to the restriction on q_{in}, it is always an *action state*. The other states are called *waiting states*. Finally, we say that the set of process types $\mathcal{P} = \{P_1, \ldots, P_\ell\}$ is wait-only iff P_i is wait-only for all $i \in [1, \ell]$. We now show that this restriction allows us to regain decidability.

Lemma 8. *When dealing with wait-only process types, the network transition relation $\Rightarrow_r$ respects the composition property for emitters.*

Proof. Let $\mathcal{G}$ be a clique-and-cloud topology and let $G_1, G_2 \in [\![\mathcal{G}]\!]$. Assume we have $c_1 \in \mathcal{C}_{G_1}$ such that $c_{in}^{G_1} \Rightarrow_r^* c_1$ and $c_2 \in \mathcal{C}_{G_2}$ such that $c_{in}^{G_2} \Rightarrow_r^* c_2$. We let $G_1 \uplus G_2$ perform the sequence of transitions $c_{in}^{G_1 \uplus G_2} \Rightarrow_r^* c_1 \uplus c_{in}^{G_2}$. This is sequence of transitions is possible as all vertices in $c_{in}^{G_2}$ are in an initial state, which are by definition action states from which no message can be received. Next, we let $G_1 \uplus G_2$ perform the same sequence of transitions as in $c_{in}^{G_2} \Rightarrow_r^* c_2$, this gives rise to $c_1 \uplus c_{in}^{G_2} \Rightarrow_r^* c_1' \uplus c_2$. Note that the taken transitions can have an impact only on the vertices of c_1 which are in a waiting state, but for all vertices $v \in V_1$ where $c_1(v) \in Q_E$ we know that these cannot react to a broadcast and will not move. Hence, for $v \in V_1$ if $c_1(v) \in Q_E$ we have $c_1'(v) = c_1(v)$. □

Finally, by Lemma 3 and Lemma 8 we fulfil the hypothesis of Theorem 3 and obtain the result of Theorem 5.

Theorem 5. `CSReach`$(\Rightarrow_r)$ *restricted to wait-only process types is decidable.*

7 Conclusion

We have shown in this paper, that the parametric safety problem for broadcast networks where the set of accepted topologies is given thanks to a graph grammar recognizing clique-and-cloud topologies is decidable for unreliable communication, undecidable for reliable communication and that in the latter case, decidability can be regained by imposing the wait-only restriction on the processes running in each node of the networks. Our technique to obtain decidability relies on a reduction to the model-checking problem for MSO formulae over graph grammars. First, this technique does not allow us to have precise complexity bounds and in the future we plan to find to which complexity classes belong the considered problems. Second, we also think that this general decidability procedure can be applied in different context or for other properties.

References

1. P. A. Abdulla, K. Cerans, B. Jonsson, and Yih-Kuen Tsay. General decidability theorems for infinite-state systems. In *LICS'96*, LICS '96, page 313, USA, 1996. IEEE Computer Society.
2. B. Aminof, T. Kotek, S. Rubin, F. Spegni, and H. Veith. Parameterized model checking of rendezvous systems. *Distrib. Comput.*, 31(3):187–222, June 2018.
3. Benjamin Aminof, Tomer Kotek, Sasha Rubin, Francesco Spegni, and Helmut Veith. Parameterized model checking of rendezvous systems. *Distributed Comput.*, 31(3):187–222, 2018.
4. R. Bloem, S. Jacobs, A. Khalimov, I. Konnov, S. Rubin, H. Veith, and J. Widder. *Decidability of Parameterized Verification*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2015.
5. M. Bozga, R. Iosif, A. Sangnier, and N. Villani. Counting abstraction and decidability for the verification of structured parameterized networks. In *CAV'25*, volume 15933 of *LNCS*, pages 238–262. Springer, 2025.
6. M.C. Browne, E.M. Clarke, and O. Grumberg. Reasoning about networks with many identical finite state processes. *Information and Computation*, 81(1):13 – 31, 1989.
7. Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2012.
8. G. Delzanno, A. Sangnier, R. Traverso, and G. Zavattaro. On the complexity of parameterized reachability in reconfigurable broadcast networks. In *FSTTCS'12*, volume 18 of *LIPIcs*, pages 289–300. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012.
9. G. Delzanno, A. Sangnier, and G. Zavattaro. Parameterized verification of ad hoc networks. In *CONCUR'10*, pages 313–327, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
10. G. Delzanno, A. Sangnier, and G. Zavattaro. On the power of cliques in the parameterized verification of ad hoc networks. In *FOSSACS'11*, volume 6604 of *LNCS*, pages 441–455. Springer, 2011.
11. Joost Engelfriet. *Context-Free Graph Grammars*, pages 125–213. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.
12. J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *LICS'99*, page 352, USA, 1999. IEEE Computer Society.
13. J. Esparza and M. Nielsen. Decidability issues for Petri nets - a survey. *Elektronische Informationsverarbeitung und Kybernetik*, 30:143–160, 01 1994.
14. S. M. German and A. Prasad Sistla. Reasoning about systems with many processes. *J. ACM*, 39(3):675–735, 1992.
15. L. Guillou, A. Sangnier, and N. Sznajder. Safety verification of wait-only non-blocking broadcast protocols. In *Petri Nets'24*. Springer Nature Switzerland, 2024.
16. Radu Iosif, Arnaud Sangnier, and Neven Villani. Verifying parameterized networks specified by vertex-replacement graph grammars. In Salem Lahlou and Madhavan Mukund, editors, *Networked Systems - 13th International Conference, NETYS 2025, Rabat, Morocco, May 21-23, 2025, Proceedings*, volume 15736 of *Lecture Notes in Computer Science*, pages 57–80. Springer, 2025.
17. Karthick Jayaraman, Nikolaj Bjørner, Jitu Padhye, Amar Agrawal, Ashish Bhargava, Paul-Andre C Bissonnette, Shane Foster, Andrew Helwer, Mark Kasten, Ivan Lee, Anup Namdhari, Haseeb Niaz, Aniruddha Parkhi, Hanukumar Pinnamraju,

Adrian Power, Neha Milind Raje, and Parag Sharma. Validating datacenters at scale. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 200–213, New York, NY, USA, 2019. Association for Computing Machinery.

