Projet PACS

Deliverable 1 : Report on Advances on Discrete Parameter Synthesis

1 Summary

Various research works have been achieved in the context of the Task 1 of ANR Project PACS whose main focus was to develop verification and synthesis techniques for models defined with parameters. In these different works, the considered parameters are taking their value in some discrete (but unbounded) state space. The main ideas of these works are provided in this Section and the corresponding articles can be found in Appendix of this document

1.1 Studying networks where the number of participants is a parameter

In order to analyze networks protocols which are designed to run over an unbounded number of participants, it is natural to consider models where a parametric number of entities have a similar behavior. Many works over such models have been done in this task and the variations between them comes from the studied family of networks and as well from the different nature of the problems.

Three of these works propose methods for a model where the communication between the entities of the network is done by broadcasting of messages and they assume as well that the network comes with a communication topology (basically a graph). In some cases, this graph can be assumed to be static, i.e. the communication topology does not change during an execution of the protocol. In other cases, the configuration topology is subjected to reconfiguration, in order to simulate both the loss of the messages and the possible mobility of the entities.

For what concerns, the last work, it studies simpler networks, where processes communicate through a shared register but the technical breakthrough comes from the fact that it provides some results which take into account a randomized scheduler in the network whose role is to indicate at each instant the process to be executed.

In [DST16], the authors study parameterized verification problems for networks of interacting register automata. The network is represented through a graph, and processes may exchange broadcast messages containing data with their neighbours. Upon reception a process can either ignore a sent value, test for equality with a value stored in a register, or simply store the value in a register. They consider safety properties expressed in terms of reachability, from arbitrarily large initial configurations, of a configuration exposing some given control states and patterns. They

investigate, in this context, the impact on decidability and complexity of the number of local registers, the number of values carried by a single message, and dynamic reconfigurations of the underlying network.

In [ADR+16], the authors study decidability and undecidability results for parameterized verification of a formal model of timed Ad Hoc network protocols. The communication topology is defined by an undirected graph and the behaviour of each node is defined by a timed automaton communicating with its neighbours via broadcast messages. They consider parameterized verification problems formulated in terms of reachability. In particular they are interested in searching for an initial configuration from which an individual node can reach an error state. They study the problem for dense and discrete time and compare the results with those obtained for (fully connected) networks of timed automata.

In [BFS15] the authors study the problems of reaching a specific control state, or converging to a set of target states, in networks with a parameterized number of identical processes communicating via broadcast. To reflect the distributed aspect of such networks, they restrict their attention to executions in which all the processes must follow the same *local strategy* that, given their past performed actions and received messages, provides the next action to be performed. They show that the reachability and target problems under such local strategies are NP-complete, assuming that the set of receivers is chosen non-deterministically at each step. On the other hand, these problems become undecidable when the communication topology is a clique. However, decidability can be regained for reachability under the additional assumption that all processes are bound to receive the broadcast messages.

In [BMR+16], the authors study the almost-sure reachability problem in a distributed system obtained as the asynchronous composition of N copies of the same automaton (that can communicate via a shared register with finite domain. The automaton has two types of transitions : write-transitions update the value of the register, while read-transitions move to a new state depending on the content of the register. Non-determinism is resolved by a stochastic scheduler. Given a protocol, they focus on almost-sure reachability of a target state by one of the processes. The answer to this problem naturally depends on the number N of processes. However, they prove that our setting has a cut-off property : the answer to the almost-sure reachability problem is constant when N is large enough ; we then develop an EXPSPACE algorithm deciding whether this constant answer is positive or negative.

1.2 Introducing parameters in the transition relation of Petri nets

In [DJLR15], the authors have studied how discrete parameters can be introduced in the transition relation of Petri net. In a system modelled as a Petri net, the number of identical processes involved, the number of identical processes required for some task, or the number of identical processes spawned by some task, can all be modeled using the marking of the net and the input or output weights of the arcs. Besides, most safety properties can be modeled by the coverability property of Petri nets : is it possible to put at least this many tokens in some given places. They have therefore studied the parameterization of markings and weights in Petri nets and, particularly, the

parametric decision problems related to coverability : does there exist a parameter valuation such that some marking in the instantiated net is coverable ? and is some marking coverable for all parameter valuations ? They prove that the problem is undecidable in general, but provide natural syntactical subclasses for which it is decidable.

1.3 Extending regular model-checking to more expressive models

Concerning regular model-checking the authors have considered in [DH16] an extension of the classical setting using words over a finite alphabet and their transformation to data words. Data words consist of letters which are pairs of an element from a finite domain and a data value from an infinite domain. They study a class of transformations of finite data words which generalizes the well-known class of regular finite string transformations described by MSO-definable transductions of finite strings. These transformations map input words to output words whereas our transformations handle data words where each position has a letter from a finite alphabet and a data value. Each data value appearing in the output has as origin a data value in the input. As is the case for regular transformations they show that our class of transformations has equivalent characterizations in terms of novel deterministic two-way and streaming string transducers.

Références

[ADR+16] Parosh Aziz Abdulla, Giorgio Delzanno, Othmane Rezine, Arnaud Sangnier, and Riccardo Traverso. Parameterized verification of time-sensitive models of ad hoc network protocols. *Theoretical Computer Science*, 612 :1–22, 2016. 2

[BFS15] Nathalie Bertrand, Paulin Fournier, and Arnaud Sangnier. Distributed local strategies in broadcast networks. In *26th International Conference on Concurrency Theory, CONCUR 2015*, volume 42 of *LIPIcs*, pages 44–57. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. 2

[BMR+16] Patricia Bouyer, Nicolas Markey, Mickael Randour, Arnaud Sangnier, and Daniel Stan. Reachability in networks of register protocols under stochastic schedulers. In *Proceedings of the 43rd International Colloquium on Automata, Languages and Programming - Part II (ICALP'16 (2))*, volume 55 of *LIPIcs*, pages 106 :1–106 :14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. 2

[DH16] Antoine Durand-Gasselin and Peter Habermehl. Regular transformations of data words through origin information. In *FOSSACS 2016*, volume 9634 of *Lecture Notes in Computer Science*, pages 285–300. Springer, 2016. 3

[DJLR15] Nicolas David, Claude Jard, Didier Lime, and Olivier H. Roux. Discrete parameters in petri nets. In *PETRI NETS'15*, volume 9115 of *Lecture Notes in Computer Science*, pages 137–156. Springer, 2015. 2

[DST16] Giorgio Delzanno, Arnaud Sangnier, and Riccardo Traverso. Adding data registers to parameterized networks with broadcast. volume 143(3-4), pages 287–316, 2016. 1

Parameterized Verification of Time-sensitive Models of Ad Hoc Network Protocols[1]

Parosh Aziz Abdulla[a], Giorgio Delzanno[b], Othmane Rezine[a], Arnaud Sangnier[c], Riccardo Traverso[b]

[a] *Uppsala University, Sweden*
[b] *University of Genova, Italy*
[c] *LIAFA, Univ Paris Diderot, CNRS, France*

Abstract

We study decidability and undecidability results for parameterized verification of a formal model of timed Ad Hoc network protocols. The communication topology is defined by an undirected graph and the behaviour of each node is defined by a timed automaton communicating with its neighbours via broadcast messages. We consider parameterized verification problems formulated in terms of reachability. In particular we are interested in searching for an initial configuration from which an individual node can reach an error state. We study the problem for dense and discrete time and compare the results with those obtained for (fully connected) networks of timed automata.
Keywords Parameterized Verification, Timed Automata, Ad Hoc Networks, Graphs, Decidability, Well Structured Transition Systems

1. Introduction

In recent years there has been an increasing interest in automated verification methods for ad hoc networks, see e.g. [18, 24, 23, 11, 12]. Ad Hoc Networks (AHN) consist of wireless hosts that, in absence of a fixed infrastructure, communicate sending broadcast messages. In this context, protocols are supposed to work independently from a specific configuration of the network. Indeed, discovery protocols are often applied in order to identify the vicinity of a given node. In the AHN model proposed in [11] undirected graphs are used to represent a network in which each node executes an instance of a fixed (untimed) interaction protocol based on broadcast communication. Since individual nodes are not aware of the network topology, in the ad hoc setting it is natural to consider verification problems that are parametric in the size and shape of the initial configuration as in [11].

[1] This work is partially supported by the ANR national research program ANR-14-CE28-0002 PACS.

In this paper we introduce a new model of distributed systems obtained by enriching the AHN model of [11] with time-sensitive specification of individual nodes. In the resulting model, called Timed Ad Hoc Networks (TAHN), the connection topology is still modelled as a graph in which nodes communicate via broadcast messages but the behaviour of a node is now defined as a timed automaton. More in detail, each node has a finite set of clocks which all advance at the same rate and transitions describing the behaviour of the nodes are guarded by conditions on clocks and have also the ability to reset clocks.

Following [11, 12], we study the decidability status of the parameterized reachability problem taking as parameters the initial configuration of a TAHN, i.e., we aim at checking the existence of an initial configuration that can evolve using continuous and discrete steps into a configuration exposing a given local state (usually representing an error). Our model presents similarities with Timed Networks introduced in [2]. A major difference between TAHN and Timed Networks lies in the fact that in the latter model the connection topology is always a fully-connected graph, i.e., broadcast communication is not selective since a message sent by a node always reaches all other nodes. For Timed Networks, it is known that reachability of a configuration containing a given control location is undecidable in the case of two clocks per node, and decidable in the case of one clock per node.

When constraining communication via a complex connection graph, the decidability frontier becomes much more complex. More specifically, our technical results are as follows:

- For nodes equipped with a single clock, parameterized reachability becomes undecidable in a very simple class of graphs in which nodes are connected so as to form stars with diameter five.

- The undecidability result still holds in the more general class of bounded path graphs, i.e., graphs in which the length of maximal simple paths is bounded by a constant. In our proof we consider a bound $N \geq 5$ on the length of simple paths. Since nodes have no information about the shape of the network topology, the undecidability proof is not a direct consequence of the result for stars. Indeed the undecidability construction requires a preliminary step aimed at discovering a two-star topology in a graph of arbitrary shape but simple paths of at most five nodes.

- The problem turns out to be undecidable in the class of cliques of arbitrary order (that contains graphs with arbitrarily long paths) in which each timed automaton has at least two clocks.

- Decidability holds for special topologies like stars with diameter three and cliques of arbitrary order assuming that the process running in each node is equipped with a single clock (as in Timed Networks).

- Finally when considering discrete time, e.g. to model time-stamps, instead of continuous time, we show that the local state reachability problem becomes decidable for processes with any number of clocks in the class of

graphs with bounded path. The same result holds for cliques of arbitrary order.

2. Preliminaries

Let $\mathbb{N}$ be the set of natural numbers and $\mathbb{R}^{\geq 0}$ the set of non-negative real numbers. For sets A and B, we use $f : A \mapsto B$ to denote that f is a total function that maps A to B. For $a \in A$ and $b \in B$, we write $f\,[a \hookleftarrow b]$ to denote the function f' defined as follows: $f'(a) = b$ and $f'(a') = f(a')$ for all $a' \neq a$. We denote by $[A \mapsto B]$ the set of all total functions from A to B.