18. D. Le Metayer. Describing software architecture styles using graph grammars. *IEEE Transactions on Software Engineering*, 24(7):521–533, 1998.
19. M. Minsky. *Computation, Finite and Infinite Machines*. Prentice Hall, 1967.

A Proof of Section 4

A.1 Building the network topology

We model the counting machine as a network topology G with processes $\mathcal{P} = \{K_1', K_1'', K_1, Ctl, K_2', K_2'', K_2\}$ in the clique-and-cloud expansion $(G, f) \in [\![\mathcal{G}]\!]$. The graph is a line of nodes shown in figure 6 with a control node Ctl in the middle and counter nodes K_i', K_i'' and K_i on each side for $i \in \{1, 2\}$. We want the Ctl node to broadcast instructions and propagate the messages through each side of the network.

Fig. 6: Network topology to simulate a run of a 2-counter Minsky machine

The class of graphs describing graphs such as the one in figure 6 is given by the following VR-graph grammar; $\Gamma_L = (\{L, P_1, P_2, P_3\}, \{\pi_1, \pi_1', \pi_2, \pi_2', \pi_3\}, \mathcal{R}, L)$ with $\mathcal{R}$ defined by the following:

$$
\begin{aligned}
L &\hookrightarrow edg_{\pi_3,\pi_2}(edg_{\pi_3,\pi_1}(P_1 \oplus P_2 \oplus P_3)) \\
&\quad \mid edg_{\pi_1,\pi_1'}(relab_{[\pi_1 \mapsto \pi_1', \pi_1' \mapsto \emptyset]}(L) \oplus P_1) \\
&\quad \mid edg_{\pi_2,\pi_2'}(relab_{[\pi_2 \mapsto \pi_2', \pi_2' \mapsto \emptyset]}(L) \oplus P_2) \\
P_1 &\hookrightarrow \bullet\pi_1(K_1, \Delta) \mid \bullet\pi_1(K_1', \Delta) \mid \bullet\pi_1(K_1'', \Delta) \\
&\quad \mid \bullet\pi_1(K_1, \therefore) \mid \bullet\pi_1(K_1', \therefore) \mid \bullet\pi_1(K_1'', \therefore) \\
P_2 &\hookrightarrow \bullet\pi_2(K_2, \Delta) \mid \bullet\pi_2(K_2', \Delta) \mid \bullet\pi_2(K_2'', \Delta) \\
&\quad \mid \bullet\pi_2(K_2, \therefore) \mid \bullet\pi_2(K_2', \therefore) \mid \bullet\pi_2(K_2'', \therefore) \\
P_3 &\hookrightarrow \bullet\pi_3(Ctl, \Delta) \mid \bullet\pi_3(Ctl, \therefore)
\end{aligned}
$$

Now, before encoding the behaviour of the counting machine instructions, we need a way of doing leader selection regardless if the node is a clique or a cloud.

A.2 Leader selection for clique-and-cloud topologies

The idea of the leader selection for clique-and-cloud graphs is similar to the intuition given in section 4 but more complex as we can have both clique and clouds. We want to propagate messages through the network, putting 1 process further and the rest in the error states and in the explanation we will not mention when a process goes into the error states as it is apparent from the figures. We will refer to the processes as q_0, $k_{1,0}'$, $k_{1,0}''$ and $k_{1,0}$ (their initial states) and write it as $q_0 - k_{1,0}' - k_{1,0}'' - k_{1,0} - k_{1,1}' - \ldots$, we will explain in detail for 1 side of the counter as the other side is identical. To accomplish this, we use the process types shown in figures 7, 8, 9 and 10. First the q_0 process will broadcast

a $!req_1^{'}$ message which is received by the next in line, $k_{1,0}^{'}$ which responds with $!ack_1^{'}$ and receives a $!ok_1^{'}$ message back. Now the $k_{1,0}^{'}$ process can broadcast $!done_1$ which moves one process to $Z_1^{'}$, and similar it will propagate $!req_1^{''}$ to the next process in line, $k_{1,0}^{''}$ which will do a similar exchange of messages, broadcasting $!done_1$ and sending a new line of messages with $!req_1$ to the next process in the line, $k_{1,0}$. Once we have done this $n+1$ times, the line will look like $q_0 - k'_{1,0} - k''_{1,0} - k_{1,0} - k'_{1,1} - \ldots k'_{1,n} - k'_{1,n+1}$ and all counters will be synchronized by $!done_1$ putting the line in the state $q_0 - Z_1^{'} - Z_1^{''} - Z_1 - \ldots$. The fact that we need $n+1$ is a technical detail of encoding the instruction behaviour and will be explained in the proof when it is needed. Next we do the exact same thing with the other side, $k_{2,0}^{'}$, $k_{2,0}^{''}$ and $k_{2,0}$ which will put q_0 in state L_0, and we have the final line: $\ldots - Z_2 - Z_2^{''} - Z_2^{'} - L_0 - Z_1^{'} - Z_1^{''} - Z_2$ The configuration of the network after leader selection will now be a line of nodes where the Ctl node is at L_0, corresponding to the first line of the Minsky machine and all counters in Z (zero), we denote the state of the network as a line $Z_{2,n+1} - \ldots - Z_{2,1} - L_0 - Z_{1,1} - \ldots - Z_{1,n+1}$ and move on to encoding the instructions of the Minsky machine that changes the states of the line.