We now recall the notion of well-quasi-ordering (which we abbreviate as wqo). A quasi-order $(A, \preceq)$ is a wqo if for every infinite sequence of elements $a_1, a_2, \ldots$ in A, there exist two indices $i < j$ such that $a_i \preceq a_j$. Given a set A with an ordering $\preceq$ and a subset $B \subseteq A$, the set B is said to be *upward closed* in A if $a_1 \in B$, $a_2 \in A$ and $a_1 \preceq a_2$ implies $a_2 \in B$. Given a set $B \subseteq A$, we define the upward closure $\uparrow B$ to be the set $\{a \in A \mid \exists a' \in B \text{ such that } a' \preceq a\}$. For a quasi-order $(A, \preceq)$, an element a is minimal for $B \subseteq A$ if for all $b \in B$, $b \preceq a$ implies $a \preceq b$. If $(A, \preceq)$ is a wqo and if B is upward closed in A, then the set of minimal elements of B is finite. If $\{b_1, \ldots, b_k\}$ is the set of minimal elements of B, then $\uparrow\{b_1, \ldots, b_k\} = B$; hence B can be represented finitely.

3. Timed Ad Hoc Networks

3.1. Syntax

A Timed Ad Hoc Network (TAHN) consists of a graph where the nodes represent processes that run a common predefined protocol defined by a communicating timed automaton. The values of the clocks manipulated by the automaton inside each process are incremented all at the same rate. In addition, processes may perform discrete transitions which are either local transitions or communication events. When firing a *local* transition, a single process changes its local state without interacting with the other processes. For what concerns communication, it is performed by means of *selective broadcast*, a process sends a broadcast message which can be received only by its neighbours in the network. We choose to represent the communication relation as a graph. Finally, transitions are guarded by conditions on values of clocks and may also reset clocks.

We now provide the formal definition of the model. We assume that each process operates on a set of clocks X. A *guard* is a boolean combination of predicates of the form $k \lhd x$ for $k \in \mathbb{N}$, $\lhd \in \{=, <, \leq, >, \geq\}$, and $x \in X$. We denote by $\mathcal{G}(X)$ the set of guards over X. A reset R is a subset of X. The guards will be used to impose conditions on the clocks of processes that participate in transitions and the resets to identify the clocks that will be reset during the transition. A *clock valuation* is a mapping $F : X \mapsto \mathbb{R}^{\geq 0}$. For a guard g and a clock valuation F, we write $F \models g$ to indicate that the formula obtained by replacing in the guard g each clock x by $F(x)$ is valid. For a clock valuation F

and a subset of clocks $Y \subseteq X$, we denote by $F[Y]$ the clock valuation such that $F[Y](x) = 0$ for all $x \in Y$ and $F[Y](x) = F(x)$ for all $x \in X \setminus Y$.

For a finite alphabet Σ of messages, we define the set of events associated to this alphabet as follows: $\mathcal{M}(\Sigma) = \{\tau\} \cup \{!!a, ??a \mid a \in \Sigma\}$. These events correspond to the following ideas:

(i) τ is used for a local move ;

(ii) $!!a$ represents the broadcast of the message a;

(iii) $??a$ denotes the reception of the message a (that has been broadcasted by another process).

We now give the definition of a protocol which will be executed by the nodes in the network.

Definition 1. *A protocol P is a tuple $\left(Q, X, \Sigma, \mathcal{R}, q^{init}\right)$ such that Q is a finite set of states, X is a finite set of clocks, Σ is a finite message alphabet, $\mathcal{R} \subseteq Q \times \mathcal{G}(X) \times \mathcal{M}(\Sigma) \times 2^X \times Q$ is a finite set of rules labelled with a guard, a message and a reset, and $q^{init} \in Q$ is an initial state.*

Intuitively P defines the protocol that is run by each of the nodes (or entities) present in the network, where Q is the set of *local states* of each node, while $\mathcal{R}$ is a set of *rules* describing the behaviour of each node. We will use the notation $\left(q, g \xrightarrow{e} R, q'\right)$ to represent the rule (q, g, e, R, q'). For a protocol $P = \left(Q, X, \Sigma, \mathcal{R}, q^{init}\right)$, we denote by $nbclocks(P)$ the size of X, i.e., the number of clocks it uses.

A TAHN $\mathcal{T}$ is then simply a pair (G, P) where:

- $G = (V, E)$ is a *connectivity graph* composed of a finite set of nodes V and a set of undirected edges without self-loops, i.e., $E \subseteq V \times V \setminus \{(v, v) \mid v \in V\}$ s.t. E is symmetric;
- P is the protocol which will be executed by the node present in the nodes of the graph.

Intuitively, the graph G characterizes potential process interactions in the network $\mathcal{T}$; the set V represents the nodes and E defines the connectivity relation between the nodes of the network. The nodes belonging to an edge are called the *endpoints* of the edge. For an edge $(u, v) \in E$, we often use the notation $u \sim v$ and say that the vertices u and v are adjacent to each other.

3.2. Operational Semantics

We now define the operational semantics of TAHN by means of a timed transition system. Let $\mathcal{T} = (G, P)$ be a TAHN with $G = (V, E)$ and $P = \left(Q, X, \Sigma, \mathcal{R}, q^{init}\right)$. A configuration γ of $\mathcal{T}$ is a pair $(\mathcal{Q}, \mathcal{X})$ where:

- $\mathcal{Q} : V \mapsto Q$ is a function that maps each node of the graph with a state of the protocol;

- $\mathcal{X} : V \mapsto [X \mapsto \mathbb{R}^{\geq 0}]$ is a function that assigns to each node a clock valuation.

An important point is that, in a configuration, each node of the graph has its own set of clocks. We denote by $\mathcal{C}_\mathcal{T}$ the set of configurations. The *initial configuration* of $\mathcal{T}$ is the configuration $\left(\mathcal{Q}^{init}, \mathcal{X}^{init}\right)$ with $\mathcal{Q}^{init}(v) = q^{init}$ and $\mathcal{X}^{init}(v)(x) = 0$ for all $v \in V$ and $x \in X$. In other words, in an initial configuration all the nodes are in the initial local state and all their associated clocks have value 0.

We now introduce a notation to characterize the nodes in a configuration that are able to receive a message a. Given a configuration $\gamma = (\mathcal{Q}, \mathcal{X})$ of the TAHN $\mathcal{T} = (G, P)$ (with $G = (V, E)$) and given a message $a \in \Sigma$, let $En_\mathcal{T}(\gamma, a)$ be the following set of nodes able to receive a in γ from the connectivity graph G:

$$En_\mathcal{T}(\gamma, a) = \{v \in V \mid \exists \left(q, g \xrightarrow{??a} R, q'\right) \in \mathcal{R} \text{ s.t. } \mathcal{Q}(v) = q \text{ and } \mathcal{X}(v) \models g\}$$

In the rest of the paper we will use $En(\gamma, a)$ when $\mathcal{T}$ is clear from the context.

The semantics associated to a TAHN $\mathcal{T}$ is then defined by the timed transition system $(\mathcal{C}_\mathcal{T}, \Longrightarrow_\mathcal{T})$, where the transition relation $\Longrightarrow_\mathcal{T} \subseteq \mathcal{C}_\mathcal{T} \times \mathcal{C}_\mathcal{T}$ corresponds to the union of a *discrete* transition relation $\Longrightarrow_{\mathcal{T},d}$, representing transitions induced by the rules of $\mathcal{T}$ and a *timed* transition relation $\Longrightarrow_{\mathcal{T},t}$ which characterizes the elapse of time.

The discrete transition relation $\Longrightarrow_{\mathcal{T},d} \subseteq \mathcal{C}_\mathcal{T} \times \mathcal{C}_\mathcal{T}$ is such that given two configurations $\gamma = (\mathcal{Q}, \mathcal{X})$ and $\gamma' = (\mathcal{Q}', \mathcal{X}')$, we have $\gamma \Longrightarrow_{\mathcal{T},d} \gamma'$ if and only if one of the following conditions is satisfied:

Local: There exists a rule $\left(q, g \xrightarrow{\tau} R, q'\right)$ and a vertex $v \in V$ such that $\mathcal{Q}(v) = q$, $\mathcal{X}(v) \models g$, $\mathcal{Q}' = \mathcal{Q}[v \hookleftarrow q']$, and $\mathcal{X}' = \mathcal{X}[v \hookleftarrow \mathcal{X}(v)[R]]$, and, for each $w \in V \setminus \{v\}$, we have $\mathcal{Q}'(w) = \mathcal{Q}(w)$, $\mathcal{X}'(w) = \mathcal{X}(w)$.

Broadcast: There exists a rule $\left(q, g \xrightarrow{!!a} R, q'\right)$ and a vertex $v \in V$ such that $\mathcal{Q}(v) = q$, $\mathcal{X}(v) \models g$, $\mathcal{Q}'(v) = q'$ and $\mathcal{X}'(v) = \mathcal{X}(v)[R]$, and, for each $w \in V \setminus \{v\}$, we have:

- either $w \sim v$ and $w \in En(\gamma, a)$ and there exists a rule of the form $\left(q_1, g_1 \xrightarrow{??a} R_1, q_1'\right)$ such that $\mathcal{Q}(w) = q_1$, $\mathcal{X}(w) \models g_1$, $\mathcal{Q}'(w) = q_1'$, and $\mathcal{X}'(w) = \mathcal{X}(w)[R_1]$.
- or $(w \not\sim v$ or $w \notin En(\gamma, a))$, $\mathcal{Q}'(w) = \mathcal{Q}(w)$, and $\mathcal{X}'(w) = \mathcal{X}(w)$.

The timed transition relation $\Longrightarrow_{\mathcal{T},t} \subseteq \mathcal{C}_\mathcal{T} \times \mathcal{C}_\mathcal{T}$ is such that given two configurations $\gamma = (\mathcal{Q}, \mathcal{X})$ and $\gamma' = (\mathcal{Q}', \mathcal{X}')$, we have $\gamma \Longrightarrow_{\mathcal{T},t} \gamma'$ if and only if:

Time: There is a $\delta \in \mathbb{R}^{\geq 0}$ such that for all $v \in V$ and $x \in X$, $\mathcal{Q}'(v) = \mathcal{Q}(v)$ and $\mathcal{X}'(v)(x) = \mathcal{X}(v)(x) + \delta$.

As said before, $\Longrightarrow_\mathcal{T}$ is then equal to $\Longrightarrow_{\mathcal{T},d} \cup \Longrightarrow_{\mathcal{T},t}$.

3.3. Topologies

As we will see, we will often restrict the connectivity graph of TAHN to belong to a family of graphs. In this paper, we consider different families of graphs that we call *topologies*. A *topology Top* is hence a class of graphs that we use to impose structural restrictions on the communication graph of a set of configurations. In the sequel we write $G \in Top$ to indicate that the graph G belongs to a given *Top*. We now list the topologies we will take into account in this work.