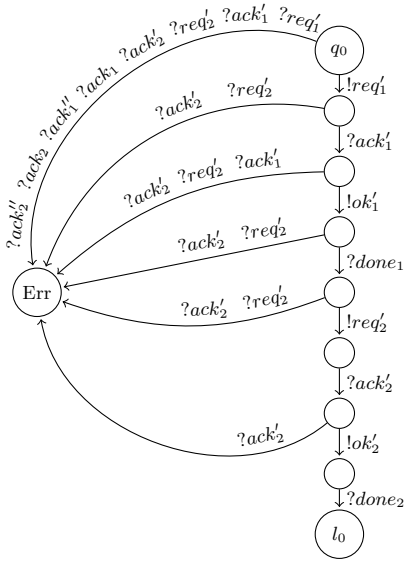

Fig. 7: Leader selection for Ctl

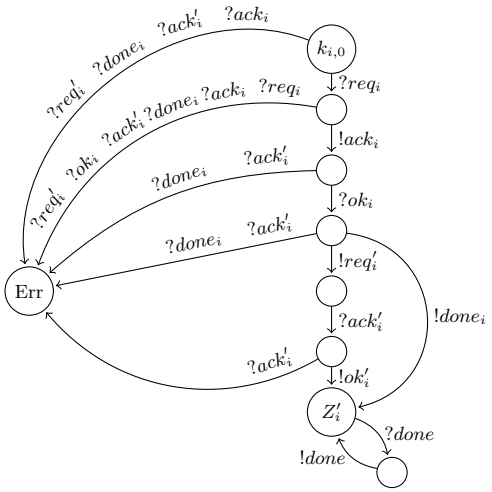

Fig. 8: Leader selection Z_i for $i \in \{1, 2\}$

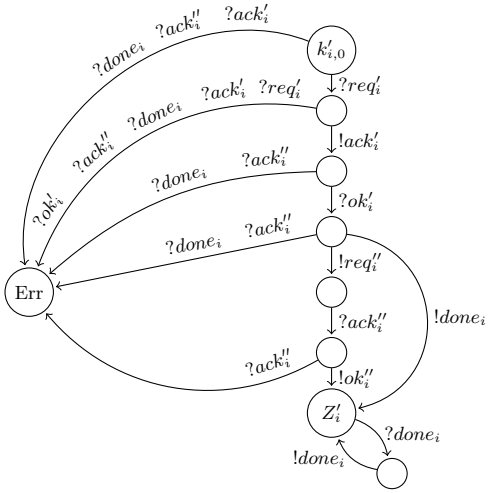

Fig. 9: Leader selection Z'_i for $i \in \{1, 2\}$

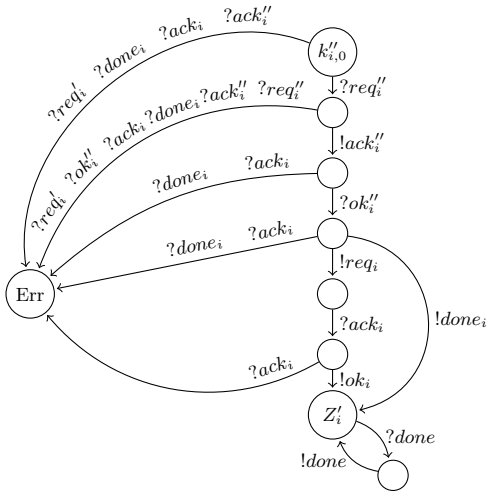

Fig. 10: Leader selection Z''_i for $i \in \{1, 2\}$

A.3 Instruction encoding

Now we will explain the encoding of instructions of the Minsky machine. We consider the figures 11, 12 and 13 like what is shown in [9]. The encoding of the increment and decrement instructions are straight forward and will be executed by the Ctl process. The controller starts in the beginning of an instruction L and sends an increment message (resp. decrement), it will broadcast an $!inc_i$ (resp. $!dec_i$) message to a counter node (K'_1 or K'_2), then it will get an $?ackinc_i$ (resp. $?ackdec_i$ or $?zero$) message back from a counter node and then the controller will move to the start of the next instruction L' (or L'' in case of the decrement instruction).

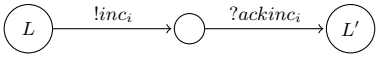

Fig. 11: Encoding of Minsky machine instruction: $L : c_i := c_i + 1;$ `goto` L'

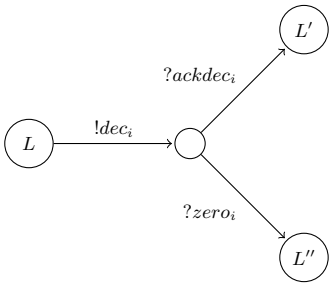

Fig. 12: Encoding of Minsky machine instruction: L: if $c_i = 0$ `then goto` L' `else` $c_i := c_i - 1$; `goto` L''

The counting behaviour shown in figure 13 for the counting nodes is more complex, since we need to propagate the messages through the network and be careful how we do so:

- If the node is in state Z and receives a decrement message dec_i, we broadcast a $zero_i$ and move back to state Z.
- If the node is in state Z and receives an increment message inc_i, we acknowledge and move to state NZ.
- If the node is in state NZ and receives an increment, we broadcast inc_i the increment to our neighbours and wait for the acknowledgement, then we broadcast our own acknowledgement and move to NZ.
- If the node is in state NZ and receives a decrement, we broadcast dec_i the decrement to our neighbours and now there are 2 cases; either the node gets a zero as response, we move to state Z, or it gets an acknowledgement as response, we broadcast our own acknowledgement and move to state NZ.

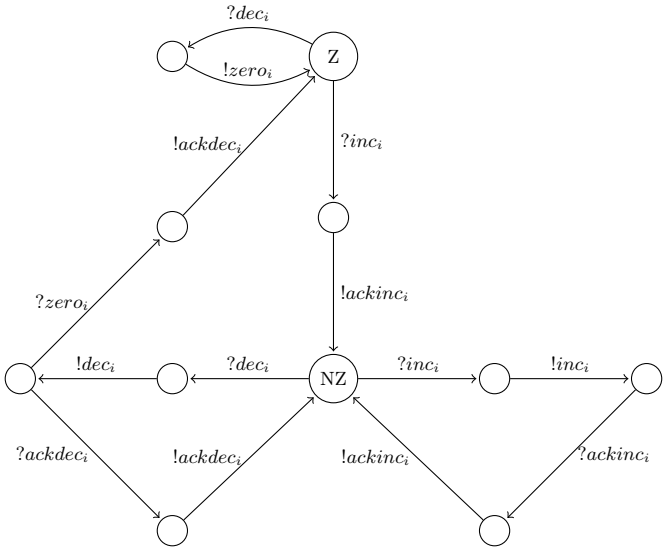

Fig. 13: The encoding of the counting behavior of the Minsky machine for counters $i \in \{1, 2\}$

A.4 Undecidability of control state reachability

Now we are ready to prove the claim made in theorem 1, and we proceed by reduction for the halting problem for (two-counter) Minsky machines.

Proof. The halting problem for Minsky machines $Halt(M)$ asks whether any Minsky machine M halts. This problem is undecidable, hence we want to show a reduction from $Halt(M)$ to $CSReach(\Rightarrow_r)$. The reduction we want to show is that for any Minsky machine we can construct a clique-and-cloud topology $G \in L(\Gamma_L)$ and show $Halt(M) \Leftrightarrow CSReach(\Rightarrow_r)$.

Given a Minsky machine M, we say the maximum value of the counters of a run is n for c_1 and n' for c_2, then to construct a clique-and-cloud topology that simulates the run we need $n+1$ nodes of K_1', K_1'' and K_1. For the other counter we need $n'+1$ nodes of K_2', K_2'' and K_2 similar to the graph presented in figure 6. For all process types we perform the leader selection by broadcasting shown in section A.2 resulting in a configuration that looks like $Z_{2,n+1} - Z_{2,n} - \ldots - Z_{2,1} - L_0 - Z_{1,1} - \ldots - Z_{1,n'} - Z_{1,n'+1}$ or, a line of nodes where all counters are set to zero and the control process at the start of the first instruction encoding. Notice that we no longer distinguish between the states Z', Z'' and Z as this was only useful for leader selection.

Now we show the reduction, first we show the first direction **(1)** $Halt(M) \Rightarrow CSReach(\Rightarrow_r)$, if the Minsky machine does an instruction L, then the network topology can perform a sequence of transitions corresponding to the instruction, in the other direction **(2)** $CSReach(\Rightarrow_r) \Rightarrow Halt(M)$ we show if the network

topology does a sequence of transitions $c \Rightarrow_r^* c'$ then the Minsky machine does the corresponding instruction $L; L'$.

(1) *$Halt(M) \Rightarrow CSReach(\Rightarrow_r)$:* Now we show that if the Minsky machine does an instruction L, then the network topology can simulate the instruction. Let the Minsky machine be at the initial instruction L_0 with both counters at 0. Let n and n' be the maximum counter value for a run in the Minsky machine and the network topology be in the state $Z_{2,n+1} - Z_{2,n} - \ldots - Z_{2,1} - L_0 - Z_{1,1} - \ldots - Z_{1,n'} - Z_{1,n'+1}$ Now, we show for both instructions that we can simulate any run of a Minsky machine. Here we only reason with a single counter that we will call c_i because extending the line with 2 counters is trivial as the other side is identical. Note that since we only have 1 counter we omit specifying the side of the line, that is we will write Z_n and NZ_n instead of $Z_{1,n}$ and $NZ_{1,n}$. For this reason we let x be the maximum value of some counter c_i (it could be n or n').