- `GRAPH` is the topology consisting of all finite graphs.
- For $\ell \geq 0$, `STAR`(ℓ) is the star topology of depth ℓ. It characterizes graphs $G = (V, E)$ for which there is a partition of V of the form $\{v_0\} \cup V_1 \cup \cdots \cup V_\ell$ such that:

  (i) $v_0 \sim v_1$ for all $v_1 \in V_1$;

  (ii) for each $1 \leq i < \ell$ and $v_i \in V_i$ there is one and only $v_{i+1} \in V_{i+1}$ with $v_i \sim v_{i+1}$ and one and only one $v_{i-1} \in V_{i-1}$ with $v_i \sim v_{i-1}$;

  (iii) no other nodes are adjacent to each other.

  In other words, in a star graph of dimension ℓ, there is a central node v_0 and an arbitrary number of *rays*. A ray consists of a path $v_1, v_2, \ldots, v_\ell$ of ℓ nodes, starting from v_1 adjacent to v_0. We call v_0 the *root*, $v_1, v_2, \ldots, v_{\ell-1}$ *internal nodes*, and v_ℓ a *leaf* of G.
- For $\ell \geq 0$, `BOUNDED`(ℓ) is the bounded path topology of bound ℓ. It characterizes graphs for which the length of the maximal simple path is bounded by ℓ. Formally, if $G \in$ `BOUNDED`(ℓ) with $G = (V, E)$ then there does not exist a finite sequence of nodes $(v_i)_{1 \leq i \leq m}$ such that $m > \ell$, and, $v_i \neq v_j$ for all i, j in $\{1, \ldots, m\}$ with $i \neq j$, and, $v_i \sim v_{i+1}$ for all $i \in \{1, \ldots, m-1\}$.
- `CLIQUE` is the set of cliques which characterizes graphs $G = (V, E)$ such that $v \sim w$ for all $v, w \in V$ with $v \neq w$.

3.4. State reachability problem

We now present the verification problem we study in this work. It consists in determining for a given protocol whether there exists a connectivity graph belonging to a certain topology such that in the obtained TAHN it is possible to reach, from the initial configuration, a configuration exhibiting a specific state (for instance an error state). We insist on the fact that we do not restrict the number of nodes appearing in the considered connectivity graphs. Notice that all the classes of graphs (called topologies) introduced previously have an infinite cardinality hence an algorithm enumerating all the graphs belonging to a given topology cannot be applied to solve our reachability problem. In fact, as we shall see, the main difficulty in this problem is that the set of connectivity graphs to consider is infinite.

Let $\mathcal{T} = (G, P)$ be a TAHN with a protocol $P = \left(Q, X, \Sigma, \mathcal{R}, q^{init}\right)$ and a connectivity graph $G = (V, E)$. We say that a configuration γ_n is *reachable* in $\mathcal{T}$ if there exists a finite path, starting at the initial configuration γ_0, of the form $\gamma_0 \Longrightarrow_{\mathcal{T}} \gamma_1 \Longrightarrow_{\mathcal{T}} \cdots \Longrightarrow_{\mathcal{T}} \gamma_n$ in the associated transition system . Given a state $q \in Q$, we say that q is reachable in the TAHN $\mathcal{T}$ if there exists a reachable configuration $\gamma = (\mathcal{Q}, \mathcal{X})$ and a vertex $v \in V$ such that $\mathcal{Q}(v) = q$.

We now define the state reachability problem $\texttt{TAHN}-\texttt{Reach}\,(\mathit{Top}, K)$ parameterized by a topology *Top* and a number of clocks K as follows:

Input: A protocol P such that $\mathit{nbclocks}(P) \leq K$ and a control state q;

Output: Is there a TAHN $\mathcal{T} = (G, P)$ with $G \in \mathit{Top}$ such that q is reachable in $\mathcal{T}$?

In [11, 12], a model of Ad Hoc Networks without time has been studied; it is the same as the one we have introduced considering protocols without clocks. The authors have shown that when the connectivity graphs are unrestricted, then the state reachability problem is undecidable. However, one can regain the decidability by restricting the graphs to have bounded path (i.e., graphs in which the length of the maximal simple path is bounded). Note also that when the reachability problem is restricted to cliques, then TAHN without clocks are equivalent to Broadcast Protocols (with no rendez-vous communication) which were introduced in [17] and for which the reachability problem is proved to be decidable. A proof, in terms of Ad Hoc Networks, of this latter result can also be found in [12]. The following theorem rephrases these results in our context.

Theorem 1. *[11, 17, 12]*

1. $\texttt{TAHN}-\texttt{Reach}\,(\texttt{GRAPH}, 0)$ *is undecidable.*
2. *For all* $N \geq 1$, $\texttt{TAHN}-\texttt{Reach}\,(\texttt{BOUNDED}(N), 0)$ *is decidable.*
3. $\texttt{TAHN}-\texttt{Reach}\,(\texttt{CLIQUE}, 0)$ *is decidable*

Remark 1. *We point out the fact that for a number of clocks K and given two topologies Top and Top$'$, if Top $\subset$ Top$'$, we cannot infer directly any relation between the decidability status of* $\texttt{TAHN}-\texttt{Reach}\,(\mathit{Top}, K)$ *and* $\texttt{TAHN}-\texttt{Reach}\,(\mathit{Top}, K')$. *For instance if* $\texttt{TAHN}-\texttt{Reach}\,(\mathit{Top}, K)$ *is undecidable, then it does not imply necessarily that* $\texttt{TAHN}-\texttt{Reach}\,(\mathit{Top}, K')$ *is undecidable, it could be in fact the case that dealing with a larger class of graphs renders the problem solvable. Similarly if* $\texttt{TAHN}-\texttt{Reach}\,(\mathit{Top}, K')$ *is undecidable, we know that* $\texttt{TAHN}-\texttt{Reach}\,(\mathit{Top}, K)$ *could be decidable (see the above theorem where we have* $\texttt{CLIQUE} \subset \texttt{GRAPH}$*).*

3.5. Example

Consider the protocol P described at Figure 1 which uses a single clock per process. In this protocol, after more than one time unit, processes can broadcast m_1 or m_3. A process in initial state can then receive a message m_1, and after reception of such a message, it can broadcast a message m_2 if the delay between the reception of m_1 and the broadcast of m_2 is strictly more than one time unit.

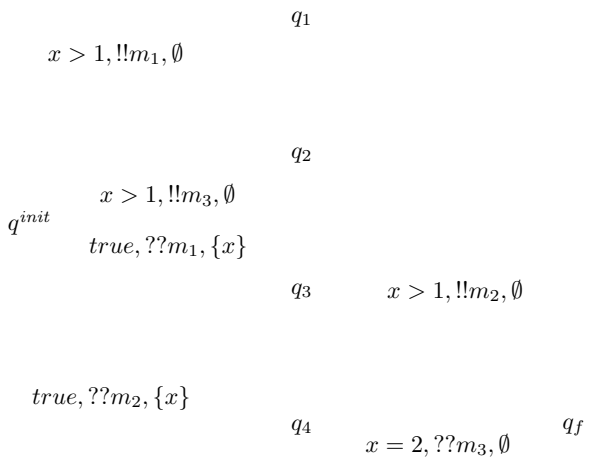


Figure 1: A protocol P

Finally, a process can reach the state q_f if it receives a message m_2 and exactly 2 times unit after, it receives a message m_3.

The Figure 2 gives two examples of connectivity graphs; the first one G_1 belongs to the topology `CLIQUE` and the second one G_2 belongs to `STAR`(2).

(a) A clique graph G_1 (b) A star graph G_2 of depth 2

Figure 2: Example of two connectivity graphs

We are interested in knowing whether q_f is reachable in (G_1, P) and (G_2, P). We first consider the TAHN (G_1, P). In this model as soon as a process broadcasts a message m_1, then all the processes in initial state have to receive it with the rule $\left(q^{init}, true \xrightarrow{??m_1} \{x\}, q_3\right)$; because of the clique graph, each broadcast message is received by all the processes in the TAHN. Consequently, there does not remain any node in q^{init} ready to receive the message m_2 that is needed to go to q_f. Indeed there does not exist any connectivity graph $G \in$ `CLIQUE`, such that q_f is reachable in the TAHN (G, P).

On the other hand, if we consider the TAHN (G_2, P), then q_f can be reached.

We describe a possible scenario. After 2 times units one of the leaf broadcasts m_1, which is received by the adjacent internal node. After 2 times units this latter node broadcasts m_2 (note that this broadcast happens at global time 4). The message m_2 is received by the root node, which resets its clock, and exactly 2 times unit after (the global time is now 6), one of the two internal nodes, which remained in state q^{init}, broadcasts a message m_3, allowing thus the root node to reach q_f (it receives m_3 exactly two times units after the reception of m_2).

4. Undecidability with Dense Time

In this section, we show undecidability of the reachability problem in TAHN for three different topologies, namely:

- `STAR`(2): *star* connectivity graphs of depth 2 (one root and several rays with two nodes); the undecidability holds even if each process uses a single clock;
- `CLIQUE`: *clique* topologies; for this case, we need at least two clocks per process to get the undecidability;
- `BOUNDED`(5) : *bounded path* topologies with maximal simple path of length at most 5; the undecidability holds even if each process uses a single clock.

In the first two cases, the undecidability results are obtained thanks to a reduction into the reachability problem for timed networks where processes are equipped with two clocks. We will hence first recall the definition of this latter model, which was originally presented in [2]. Afterwards, we will provide the reduction allowing to lift the undecidability result for Timed Network to the case of TAHN.

4.1. Timed Networks

In [2], the authors introduce a model called Timed Network (TN) which can be used to describe a system consisting of an arbitrary number of processes, each of which being a finite-state system operating on real-valued clocks. The differences between the TN model and TAHN can be summarized as follows:

1. A TN contains a distinguished *controller* that is a finite-state automaton without any clocks [2] (note that adding clocks to the controller does not affect our results).
2. Each process in a TN may communicate with all the other processes and hence it is not meaningful to describe connectivity graphs in the case of TN.
3. Communication takes place through rendez-vous between fixed sets of processes rather than broadcast messages.

Following [2], we provide the syntax and the semantics of Timed Networks.

4.1.1. Syntax

Definition 2. *A Timed Network (TN)* $\mathcal{N}$ *is a tuple* $(Q^{ctrl}, Q^{proc}, X, \mathcal{R}, q^{init}_{ctrl}, q^{init}_{proc})$ *where* Q^{ctrl} *is a finite set of* controller states*,* Q^{proc} *is a finite set of* process states*,* X *is finite set of clocks,* $q^{init}_{ctrl} \in Q^{ctrl}$ *is an initial controller stater,* $q^{init}_{proc} \in Q^{proc}$ *is an initial process state and* $\mathcal{R}$ *is a finite set of* rules*. A rule is of the form:*

$$\left[\begin{array}{c} q_0 \\ \rightarrow \\ q'_0 \end{array}\right] \quad \left[\begin{array}{c} q_1 \\ g_1 \rightarrow R_1 \\ q'_1 \end{array}\right] \quad \cdots \quad \left[\begin{array}{c} q_n \\ g_n \rightarrow R_n \\ q'_n \end{array}\right]$$

such that $q_0, q'_0 \in Q^{ctrl}$*, and,* $q_i, q'_i \in Q^{proc}$*,* $g_i \in \mathcal{G}(X)$ *and* $R_i \in 2^X$ *for all* $i \in \{1, \ldots, n\}$*.*

4.1.2. Operational semantics

As for TAHN, we give the semantics associated to a TN in term of a timed transition system. We consider a TN $\mathcal{N} = \left(Q^{ctrl}, Q^{proc}, X, \mathcal{R}, q^{init}_{ctrl}, q^{init}_{proc}\right)$. A configuration γ is a tuple of the form $(I, q, \mathcal{Q}, \mathcal{X})$ with:

- I is a finite set of indices;
- $q \in Q^{ctrl}$;
- $\mathcal{Q} : I \mapsto Q^{proc}$;
- $\mathcal{X} : I \mapsto [X \rightarrow \mathbb{R}^{\geq 0}]$.

Intuitively, the configuration refers to the controller whose state is q, and to a finite set of processes, each one associated to an element of I. The mapping $\mathcal{Q}$ describes the states of the processes and the mapping $\mathcal{X}$ their associated clock values. More precisely, for $i \in I$ and $x \in X$, $\mathcal{X}(i)(x)$ gives the value of clock x in the process of index i. We use $|\gamma|$ to denote the number of processes in γ, i.e., $|\gamma| = |I|$. Let $\mathcal{C}_{\mathcal{N}}$ be the set of configurations of $\mathcal{N}$. A configuration $\gamma^{init} = (I, q, \mathcal{Q}, \mathcal{X})$ is said to be *initial* if $q = q^{init}_{ctrl}$, $\mathcal{Q}(i) = q^{init}_{proc}$, and $\mathcal{X}(i)(x) = 0$ for each $i \in I$ and $x \in X$. This means that an execution of a timed network starts from a configuration where the controller and all the processes are in their initial states, and the clock values are all equal to 0. Note that there is an infinite number of initial configurations, namely one for each finite index set I.

Before we give the formal definition of the transition relation associated to $\mathcal{N}$, let us explain intuitively the behaviour of a rule of the form:

$$\left[\begin{array}{c} q_0 \\ \rightarrow \\ q'_0 \end{array}\right] \quad \left[\begin{array}{c} q_1 \\ g_1 \rightarrow R_1 \\ q'_1 \end{array}\right] \quad \cdots \quad \left[\begin{array}{c} q_n \\ g_n \rightarrow R_n \\ q'_n \end{array}\right]$$

Such a rule is enabled from a given configuration, if the state of the controller is q_0 and if there are n processes with states $q_1, \cdots, q_n$ whose clock values satisfy the corresponding guards. The rule is then executed by simultaneously changing the state of the controller to q'_0, the states of the n processes to $q'_1, \cdots, q'_n$ and by resetting the clocks belonging to the sets $R_1, \ldots, R_n$.

The semantics associated to the TN $\mathcal{N}$ is given by the timed transition system $(\mathcal{C}_\mathcal{N}, \longrightarrow_\mathcal{N})$ where the transition relation $\longrightarrow_\mathcal{N} \subseteq \mathcal{C}_\mathcal{N} \times \mathcal{C}_\mathcal{N}$ is defined as the union of a *discrete* transition relation $\longrightarrow_{\mathcal{N},d}$, representing transitions induced by the rules of $\mathcal{N}$ and a *timed* transition relation $\longrightarrow_{\mathcal{N},t}$ which characterizes the elapse of time.

We begin by describing the transition relation $\longrightarrow_{\mathcal{N},d} \subseteq \mathcal{C}_\mathcal{N} \times \mathcal{C}_\mathcal{N}$. For this matter, we define a transition relation $\longrightarrow_{\mathcal{N},r}$ for each rule $r \in \mathcal{R}$ of the TN $\mathcal{N}$. We consider a rule r described as above. Let $\gamma = (I, q, \mathcal{Q}, \mathcal{X})$ and $\gamma' = (I', q', \mathcal{Q}', \mathcal{X}')$ be two configurations in $\mathcal{C}_\mathcal{N}$. We have $\gamma \longrightarrow_{\mathcal{N},r} \gamma'$ if and only if $I = I'$ and there exists an injection $h : \{1, \ldots, n\} \to I$ such that, for all $i \in \{1, \ldots, n\}$:

1. $q = q_0$, $\mathcal{Q}(h(i)) = q_i$ and $\mathcal{X}(h(i)) \models g_i$ (i.e., the rule r is enabled);
2. $q' = q'_0$ and $\mathcal{Q}'(h(i)) = q'_i$ (i.e. the states of the processes are changed according to r);
3. $\mathcal{X}'(h(i)) = \mathcal{X}(h(i))[R_i]$ (i.e. a clock is reset to 0 if it occurs in the set R_i, otherwise its value remains unchanged).
4. $\mathcal{Q}'(j) = \mathcal{Q}(j)$ and $\mathcal{X}'(j) = \mathcal{X}(j)$ for all $j \in I \setminus \{h(i) \mid i \in \{1, \ldots, n\}\}$.

The discrete transition relation $\longrightarrow_{\mathcal{N},d}$ is then equal to $\bigcup_{r \in \mathcal{R}} \longrightarrow_{\mathcal{N},r}$.

We now provide the definition of the timed transition relation $\longrightarrow_{\mathcal{N},t} \subseteq \mathcal{C}_\mathcal{N} \times \mathcal{C}_\mathcal{N}$. Given two configurations $\gamma = (I, q, \mathcal{Q}, \mathcal{X})$ and $\gamma' = (I', q', \mathcal{Q}', \mathcal{X}')$ in $\mathcal{C}_\mathcal{N}$, we have $\gamma \longrightarrow_{\mathcal{N},t} \gamma'$ if and only if $I' = I$, $q' = q$, $\mathcal{Q}' = \mathcal{Q}$ and there exists $\delta \in \mathbb{R}^{\geq 0}$ such that $\mathcal{X}'(i)(x) = \mathcal{X}(i)(x) + \delta$ for all $i \in I$ and all $x \in X$. Hence, as in TAHN, a timed transition has no effect on the states of the different processes but its effect changes the value of the different clocks making them evolve at the same rate.

Finally we define $\longrightarrow_\mathcal{N}$ to be $\longrightarrow_{\mathcal{N},d} \cup \longrightarrow_{\mathcal{N},t}$. Note that if $\gamma \longrightarrow_\mathcal{N} \gamma'$ then the index sets of γ and γ' are identical (by definition of the transition relation) and therefore $|\gamma| = |\gamma'|$. This reflects the fact that in the considered networks, the number of processes is indeed parametric but once fixed it does not change during an execution, in other words there is no dynamic creation or deletion of processes.

4.1.3. State reachability problem

Similarly to the case of TAHN, we will present here the state reachability problem for TN. Here also this problem is parameterized by the number of processes involved in the execution, that is why we do not impose any bounds on the size of the initial configurations and we investigate whether there exists an initial configuration from which the system can reach another configuration in which the controller is in a given control state (for instance an error state).

Let $\mathcal{N} = \left(Q^{ctrl}, Q^{proc}, X, \mathcal{R}, q^{init}_{ctrl}, q^{init}_{proc}\right)$ be a TN. We say that a configuration γ_n in $\mathcal{C}_\mathcal{N}$ is reachable in $\mathcal{N}$ if there exists a finite path $\gamma_0 \longrightarrow_\mathcal{N} \gamma_1 \longrightarrow_\mathcal{N} \cdots \longrightarrow_\mathcal{N} \gamma_n$ in the transition system associated to $\mathcal{N}$. As for TAHN, a controller state q is said to be reachable in $\mathcal{N}$ if there is a reachable configuration of the form $(I, q, \mathcal{Q}, \mathcal{X})$. The TN state reachability problem $\texttt{TN}-\texttt{Reach}(K)$, parametric in the a number of clocks K, is defined as follows:

Input: A TN $\mathcal{N} = \left(Q^{ctrl}, Q^{proc}, X, \mathcal{R}, q^{init}_{ctrl}, q^{init}_{proc}\right)$ with $|X| \leq K$ and a controller state $q \in Q^{ctrl}$;

Output: Is q reachable in $\mathcal{N}$?

As said earlier, Timed Networks have already been introduced in [2] where results for the state reachability problems are also presented. These latter result can be expressed as follows:

Theorem 2. *[4]*

1. $\texttt{TN}-\texttt{Reach}\,(2)$ *is undecidable.*
2. $\texttt{TN}-\texttt{Reach}\,(1)$ *is decidable.*

4.2. Two-Star Topologies

In this section we prove that the reachability problem for the star topology is undecidable even when the rays are restricted to have length 2 and the nodes are restricted to have a single clock. The proof is based on the encoding of a generic TN $\mathcal{N}$ with two clocks per process into a protocol P of TAHNs. We will refer to the clocks inside a process of $\mathcal{N}$ as x_1 and x_2 respectively. For each state q in $\mathcal{N}$, we will have a corresponding state $\kappa(q)$ in the protocol P. Furthermore, we will have a number of auxiliary states in P that we need to perform the simulation. We omit state labels in the automata representation when their names are not relevant.

Given a TN $\mathcal{N} = \left(Q^{ctrl}, Q^{proc}, X, \mathcal{R}, q^{init}_{ctrl}, q^{init}_{proc}\right)$ and a controller state q in $\mathcal{N}$, we define a protocol P with $nbclocks(P) = 1$ together with a local state $\kappa(q)$ satisfying the following property: there exists a $\mathcal{T} = (G, P)$ with $G \in \texttt{STAR}(2)$ such that such that $\kappa(q)$ is reachable in $\mathcal{T}$ iff q is reachable in $\mathcal{N}$. The root of G plays the role of the controller in $\mathcal{N}$, while each ray in G plays the role of one process in $\mathcal{N}$. The local state of a process in $\mathcal{N}$ is stored in the internal node of the corresponding ray. Furthermore, the two clocks x_1 and x_2 of a process are represented respectively by the clock of the internal node and by the clock of the leaf of the ray. For technical reasons, we require that the connectivity G of the considered TAHNs has at least three rays needed in the initialization phase. In case $\mathcal{N}$ has fewer than three processes, the additional rays will not simulate any processes, and remain passive (except during the initialization phase; see below).