Increment Instruction: If the Minsky machine does the instruction $L : c_i := c_i + 1$; `goto` L', then we can simulate it on the network. The network simulates the increment instruction with 2 types of behaviour from figure 11 and 13 depending on the value of the counter:

- If $c_i = 0$, then we have the line $l - Z_1 - \ldots - Z_x - Z_{x+1}$ and Ctl will broadcast a $!inc_i$ message and have Z_1 acknowledge by $!ackinc_i$ and resulting in the line $L - NZ_1 - Z_2 - \ldots - Z_x - Z_{x+1}$
- If $c_i > 0$, then we have the line $L - NZ_1 - \ldots - NZ_{c_i} - Z_{c_i+1} - \ldots - Z_x - Z_{x+1}$, and Ctl will broadcast a $!inc_i$ message and the other NZ nodes will propagate the $!inc_i$ message through the network (shown by $\xrightarrow{?inc_i}$ and $\xrightarrow{!inc_i}$ in figure 13) until we reach the node $Z_{c_{i+1}}$ which will acknowledge and move to $NZ_{c_{i+1}}$ and the acknowledgement $ackinc_i$ will be propagated back through the line putting all states in NZ until it reaches Ctl.

Once Ctl receives $?ackinc_i$ and moves to the next instruction L' from figure 11 and the resulting line becomes $L' - NZ_1 - \ldots - NZ_{c_i} - NZ_{c_i+1} - Z_{c_i+2} - \ldots - Z_x - Z_{x+1}$ is clearly a simulation of the increment instruction, since the Minsky machine incremented the counter $c_i := c_i + 1$ and the network put node Z_{c_i+1} in NZ_{c_i+1} and both moved to L'.

Decrement instruction: Now we look at the instruction L: if $c_i = 0$ `then goto` L' `else` $c_i := c_i - 1$; `goto` L'' and show the network topology can simulate the instruction with 2 types of behaviour from 12 and 13 depending on the value of the counter:

- If $c_i = 0$ we have the line $l - Z_1 - \ldots - Z_{x+1}$, then Ctl will broadcast a $!dec_i$ message and the Z_0 node will react by sending back a $!zero_i$ and go back to state Z and Ctl moves to L''.

– If $c_i > 0$ then we have the line $l - NZ_1 - \ldots - NZ_{c_i} - Z_{c_{i+1}} - \ldots Z_{x+1}$, and Ctl broadcast a $!dec_i$ message which will be propagated until it reaches $Z_{c_{i+1}}$ which will respond with $!zero$, then the NZ_{c_i} node will go to Z_{c_i} and will propagate the acknowledgement back, eventually reaching Ctl which receives the $!ackdec_i$ message and go to L'. This also shows why we need $x + 1$ counting nodes to simulate a run in the Minsky machine.

The Ctl node can receive 2 messages back from the exchange, that is either a $!zero_i$ or a $!ackdec_i$. For the $!ackdeck_i$ it will go to L' and for the $!zero_i$ case the process will move to L'' The resulting line will be $l - NZ_1 - \ldots - NZ_{c_{i-1}} - Z_{c_i} - \ldots Z_{x+1}$, clearly we simulated the decrement, having the non-zero counting node at $NZ_{c_{i-1}}$ and the Ctl node at either L' or L'' which concludes this direction.

(2) $CSReach(\Rightarrow_r) \Rightarrow Halt(M)$*:* For this direction we show that if the clique-and-cloud topology can reach control state q then the Minsky machine halts. We again assume the initial line $Z_{2,n+1} - Z_{2,n} - \ldots - Z_{2,1} - L_0 - Z_{1,1} - \ldots - Z_{1,n'} - Z_{1,n'+1}$ and the maximum value x of the counter c_i is at most n and n' respectively. Now we show if $c \Rightarrow_r^* c'$ we have the corresponding instruction $L; L'$.

Increment Instruction: Depending on the counter of the value and amount of processes denoted by $\#_c$ we have 2 cases:

– If $c_i = 0$ and $\#_c(L_0) = 1$ then we know we must have the line $L_0 - Z_1 - \ldots - Z_{x+1}$. If Ctl broadcast an $!inc_i$ message, the first counter node Z_0 will respond with an acknowledgement $!ackinc_i$ which will make Ctl move to instruction L', which clearly shows the sequence of transitions corresponds to an increment instruction in the Minsky machine.
– If $c_i > 0$ and $\#_c(L) = 1$ then we know at least one node is in state NZ and when Ctl broadcasts an $!inc_i$ message, the NZ nodes will propagate the message until we reach the first node in state Z. This node will respond by moving to NZ and propagate $!ackinc_i$ back through the network until it reaches Ctl, which will move to the next instruction L'. Again this sequence of transitions corresponds to an increment instruction in the Minsky machine.

Decrement instruction: For this case, we make the same assumptions as the latter case.

– If $c_i = 0$ and $\#_G(L_0) = 1$ then we know the line is $L_0 - Z_0 - \ldots - Z_{x+1}$ and Ctl will broadcast a $!dec_i$ message to the node Z_0 which will respond with a $!zero_i$ moving Ctl to the instruction L''. Clearly this sequence of transitions corresponds to the decrement instruction where the decrement fails.
– If $c_i > 0$ and $\#_G(L) = 1$, now we know we must have a line with at least one node in state NZ, with the line $L - NZ_1 - \ldots - NZ_{c_i} - Z_{c_{i+1}} \ldots - Z_{x+1}$. The decrement is propagated until it reaches $Z_{c_{i+1}}$ where a $!zero_i$ is sent back to NZ_{c_i} which will go to Z_{c_i} and broadcast $ackdec_i$ which will be propagated

back to Ctl. Once the $ackdec_i$ reaches Ctl it will go to the instruction l' which clearly shows the sequence of transitions corresponds to a decrement instruction succeeding.