Notation. We will assume without loss of generality that the guards present in the TN $\mathcal{N}$ are conjunctions of predicates of the form $k \lhd x$ for $k \in \mathbb{N}$, $\lhd \in \{=, <, \leq, >, \geq\}$, and $x \in X$. In the sequel, (e.g. Fig. 4, 5, and 6), we will write $g(x_j \leftarrow x)$ to denote the guard obtained by first projecting g on the constraints involving only the variable x_j (this is done by deleting the predicates on the other variables), and then by replacing x_j (a clock of $\mathcal{N}$) with x (the clock of P) in the resulting formula. For instance, if g is $x_1 \geq 2 \wedge x_2 = 4$, then $g(x_1 \leftarrow x)$ is equal to $x \geq 2$ and $g(x_2 \leftarrow x)$ equals $x = 4$. For a reset R, we will write $R(x_j \leftarrow x)$ for the reset $\{x\}$ if $x_j \in R$, or for $\emptyset$ otherwise, i.e., we map a

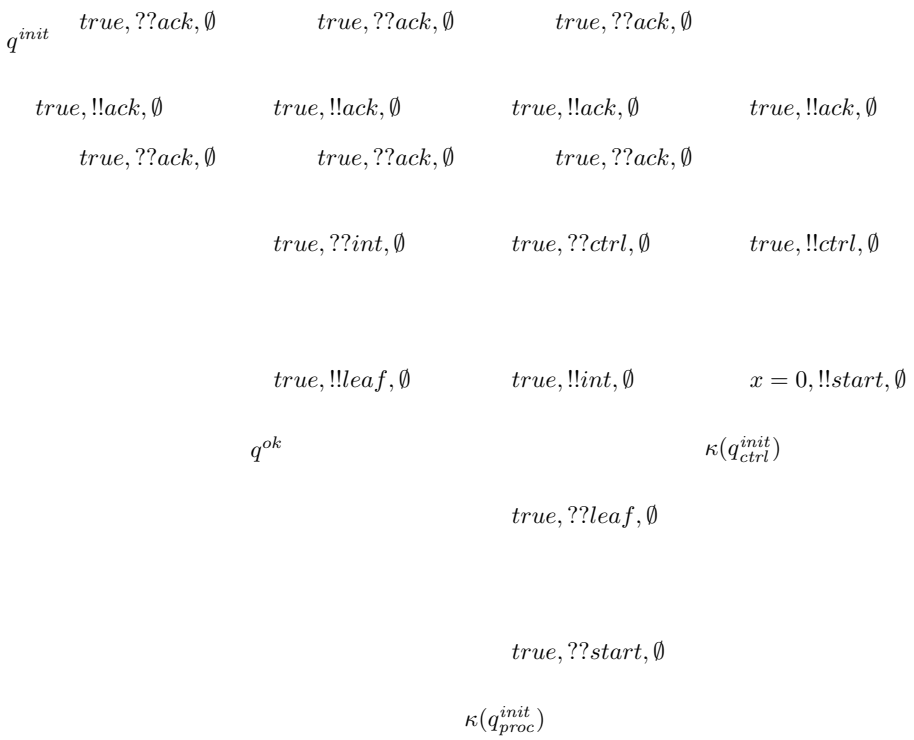


Figure 3: Initializing the simulation

reset on x_j to a reset on the clock variable x of P. For instance, if $R = \{x_1\}$, then $R(x_1 \leftarrow x) = \{x\}$ and $R(x_2 \leftarrow x) = \emptyset$. We are now ready to describe the simulation protocol. It consists of two phases.

Initialization. Recall that the nodes of a TAHN are identical in the sense that they execute the same (predefined) protocol. This means that the nodes are not *a priori* aware of their positions inside the network. The purpose of the initialization phase (Fig. 3) is to identify the nodes that play the roles of the controller and those that play the roles of the different processes.

As shown in Fig. 3, a node starts by broadcasting/receiving an *ack* message to/from his neighbours. The messages of type *ack* are used for the election phase. The elected node becomes the controller of the TN $\mathcal{N}$. To be elected, a node has to receive acknowledgements (messages *ack*) from at least three other processes. This implies that only the root of our star configuration can be elected. Indeed, it is the only node that is connected to more than two other nodes (the internal nodes are connected to two other nodes while the leafs are connected to only one other node). Notice that a node can become a controller via several different sequences of receive and send actions, the important points is that they contain three $??ack$ actions and one $!!ack$ actions in any possible order. Sending $!!ack$ after $??ack$-actions is necessary to synchronize with the

other nodes.

Once the root has become the controller, it will make the internal nodes aware of their positions by sending the broadcast message $ctrl$. Due to the star topology, this message is received only by the internal nodes. A node receiving this broadcast message will initiate a subprotocol defined as follows.

(i) It changes local state to accept the role of internal node.

(ii) It makes the leaf of the ray aware of its position by broadcasting a message int. Such a message is received only by the leaf of the ray and by the root (controller). The root ignores the message.

(iii) The leaf broadcasts the acknowledgement message $leaf$ that can only be received by the internal node of the ray and goes to state q^{ok}.

(iv) The internal node changes state when it receives the acknowledgement and declares itself ready for the next step.

Remark that the internal node and the leaf may choose to ignore performing steps (ii) or (iv). In such a case we say that the protocol fails for the considered ray, otherwise we declare the ray to be successful.

In the last step of the initialization, the root will send one more broadcast where the following step take place:

(i) It changes local state to $\kappa(q^{init}_{ctrl})$ which means that it is now simulating the initial controller state.

(ii) It checks that its clock is equal to 0 which means that the initialization phase has been performed instantaneously.

(iii) The internal nodes of the successful rays will change state to $\kappa(q^{init}_{proc})$. The rest of the nodes will remain passive throughout the rest of the simulation.

Now all the nodes are ready: the root of G in $\mathcal{T}$ is in the initial state of the controller of $\mathcal{N}$; the internal nodes of the successful rays are in the initial states of the processes of $\mathcal{N}$; the leafs are in state q^{ok} and all clocks have values equal to 0.

Simulating Discrete Transitions. Below, we show how $\mathcal{T}$ simulates a rule r of the form

$$\begin{bmatrix} q_0 \\ \to \\ q_0' \end{bmatrix} \quad \begin{bmatrix} q_1 \\ g_1 \to R_1 \\ q_1' \end{bmatrix} \quad \cdots \quad \begin{bmatrix} q_n \\ g_n \to R_n \\ q_n' \end{bmatrix}$$

The behaviour of the root, internal, and leaf nodes is detailed respectively in Fig. 4, Fig. 5, and Fig. 6. At first, the root of G in $\mathcal{T}$ is in the state $\kappa(q_0)$ and executes a transition to reset its clock to 0. The reset is used later to ensure that a simulation step has taken no time. The simulation consists of different phases, where in each phase the root tries to identify a ray that can play the

role of process k for $1 \leq k \leq n$. To find the first ray, it sends a broadcast message $!!sel_1^r$. An (internal) node that receives the message and whose local state is q_1 may either decide to ignore the message or to try to become the node that simulates the first process in the rule. In the latter case it will enter a temporary state from which it initiates a sub-protocol whose goal is to confirm its status as the simulator of the first process. In doing so, the node guesses that its clocks satisfy the values specified by the guard. If the guess is not correct it will eventually be excluded from the rest of the simulation (will remain passive in the rest of the simulation). At the end of this phase, exactly one node will be chosen among the ones that have correctly guessed that their clocks satisfy g_1. The successful node will be the one that plays the role of the first process. The sub-protocol proceeds as follows:

(i) The internal node checks whether the value of its clock satisfies the guard g_1. Recall that each node contains one clock. Since the guard g_1 only compares the clocks x_1, x_2 with constants, the conditions of g_1 can be tested on separate nodes. Namely a node v can deal with the sub-guard involving x_1 and another node w can deal with the sub-guard involving clock x_2. The condition is then satisfied if both v and w acknowledge a certain request. If the clock of the node does not satisfy g_1 (which means that x_1 does not satisfy g_1), the node will remain passive from now on (it has made the wrong guess). Otherwise, the node resets its clock if R_1 contains x_1, and then broadcasts a message (such a message is received by the leaf of the ray).

(ii) The leaf checks whether the value of its clock satisfies the guard g_1 (i.e., if x_2 satisfies g_1); if *yes* it resets its clock if x_2 is included in R_1, and then broadcasts an acknowledgement.

(iii) Upon receiving the above acknowledgement, the internal node declares itself ready for the next step by broadcasting an acknowledgement. At the same time, it moves to new local state and waits for a last acknowledgement from the root (described below) after which it will move to local state $\kappa(q_1')$.

(iv) When the root receives the acknowledgement it sends a broadcast declaring that it has successfully found a ray to simulate the first process. All the nodes in temporary states will now enter local states from which they remain passive. To prevent multiple nodes from playing the role of the first process, the root enters an error state on reception of an acknowledgement from more than one internal node.

The root now proceeds to identify the ray to simulate the second process. This continues until all n processes have been identified. Then the root makes one final move where the following events take place: (i) It moves its local state to $\kappa(q_0')$. (ii) It sends a final broadcast where the node ready for simulating the i^{th} process will now move to $\kappa(q_i')$ for all $i : 1 \leq i \leq n$ (notice that there is at

most one such node for each i). (iii) It checks that its clock is equal to 0 (the simulation of the rule has not taken any time).

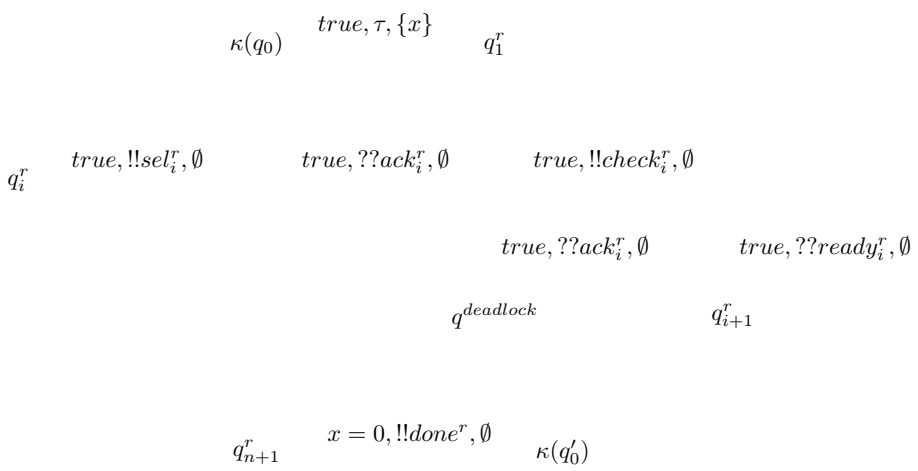


Figure 4: Ray selection: root node

Simulating Timed Transitions. This is done in a straightforward manner by letting time pass in $\mathcal{T}$ by the same amount as in $\mathcal{N}$.

Putting together the different phases, we obtain a complete simulation of a TN with two clocks per node. Since reachability of a given control location (from an arbitrary initial configuration) is undecidable for TN, we deduce the following negative result.

Theorem 3. $\texttt{TAHN}-\texttt{Reach}\,(\texttt{STAR}(2), 1)$ *is undecidable.*

4.3. Cliques and Nodes with Two Clocks

We now show that the reachability problem for the clique topology is undecidable if each node manipulates two clocks. For this purpose, we build a protocol P with $nbclocks(P) = 2$ which will simulate $\mathcal{N}$ on connectivity graphs belonging to the clique topology. In a similar manner to the case of star topologies, the simulation consists of two phases.

Initialization Phase. The purpose of the initialization phase it to choose a node that will simulate the controller. This choice is done non-deterministically through a protocol that is initialized by a broadcast message. Notice that this protocol exists in all the nodes since they run the same pre-defined protocol. The first node which will perform the broadcast will become the controller (from now on we refer to this node as the *controller node*). When the controller node performs the above broadcast it moves to the state $\kappa(q_{ctrl}^{init})$, while all the other nodes will move to $\kappa(q_{proc}^{init})$.

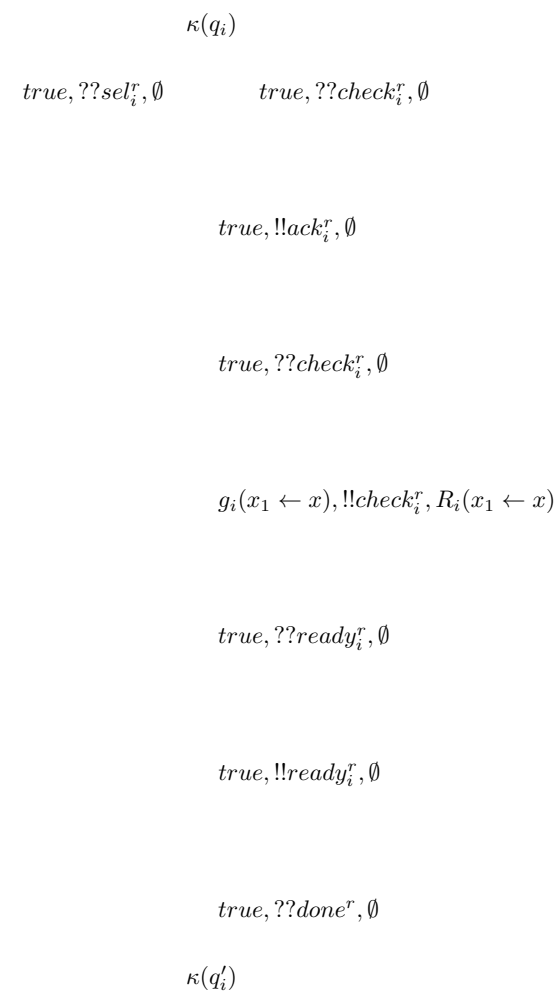


Figure 5: Ray selection: internal node

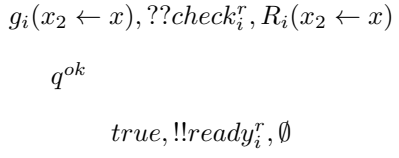


Figure 6: Ray selection: leaf node

Simulating Discrete Transitions. Below, we show how a rule of the form of the previous sub-section is simulated. In a similar manner to the case of stars, the controller node first resets its clock to 0. The simulation again consists of different phases, where in each phase the controller node tries to identify a node that can play the role of process i for $1 \leq i \leq n$. To find the first process it sends a broadcast. A node that receives the broadcast, whose local state is q_1, and whose clocks (x_1 and x_2) satisfy the guard g_1, may decide to ignore the message or try to become the node that simulates the first process in the rule. In the latter case, the node declares itself ready for the next step by broadcasting an acknowledgement. At the same time, it moves to new local state and waits for a last acknowledgement from the controller node (described below) after which it will move to local state $\kappa(q_1')$. To prevent multiple nodes from playing the role of the first process, the controller node enters an error state on reception of an acknowledgement from more than one node. The controller node now proceeds to identify the node to simulate the second process. This continues until all n processes have been identified. Then the controller node performs the same three steps as the ones in the final phase of the simulation described above for stars.

By exploiting undecidability of control state reachability for Timed Networks, we obtain the following theorem.

Theorem 4. $\texttt{TAHN}-\texttt{Reach}\,(\texttt{CLIQUE}, 2)$ *is undecidable.*

4.4. Bounded Path Topologies

Using the result of Theorem 3 we now show that the undecidability proof for the reachability problem can be extended to bounded path topologies. The result uses a reduction to the two-star case, thus we need to consider topologies in which the simple paths can have 5 nodes in order to be able to rebuild stars with rays of depth 2.

For such a reduction, we need a preliminary protocol that discovers a two-star topology in a graph of arbitrary shape but simple paths of (at most) five nodes. The discovery protocol first selects root, internal and leaf candidates and then verifies that they are connected in the desired way by sending all other nodes in their vicinities to a special null state.

The discovery protocol is defined as P with $nbclocks(P) = 1$ with $q^{init} \in Q$ as initial state. We denote by x the clock used by P. We first define three transitions labelled with empty event that non-deterministically select the role of a each node: the root (control state q_0), an internal node (control state r_0) or a leaf (control state s_0) of the star topology. These three rules have then following form:

$$\left(q^{init}, x = 0 \xrightarrow{\tau} \emptyset, q_0\right)$$
$$\left(q^{init}, x = 0 \xrightarrow{\tau} \emptyset, r_0\right)$$
$$\left(q^{init}, x = 0 \xrightarrow{\tau} \emptyset, s_0\right)$$

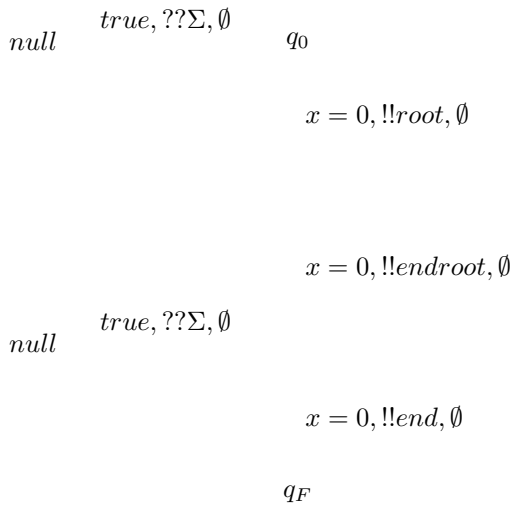


Figure 7: Discovery protocol: node chosen as the root

The behaviour of the root node (state q_0) is given in Figure 7. It broadcasts message *root* to notify its neighbours that it is the root. A node in state q_0 moves to an error state if it receives notifications/requests from other nodes. This protocol ensures that all the nodes in state q_0 connected to the root move to an error state (remember that communication is synchronous). On reception of a message of type *root*, a node in state r_0 runs the protocol in Figure 8. Specifically, it first reacts by sending *ackroot*. This message is needed to send all of its neighbours in states derived from r_0 in the *null* error state. In fact if two adjacent nodes in state r_0 receive a message *root*, the first one sending *ackroot* will send the other one in the state *null*. The *ackroot* message is also needed to ensure that an internal node is never connected to two different root nodes. On reception of a message of type *endroot* from the root, the considered node moves to state r'_0. By the above described properties, when a node reaches state r'_0, then it is connected to at most one root node, and it is not connected to any node which was previously in state r_0 and which is not in state *null*. At this point several $leaf$ nodes can still be connected to nodes in state r'_0. The last part of the protocol deals with this case.

Figures 9 and 10 show the handshaking protocol between leaf and internal nodes. A node in state s_0 sends a message $leaf$ to its adjacent nodes. An internal node can react to the message only in state r'_0, otherwise it goes to an error state. Furthermore, a leaf node that receives the $leaf$ notification moves to an error state. By construction, the following properties then hold.

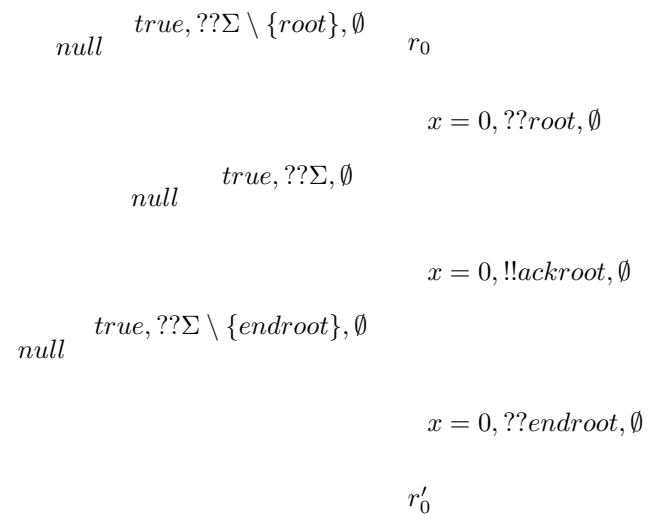


Figure 8: Discovery protocol: node chosen as an internal node (communication with the root)

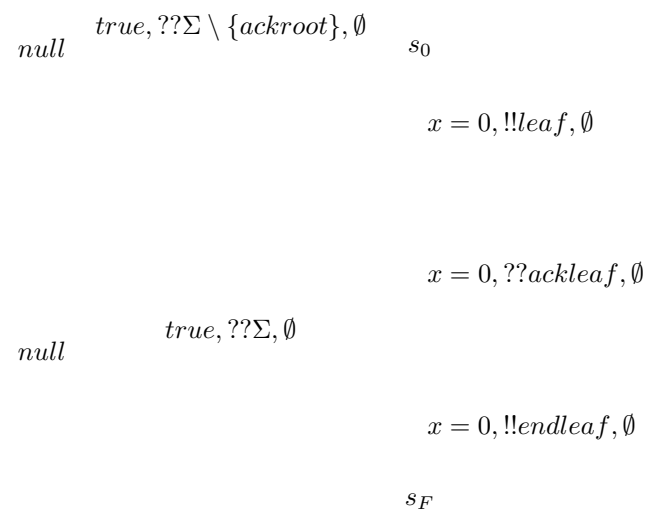


Figure 9: Discovery protocol: node chosen as a leaf

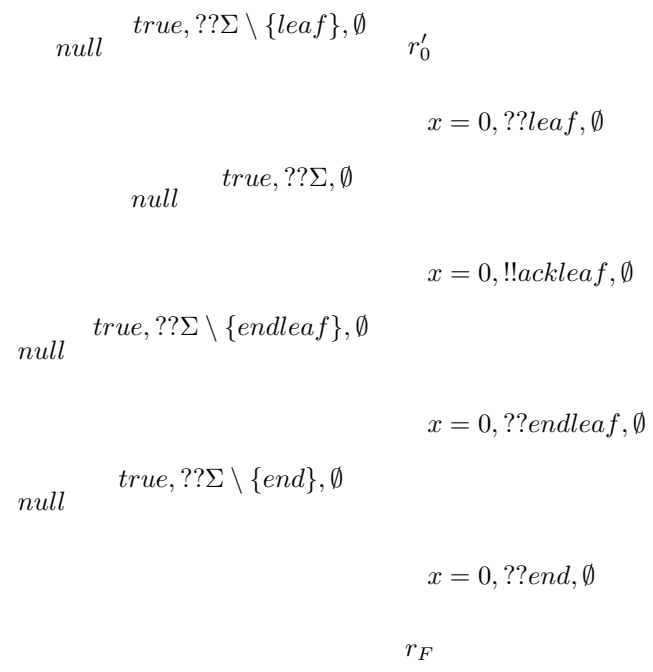


Figure 10: Discovery protocol: node chosen as an internal one (communication with the leaf)