Following this reasoning, let L_F be the final instruction of the Minsky machine and let q correspond to the encoding of L_F. Clearly if we follow the simulation, the Ctl node will reach state q if the Minsky machine halts, which concludes the proof

B Proof of Section 5

B.1 Proof of Lemma 1

Proof. "⇒" Assume that there exists $(G, f) \in [\![\mathcal{G}]\!]$ and $c \in \mathcal{C}_G$, such that $c^G_{in} \Rightarrow^* c$ and $\#_c(q) > 0$. Since $\#_c(q) > 0$, it must be the case that $c(u) = q$ for some u and we have $f(u) \in \mathcal{V}$. Then, $q \in \gamma^{\Rightarrow}_{reach}(f(u))$, by Definition 5. "⇐" Assume that there exists a vertex $v \in \mathcal{V}$ such that $q \in \gamma^{\Rightarrow}_{reach}(v)$. Then, by Definition 5, there must exist a vertex $u \in f^{-1}(v)$ such that $c(u) = q$ and $c^G_{in} \Rightarrow^* c$. We obtain $\#_c(q) > 0$, thus concluding the proof. □

B.2 Proof of Lemma 2

Proof. Let $G_1 = (V_1, E_1, lab_1)$ where $(G_1, f_1) \in [\![\mathcal{G}]\!]$ and $G_2 = (V_2, E_2, lab_2)$ with $(G_2, f_2) \in [\![\mathcal{G}]\!]$. By construction of $lab : V \mapsto \mathcal{P}$ we have $lab(v) = lab_1(v)$ for all $v \in V_1$ and $lab(v) = lab_2(v)$ for all $v \in V_2$, then we obtain that $lab(v) = Lab(f(v))$ for all $v \in V_1 \uplus V_2$ by definition of $[\![\mathcal{G}]\!]$. By construction of E we have all edges of E_1 and E_2 along with the edges that form all the expansion pairs $\{(v, v') \in V_1 \times V_2 \mid f_1(v) = f_2(v') \text{ and } f_1(v) \in \mathcal{V}_\Delta\} \cup \{(v, v') \in V_2 \times V_1 \mid f_2(v) = f_1(v') \text{ and } f_2(v) \in \mathcal{V}_\Delta\}$ verifying the conditions $(u, v) \in E$ iff $(f(u), f(v)) \in \mathcal{E}$ or $(f(u) = f(v)$ and $f(u) \in \mathcal{V}_\Delta)$ from the definition of $[\![\mathcal{G}]\!]$. Now we have that both conditions for $G_1 \uplus G_2$ hold, and we can conclude $(G_1 \uplus G_2, f) \in [\![\mathcal{G}]\!]$. □

B.3 Proof of Lemma 3

Proof. Reliable communication ($\Rightarrow_r$): "⇒" Let $G = (V, E, lab) \in [\![G]\!]$ be a topology, $c \in \mathcal{C}_G$ be a configuration and $v \in V$ be a vertex, such that $c \Rightarrow_r c'$ and $c(v) \neq c'(v)$, for some configuration $c' \in \mathcal{C}_G$. By definition of $\Rightarrow_r$ we have two possibilities, either:

- v emits a message $m \in \Sigma$, i.e., $c(v) \xrightarrow{!m}_\delta c'(v)$: this case corresponds to condition (1) of Definition 7, or
- there exists a vertex $u \in V$ where $(u, v) \in E$ and state $q \in Q$ where $c(u) \xrightarrow{!m}_\delta q$ and $c(v) \xrightarrow{?m}_\delta c'(v)$: this case corresponds to condition (2) of Definition 7.

"$\Leftarrow$" Let $G = (V, E, lab) \in [\![G]\!]$ and assume $c \in \mathcal{C}_G$, $v \in V$ and $m \in \Sigma$ verify either condition (1) or (2) from Definition 7. If (1) is satisfied, then we know by definition of $\Rightarrow_r$ that there exists a $c' \in \mathcal{C}_G$ such that $c \Rightarrow_r c'$ and $c(v) \neq c'(v)$ with $c(v) \xrightarrow{!m}_\delta c'(v)$. Else, if condition (2) is satisfied, we know that there exists a configuration $c' \in \mathcal{C}_G$ such that $c \Rightarrow_r c'$ with $c(v) \xrightarrow{?m}_\delta c'(v)$ and $c(v) \neq c'(v)$ and there exists a vertex $u \in V$ such that $c(u) \xrightarrow{!m}_\delta q$ and $q \in Q$.

Unreliable communication ($\Rightarrow_u$): The proof for this case is the same as in the $\Rightarrow_r$ case. □

B.4 Proof of Lemma 5

Proof. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, Lab, \mathcal{V}_\Delta, \mathcal{V}_{\cdot,\cdot})$ be a clique-and-cloud topology and ν be a variable store for $\mathcal{G}$. We first assume that $\mathcal{G} \models_\nu \varphi_{ind}(X_1, \ldots, X_n)$ and we will show that γ_ν satisfy the four statements of the inductive reachability property:

1. Let $v \in \mathcal{V}$ and assume $Lab(v) = P_j$ and $q_{in,j} = q_i$. We need to prove that $q_i \in \gamma_\nu(v)$. By hypothesis, we have $\mathcal{G} \models_{\nu[x \mapsto v]} (\bigvee_{s=1}^{4} \varphi_{s,i}(x, X_1, \ldots, X_n)) \implies X_i(x)$. But since $Lab(v) = P_j$, we deduce as well that $\mathcal{G} \models_{\nu[x \mapsto v]} \varphi_{1,i}(x, X_1, \ldots, X_n))$. As a consequence, we have $\mathcal{G} \models_{\nu[x \mapsto v]} X_i(x)$ and hence $v \in \nu(X_i)$. By definition of γ_ν, we obtain $q_{in,j} = q_i \in \gamma_\nu(v)$.
2. Let $v \in \mathcal{V}$ be a vertex such that $q_j \in \gamma_\nu(v)$ and $q_j \xrightarrow{!m}_\delta q_i$. We need to prove that $q_i \in \gamma_\nu(v)$. By hypothesis, we have $\mathcal{G} \models_{\nu[x \mapsto v]} (\bigvee_{s=1}^{4} \varphi_{s,i}(x, X_1, \ldots, X_n)) \implies X_i(x)$. Thanks to the previous assumptions and the definition of γ_ν, we have that $v \in \nu(X_j)$, and as matter of fact $\mathcal{G} \models_{\nu[x \mapsto v]} X_j(x)$. This allows to deduce that $\mathcal{G} \models_{\nu[x \mapsto v]} \varphi_{2,i}(x, X_1, \ldots, X_n)$. Similarly to the previous point, we can then conclude that $q_i \in \gamma_\nu(v)$.
3. Let $v \in \mathcal{V}_\Delta$ such that there exist $q_k, q_{k'} \in \gamma_\nu(v)$ and $q_k \xrightarrow{!m}_\delta q$ and $q_{k'} \xrightarrow{?m}_\delta q_i$. We need to prove that $q_i \in \gamma_\nu(v)$. By hypothesis, we have $\mathcal{G} \models_{\nu[x \mapsto v]} (\bigvee_{s=1}^{4} \varphi_{s,i}(x, X_1, \ldots, X_n)) \implies X_i(x)$. Thanks to the previous assumptions and the definition of γ_ν, we have that $v \in \nu(X_k)$ and $v \in \nu(X_{k'})$ and we recall that $v \in \mathcal{V}_\Delta$, as matter of fact $\mathcal{G} \models_{\nu[x \mapsto v]} X_k(x) \wedge X_{k'}(x) \wedge \Delta(x)$. This allows to deduce that $\mathcal{G} \models_{\nu[x \mapsto v]} \varphi_{3,i}(x, X_1, \ldots, X_n))$. As previously, we can then conclude that $q_i \in \gamma_\nu(v)$.
4. Let $v, v' \in \mathcal{V}$ such that $(v, v') \in \mathcal{E}$ and such that there exist $q_k \in \gamma_\nu(v)$ and $q_{k'} \in \gamma_\nu(v')$ and $q_k \xrightarrow{!m}_\delta q$ and $q_{k'} \xrightarrow{?m}_\delta q_i$. We need to prove that $q_i \in \gamma_\nu(v')$. By hypothesis, we have $\mathcal{G} \models_{\nu[x \mapsto v']} (\bigvee_{s=1}^{4} \varphi_{s,i}(x, X_1, \ldots, X_n)) \implies X_i(x)$. Thanks to the previous assumptions and the definition of γ_ν, we have that $v \in \nu(X_k)$ and $v' \in \nu(X_{k'})$ and we recall that $(v, v') \in \mathcal{E}$, as matter of fact $\mathcal{G} \models_{\nu[x \mapsto v', x' \mapsto v]} X_k(x') \wedge X_{k'}(x) \wedge E(x', x)$. This allows to deduce that $\mathcal{G} \models_{\nu[x \mapsto v]} \varphi_{4,i}(x, X_1, \ldots, X_n))$. As previously, we can then conclude that $q_i \in \gamma_\nu(v')$.

Consequently, γ_ν respects the inductive reachability property.

We now suppose that γ_ν satisfies the inductive reachability property, and we will show that $\mathcal{G} \models_\nu \varphi_{ind}(X_1, \ldots, X_n)$. Let $v \in \mathcal{V}$. We show for all $i \in [1, n]$ that $\mathcal{G} \models_{\nu[x \mapsto v]} \bigwedge_{s=1}^{4} (\varphi_{s,i}(x, X_1, \ldots, X_n) \implies X_i(x))$ (which is equivalent to $\mathcal{G} \models_{\nu[x \mapsto v]} (\bigvee_{s=1}^{4} \varphi_{s,i}(x, X_1, \ldots, X_n)) \implies X_i(x)$). We deal with its conjunct in a separate matter and take $i \in [1, n]$.