Proposition 1. *In a TAHN $\mathcal{T} = (G, P)$ with $G \in \texttt{BOUNDED}(5)$, all configurations $\gamma = (G, \mathcal{Q}, \mathcal{X})$ reachable in $\mathcal{T}$ satisfy the following properties:*

- *for all nodes $v \in V$ such that $\mathcal{Q}(v) = q_F$, for all $v' \in V$ such that $v \sim v'$, we have $\mathcal{Q}(v') = r_F$ or $\mathcal{Q}(v') = null$,*
- *for all $v \in V$ such that $\mathcal{Q}(v) = r_F$, there exists two nodes $v_1, v_2 \in V$ such that $v \sim v_1$, $v \sim v_2$ and $\mathcal{Q}(v_1) = q_F$ and $\mathcal{Q}(v_2) = s_F$ and for all nodes $v' \in V \setminus \{v_1, v_2\}$, $v' \sim v$ implies $\mathcal{Q}(v') = null$,*
- *for all $v \in V$ such that $\mathcal{Q}(v) = s_F$, there exists at most one vertex $v' \in V$ such that $v \sim v'$ and $\mathcal{Q}(v') \neq null$ and furthermore it is such that $\mathcal{Q}(v') = r_F$.*

In other words if γ is a configuration reachable in a TAHN $\mathcal{T} = (G, P)$ with $G \in \texttt{BOUNDED}(5)$ then all the nodes in the state q_F can be seen as the root node of a star of depth 2 where the internal nodes are in state r_F, the leaves are in state s_F, and all the other nodes connected to these nodes are in state *null* and will not take part to the further communications. Hence from q_F using the protocol proposed in the proof of Theorem 3, we can now simulate the behaviour of a TN as if we were in a star of depth 2. Indeed, combining the above described discovery protocol and the undecidability results for two-star topology, we obtain the following theorem.

Theorem 5. $\texttt{TAHN}-\texttt{Reach}\,(\texttt{BOUNDED}(5), 1)$ *is undecidable.*

Proof. We reduce reachability of a TN $\mathcal{N}$ with two clocks. Specifically, we define a new protocol P' with a single clock that combines the discovery

protocol and the simulation protocol described in the proof of Theorem 3. The final (non-null) states of the discovery protocol (namely q_F, r_F, and s_F) become the initial states of the simulation protocol. The following properties then hold:

- From Theorem 3, we know that there exists a protocol P and a TAHN $\mathcal{T} = (G, P)$ with $G \in \texttt{STAR}(2)$ where it is possible to correctly simulate a TN $\mathcal{N}$.
- From Proposition 1, we know that any star of depth 2 can be obtained with our pre-protocol. Furthermore, the protocol which uses a single clock guarantees the existence of an initial configuration from which we can mark (using states q_F, r_F, and s_F) a subgraph in $\texttt{STAR}(2)$ when the graphs of the initial configurations belongs to $\texttt{BOUNDED}(5)$.

Combining the two properties, we deduce that the there exists a protocol P' using a single clock and a TAHN $\mathcal{T}' = (G', P')$ with $G' \in \texttt{BOUNDED}(5)$ which can correctly simulate a $\mathcal{N}$ with two clocks per node. □

5. Decidability with Dense Time

In the previous sections, we have shown that $\texttt{TAHN}-\texttt{Reach}\,(\texttt{STAR}(2), 1)$ is undecidable. We consider here two other topologies for which reachability becomes decidable when nodes have a single clock, namely the toplogies $\texttt{STAR}(1)$ and $\texttt{CLIQUE}$. For this we mix the technique, proposed in [2], to prove that the reachability problem is decidable in Timed Networks where each process is equipped with a single clock (see Theorem 2) and the one, used in [12], to show that, for TAHN with no clock and restricted to cliques, the reachability problem is decidable (see Theorem 1).

Our proof will be based on the following steps:

1. Define a symbolic way to represent graphs and their associated configurations.
2. Exhibit a well-quasi-ordering over the symbolic configurations which corresponds to the inverse of set inclusion on the associated concrete configurations.
3. Show that it is possible to compute symbolically the predecessors of a symbolic configuration.
4. Give an iterative method to compute all the elements from which a given symbolic configuration can be reached. Termination is ensured by the well-quasi-ordering of symbolic configurations.

5.1. Decidability of $\texttt{TAHN}-\texttt{Reach}\,(\texttt{CLIQUE}, 1)$

In the sequel, we fix a protocol $P = \big(Q, X, \Sigma, \mathcal{R}, q^{init}\big)$.

5.1.1. Symbolic representation of configurations.

We recall that a TAHN is composed both by a connectivity graph $G = (V, E)$ and the protocol P and that the configurations of such a TAHN are of the form $(\mathcal{Q}, \mathcal{X})$ where $\mathcal{Q} : V \mapsto Q$ and $\mathcal{X} : V \mapsto [X \mapsto \mathbb{R}^{\geq 0}]$ is a function that assigns to each node a clock valuation. We will now introduce a way to represent symbolically connectivity graphs and associated configurations. Note that in this part, we focus on TAHN whose connectivity graphs are cliques, hence we only need to take into account the number of nodes in the graphs (the edges can indeed be deduced from this information). The symbolic representation we propose is very similar to the one from [2] used for the analysis of Timed Networks, the main differences being that in our model we do not have a special process playing the role of controller, and the discrete symbolic predecessor relation is different, since we do not deal with rendez-vous communication but with broadcast.

In what follows, we denote by max the maximal constant occurring in the guards of P. Furthermore for a quasi-order $\sqsubseteq$, we use the notation $a \equiv b$ whenever $a \sqsubseteq b$ and $b \sqsubseteq a$ (resp. $\equiv'$ for a quasi-order $\sqsubseteq'$, and, $\equiv_i$ for $\sqsubseteq_i$). A symbolic configuration φ for the protocol P is a tuple $\left(m, \mathcal{Q}^{symb}, \mathcal{X}^{symb}, \sqsubseteq\right)$ where:

- m is a natural number so that $\{1, \ldots, m\}$ is a set of indices for the processes present in the network;
- $\mathcal{Q}^{symb} : \{1, \ldots, m\} \mapsto Q$ maps indices to protocol states;
- $\mathcal{X}^{symb} : \{1, \ldots, m\} \mapsto \{0, \ldots, max\}$ maps process indices to a natural number less or equal than the constant max;
- $\sqsubseteq$ is a total preorder on the set $\{1, \ldots, m\} \cup \{\bot, \top\}$ such that:
  - $\bot$ and $\top$ are respectively the minimal and maximal elements of $\sqsubseteq$ with $\bot \not\equiv \top$;
  - for $j \in \{1, \ldots, m\}$, if $\mathcal{X}^{symb}(j) = max$ then $j \equiv \bot$ or $j \equiv \top$;
  - for $j \in \{1, \ldots, m\}$, if $j \equiv \top$ then $\mathcal{X}^{symb}(j) = max$.

We denote by $\mathcal{S}_P$ the set of symbolic configurations for P. The intuition behind a symbolic configuration $\varphi = \left(m, \mathcal{Q}^{symb}, \mathcal{X}^{symb}, \sqsubseteq\right)$ is the following: it corresponds to a set of clique graphs and associated configurations where at least m processes are involved, and each of this m process is given an index $j \in \{1, \ldots, m\}$ such that $\mathcal{Q}^{symb}(j)$ is the state of the process, $\mathcal{X}^{symb}(j)$ is either the integral part of the clock value or max, and, the relation $\sqsubseteq$ provides an ordering for the processes corresponding to the ordering of the fractional part of their respective clock values. Finally, if $j \equiv \bot$, this means that the clock value of process j is at most max and its fractional part is equal to 0, and, if $j \equiv \top$, then the clock value of process j is strictly greater than max.

Note that the couple $(\mathcal{X}^{symb}, \sqsubseteq)$ corresponds exactly to the clock regions for the m clocks represented in the abstract configuration φ. This region construction was originally introduced in [5] for the analysis of timed automaton since

it allows to get rid of the precise value of the clocks by keeping an abstraction over the possible different values. It was then reused in [2] in the context of timed networks equipped with a single clock. In this latter work, the authors show how to adapt a quasi order over such abstract configurations, since we need the same tool, we adopt in this work the same presentation for abstract configurations.

As done in [2] for the case of Timed Networks, we formalize the previous intuition by providing a formal definition of the set $[\![\varphi]\!]$ of graphs and concrete configurations represented by the symbolic configuration φ. We consider a graph $G = (V, E)$ in `CLIQUE` and a configuration $\gamma = (\mathcal{Q}, \mathcal{X})$ of the TAHN (G, P) and a symbolic configuration $\varphi = \left(m, \mathcal{Q}^{symb}, \mathcal{X}^{symb}, \sqsubseteq\right)$ of P. We have $(G, \gamma) \in [\![\varphi]\!]$ if and only if there exists an injective function $h : \{1, \ldots, m\} \mapsto V$ such that for all $j, j' \in \{1, \ldots, m\}$:

- $\mathcal{Q}(h(j)) = \mathcal{Q}^{symb}(j)$;
- $\min(max, \lfloor \mathcal{X}(h(j)) \rfloor) = \mathcal{X}^{symb}(j)$ (where $\lfloor \mathcal{X}(h(j)) \rfloor$ denotes the integral part of $\mathcal{X}(h(j))$);
- $j \equiv \bot$ if and only if $\mathcal{X}(h(j)) \leq max$ and $frac(\mathcal{X}(h(j))) = 0$ (where $frac(\mathcal{X}(h(j)))$ denotes the fractional part of $\mathcal{X}(h(j))$);
- $j \equiv \top$ if and only if $\mathcal{X}(h(j)) > max$;
- if $j \not\equiv \top$ and $j' \not\equiv \top$ then $frac(\mathcal{X}(h(j))) \leq frac(\mathcal{X}(h(j')))$ if and only if $j \sqsubseteq j'$.

When it exists, such an injective function h will be called a *mapping associated to* $((G, \gamma), [\![\varphi]\!])$. Note that in the above definition, we do not require the number of nodes in G and the number of processes in φ to be the same, but only that each process of φ can be matched with a process of the TAHN (G, P) in the configuration γ. For a set of symbolic configurations $\Phi \subseteq \mathcal{S}_P$, we denote by $[\![\Phi]\!]$ the set $\bigcup_{\varphi \in \Phi} [\![\varphi]\!]$.

We will now equip the set of symbolic configurations $\mathcal{S}_P$ with a quasi-order $\preceq$. Given two symbolic configurations $\varphi_1 = \left(m_1, \mathcal{Q}_1^{symb}, \mathcal{X}_1^{symb}, \sqsubseteq_1\right)$ and $\varphi_2 = \left(m_2, \mathcal{Q}_2^{symb}, \mathcal{X}_2^{symb}, \sqsubseteq_2\right)$, we have $\varphi_1 \preceq \varphi_2$ if and only if there exists an injective mapping $g : \{1, \ldots, m_1\} \mapsto \{1, \ldots, m_2\}$ such that for all $j, j' \in \{1, \ldots, m_1\}$:

- $\mathcal{Q}_2^{symb}(g(j)) = \mathcal{Q}_1^{symb}(j)$;
- $\mathcal{X}_2^{symb}(g(j)) = \mathcal{X}_1^{symb}(j)$;
- $g(j) \equiv_2 \bot$ if and only if $j \equiv_1 \bot$;
- $g(j) \equiv_2 \top$ if and only if $j \equiv_1 \top$;
- $g(j) \sqsubseteq_2 g(j')$ if and only if $j \sqsubseteq_1 j'$.