1. We show that $\mathcal{G} \models_{\nu[x \mapsto v]} (\varphi_{1,i}(x, X_1, \ldots, X_n) \implies X_i(x))$. For this matter, we suppose that $\mathcal{G} \models_{\nu[x \mapsto v]} (\varphi_{1,i}(x, X_1, \ldots, X_n)$, hence we have $Lab(v) = P_j$ with $q_i = q_{in,j}$. Since γ_ν respects the inductive reachability property, we know that $q_i \in \gamma_\nu(v)$. By definition of γ_ν, we have $v \in \nu(X_i)$, and consequently $\mathcal{G} \models_{\nu[x \mapsto v]} X_i(x)$.
2. We show that $\mathcal{G} \models_{\nu[x \mapsto v]} (\varphi_{2,i}(x, X_1, \ldots, X_n) \implies X_i(x))$. For this matter, we suppose that $\mathcal{G} \models_{\nu[x \mapsto v]} (\varphi_{2,i}(x, X_1, \ldots, X_n)$. Then we know that there is some transition $q_j \xrightarrow{!m}_\delta q_i$ such that $\mathcal{G} \models_{\nu[x \mapsto v]} X_j(x)$. This allows us to deduce that $q_j \in \gamma_\nu(v)$. Since γ_ν respects the inductive reachability property, we know that $q_i \in \gamma_\nu(v)$. As before, this allows to conclude that $\mathcal{G} \models_{\nu[x \mapsto v]} X_i(x)$.
3. We show that $\mathcal{G} \models_{\nu[x \mapsto v]} (\varphi_{3,i}(x, X_1, \ldots, X_n) \implies X_i(x))$. For this matter, we suppose that $\mathcal{G} \models_{\nu[x \mapsto v]} (\varphi_{3,i}(x, X_1, \ldots, X_n)$. Then we know there exists two transitions $q_k \xrightarrow{!m}_\delta q$ and $q_{k'} \xrightarrow{?m}_\delta q_i$ such that $\mathcal{G} \models_{\nu[x \mapsto v]} X_k(x) \wedge X_{k'}(x) \wedge \Delta(x)$. This allows us to deduce that $q_k, q_{k'} \in \gamma_\nu(v)$ and $v \in \mathcal{V}_\Delta$. Since γ_ν respects the inductive reachability property, we know that $q_i \in \gamma_\nu(v)$. As before, this allows to conclude that $\mathcal{G} \models_{\nu[x \mapsto v]} X_i(x)$.
4. We show that $\mathcal{G} \models_{\nu[x \mapsto v]} (\varphi_{4,i}(x, X_1, \ldots, X_n) \implies X_i(x))$. For this matter, we suppose that $\mathcal{G} \models_{\nu[x \mapsto v]} (\varphi_{4,i}(x, X_1, \ldots, X_n)$. Then we know there exists two transitions $q_k \xrightarrow{!m}_\delta q$ and $q_{k'} \xrightarrow{?m}_\delta q_i$ and a node $u \in \mathcal{V}$ such that $\mathcal{G} \models_{\nu[x \mapsto v, x' \mapsto u]} X_k(x') \wedge X_{k'}(x) \wedge E(x', x)$. This allows us to deduce that $q_{k'} \in \gamma_\nu(v)$ and $q_k \in \gamma_\nu(u)$ and $(u, v) \in \mathcal{E}$. Since γ_ν respects the inductive reachability property, we know that $q_i \in \gamma_\nu(v)$. As before, this allows to conclude that $\mathcal{G} \models_{\nu[x \mapsto v]} X_i(x)$.

We have hence proven that $\mathcal{G} \models_\nu \varphi_{ind}(X_1, \ldots, X_n)$. □

B.5 Proof of Theorem 3

Proof. Let $\Rightarrow$ be a network transition relation satisfying the composition property for emitters and the local successor property and Γ be a VR-graph grammar and a q be a control state (w.l.o.g. we assume $q = q_1$). We show that the answer to $\texttt{CSReach}(\Rightarrow)$ with Γ and q_1 as input is positive iff $\mathcal{L}(\Gamma) \cap [\![\varphi]\!] \neq \emptyset$. This will complete the proof as this latter property is decidable thanks to Theorem 2.

First assume that there exist $\mathcal{G} \in L(\Gamma)$ (with $\mathcal{G} = (\mathcal{V}, \mathcal{E}, Lab, \mathcal{V}_\Delta, \mathcal{V}_\_)$) and $G \in [\![\mathcal{G}]\!]$ and $c \in \mathcal{C}_G$ such that: $\#_c(q_1) > 0$ and $c_{in}^G \Rightarrow^* c$. Thanks to Lemma 1, there exists $v \in \mathcal{V}$ such that $q_1 \in \gamma^{\Rightarrow}_{reach}(v)$. Take the variable store ν such that $\gamma_\nu = \gamma^{\Rightarrow}_{reach}$. Using Lemma 6, we have $\mathcal{G} \models_\nu \Phi_{reach}(X_1, \ldots, X_n)$. Since $q_1 \in \gamma^{\Rightarrow}_{reach}(v)$, we deduce by definition of ν that $\mathcal{G} \models_{\nu[x \mapsto v]} \Phi_{reach}(X_1, \ldots, X_n) \wedge X_1(x)$. Consequently, we have $\mathcal{G} \models \exists x. \exists X_1. \ldots. \exists X_n \Phi_{reach}(X_1, \ldots, X_n) \wedge X_1(x)$.

Now assume that $\mathcal{G} \in L(\Gamma)$ (with $\mathcal{G} = (\mathcal{V}, \mathcal{E}, Lab, \mathcal{V}_\Delta, \mathcal{V}_{\dot{:}})$) such that $\mathcal{G} \models \exists x.\exists X_1 \ldots . \exists X_n \Phi_{reach}(X_1, \ldots, X_n) \wedge X_1(x)$. We will show that this implies that there exists $v \in \mathcal{V}$ such that $q_1 \in \gamma^{\Rightarrow}_{reach}(v)$ and Lemma 1 will allow us to conclude. Take the store variable ν such that $\mathcal{G} \models_\nu \Phi_{reach}(X_1, \ldots, X_n) \wedge X_1(x)$. Using Lemma 6, we know that $\gamma_\nu = \gamma^{\Rightarrow}_{reach}$. By definition of γ_ν since we have $\nu(x) \in \nu(X_1)$ we deduce that $q_1 \in \gamma_\nu(\nu(x))$, hence $q_1 \in \gamma^{\Rightarrow}_{reach}(\nu(x))$. $\square$