We have then the following proposition concerning this order.

Proposition 2.

1. *Given* $\varphi_1, \varphi_2 \in \mathcal{S}_P$*, we have* $\varphi_1 \preceq \varphi_2$ *if and only if* $[\![\varphi_2]\!] \subseteq [\![\varphi_1]\!]$.
2. $(\mathcal{S}_P, \preceq)$ *is a well-quasi-order.*

Proof. We will show the first point. Let $\varphi_1 = \left(m_1, \mathcal{Q}_1^{symb}, \mathcal{X}_1^{symb}, \sqsubseteq_1\right)$ and $\varphi_2 = \left(m_2, \mathcal{Q}_2^{symb}, \mathcal{X}_2^{symb}, \sqsubseteq_2\right)$ be two symbolic configurations in $\mathcal{S}_P$.

Suppose that $\varphi_1 \preceq \varphi_2$ and let $g : \{1, \ldots, m_1\} \mapsto \{1, \ldots, m_2\}$ be the corresponding mapping associated to the definition of the quasi-order $\preceq$. We then take $(G, \gamma) \in [\![\varphi_2]\!]$ with $G = (V, E)$ in `CLIQUE` and $\gamma = (\mathcal{Q}, \mathcal{X})$ and let $h : \{1, \ldots, m_2\} \mapsto V$ be the injective function associated to $((G, \gamma), [\![\varphi_2]\!])$. It is then clear that the composed function $h \circ g : \{1, \ldots, m_1\} \mapsto V$ is an injective function matching the condition for $(G, \gamma) \in [\![\varphi_1]\!]$. From this we deduce that $[\![\varphi_2]\!] \subseteq [\![\varphi_1]\!]$.

We assume that $[\![\varphi_2]\!] \subseteq [\![\varphi_1]\!]$. We consider the graph $G = (\{v_1, \ldots, v_{m_2}\}, E)$ in `CLIQUE` and we build the configuration $\gamma = (\mathcal{Q}, \mathcal{X})$ of the TAHN (G, P) in order that it verifies $\mathcal{Q}(v_i) = \varphi_2(i)$ for all $i \in \{1, \ldots, m_2\}$ and $\mathcal{X}$ verifies for all $i, i' \in \{1, \ldots, m_2\}$ the following points:

- if $i \equiv_2 \top$ then $\mathcal{X}(v_i) = max + 1$;
- if $i \not\equiv_2 \top$ then $\lfloor \mathcal{X}(v_i) \rfloor = \mathcal{X}_2^{symb}(i)$;
- if $i \equiv_2 \bot$ then $frac(\mathcal{X}(v_i)) = 0$;
- if $i \not\equiv_2 \bot$ then $frac(\mathcal{X}(v_i)) > 0$;
- if $i \not\equiv_2 \top$ and $i' \not\equiv_2 \top$ and $i \sqsubseteq_2 i'$ and $i \equiv_2 i'$ then $frac(\mathcal{X}(v_i)) = frac(\mathcal{X}(v_{i'}))$;
- if $i \not\equiv_2 \top$ and $i' \not\equiv_2 \top$ and $i \sqsubseteq_2 i'$ and $i \not\equiv_2 i'$ then $frac(\mathcal{X}(v_i)) < frac(\mathcal{X}(v_{i'}))$.

We consider then the bijective function $h_2 : \{1, \ldots, m_2\} \mapsto V$ such that $h_2(i) = v_i$ for all $i \in \{1, \ldots, m_2\}$. It is clear that h_2 satisfies the different conditions of a mapping associated to $((G, \gamma), [\![\varphi_2]\!])$. Hence $(G, \gamma) \in [\![\varphi_2]\!]$ and since $[\![\varphi_2]\!] \subseteq [\![\varphi_1]\!]$, we also have $(G, \gamma) \in [\![\varphi_1]\!]$. Let $h_1 : \{1, \ldots, m_1\} \mapsto V$ be the injective mapping associated to $((G, \gamma), [\![\varphi_1]\!])$. We consider now the composed function $h_2^{-1} \circ h_1 : \{1, \ldots, m_1\} \mapsto \{1, \ldots, m_2\}$. On the way we build the pair (G, γ) and the function h_2 and by definition of the mapping associated to $((G, \gamma), [\![\varphi_1]\!])$, one can easily deduce that the mapping $g = h_2^{-1} \circ h_1$ is effectively injective and satisfies the conditions required in the definition of the quasi-order $\preceq$ and hence this reasoning allows to obtain $\varphi_1 \preceq \varphi_2$.

For what concerns the proof that $(\mathcal{S}_P, \preceq)$ is a well-quasi-order, it can be found in [2], where quasi identical symbolic configurations are used (and the same quasi-order is defined), the only difference being that, in our case, we do not have a controller state in the symbolic configurations. □

5.1.2. Computing the symbolic predecessors.

We describe next how to compute symbolically the set of predecessors of the graphs and configurations described by a symbolic configuration. For a symbolic configuration $\varphi \in \mathcal{S}_P$, we will see how to build a finite set of symbolic configurations corresponding to the union of the two following sets:

$$\begin{array}{rcl} pre_d(\varphi) & = & \{(G,\gamma) \mid G \in \texttt{CLIQUE} \text{ and } \gamma \in \mathcal{C}_{(G,P)} \text{ and} \\ & & \exists \gamma' \in \mathcal{C}_{(G,P)} \text{ s.t. } (G,\gamma') \in [\![\varphi]\!] \text{ and } \gamma \Longrightarrow_{(G,P),d} \gamma'\} \\ pre_t(\varphi) & = & \{(G,\gamma) \mid G \in \texttt{CLIQUE} \text{ and } \gamma \in \mathcal{C}_{(G,P)} \text{ and} \\ & & \exists \gamma' \in \mathcal{C}_{(G,P)} \text{ s.t. } (G,\gamma') \in [\![\varphi]\!] \text{ and } \gamma \Longrightarrow_{(G,P),t} \gamma'\} \end{array}$$

Hence $pre_d(\varphi)$ characterizes the symbolic predecessors for the discrete transition relation and $pre_t(\varphi)$ does the same for the timed transition relation. We will in fact show that it is possible to build a finite set of symbolic configurations Φ such that

$$[\![\Phi]\!] = pre_d(\varphi) \cup pre_t(\varphi)$$

First we begin by the predecessors obtained by considering the discrete transition relation. Following the idea used in [2] for Timed Networks, we begin with seeing how to test whether a guard is satisfied by the clock value of a process in the symbolic configuration. For a guard $g \in \mathcal{G}(X)$ (we recall that $|X| = 1$) in which the maximal constant is max, a symbolic configuration $\varphi = \left(m, \mathcal{Q}^{symb}, \mathcal{X}^{symb}, \sqsubseteq\right)$ and a natural number $j \in \{1, \ldots, m\}$, we define the relation $(\varphi, j) \models g$ inductively as follows:

- $(\varphi, j) \models k \leq x$ for $k \in \{0, \ldots, max\}$ iff $k \leq \mathcal{X}^{symb}(j)$;
- $(\varphi, j) \models k < x$ for $k \in \{0, \ldots, max\}$ iff either $k < \mathcal{X}^{symb}(j)$ or $(k = \mathcal{X}^{symb}(j)$ and $\bot \sqsubseteq j$ and $j \not\equiv \bot)$;
- $(\varphi, j) \models k \geq x$ for $k \in \{0, \ldots, max\}$ iff either $k > \mathcal{X}^{symb}(j)$ or $(k = \mathcal{X}^{symb}(j)$ and $j \equiv \bot)$;
- $(\varphi, j) \models k > x$ for $k \in \{0, \ldots, max\}$ iff $k > \mathcal{X}^{symb}(j)$;
- $(\varphi, j) \models k = x$ for $k \in \{0, \ldots, max\}$ iff $k = \mathcal{X}^{symb}(j)$ and $j \equiv \bot$;
- $(\varphi, j) \models g_1 \wedge g_2$ iff $(\varphi, j) \models g_1$ and $(\varphi, j) \models g_1$;
- for what concerns the negation, we assume that there are pushed inwards in the standard way before applying the definition.

Adapting the proof proposed in [2] for a similar result, we can deduce the following lemma about the satisfiability relation $\models$ on symbolic configurations.

Lemma 1. *Let $\varphi = \left(m, \mathcal{Q}^{symb}, \mathcal{X}^{symb}, \sqsubseteq\right)$ be a symbolic configuration, $G \in$ CLIQUE be a connectivity clique and $\gamma = (\mathcal{Q}, \mathcal{X})$ be a concrete configuration such that $(G, \gamma) \in [\![\varphi]\!]$ and let g be a guard in $\mathcal{G}(X)$ (for which the maximal appearing constant is max). Then for any mapping h associated to $((G, \gamma), [\![\varphi]\!])$ and $j \in \{1, \ldots, m\}$, we have that $(\varphi, j) \models g$ if and only if $\mathcal{X}(h(j)) \models g$.*

We will now show, for each rule $r \in \mathcal{R}$ of P and each symbolic configuration $\varphi \in \mathcal{S}_P$, how to compute the set $\texttt{Pre}(r, \varphi)$ of symbolic configurations corresponding to the symbolic predecessors of φ with respect to the rule r. Let $r = \left(q, g \xrightarrow{e} R, q'\right)$ be a rule of the protocol P and $\varphi = \left(m, \mathcal{Q}^{symb}, \mathcal{X}^{symb}, \sqsubseteq\right)$ be a symbolic configuration. We now provide the conditions for a symbolic configuration $\varphi_2 = \left(m_2, \mathcal{Q}_2^{symb}, \mathcal{X}_2^{symb}, \sqsubseteq_2\right)$ to belong to the set $\texttt{Pre}(r, \varphi)$. This definition is done by a case analysis on the message labeling the rule r. We have $\varphi_2 \in \texttt{Pre}(r, \varphi)$ iff $m \leq m_2 \leq m + 1$ and one of the following conditions is satisfied:

1. $e = !!a$ and there exists $j \in \{1, \ldots, m_2\}$ such that, if $m_2 = m + 1$, then $j = m + 1$, and such that $\mathcal{Q}_2^{symb}(j) = q$ and $\mathcal{X}_2^{symb}(j) \models g$ and such that the following conditions are satisfied: :
   - if $m_2 = m$, then $\mathcal{Q}^{symb}(j) = q'$ and
     - if $R = \emptyset$ then $\mathcal{X}^{symb}(j) = \mathcal{X}_2^{symb}(j)$ and $j \equiv \bot$ iff $j \equiv_2 \bot$, and, $j \equiv \top$ iff $j \equiv_2 \top$;
     - if $R \neq \emptyset$ then $\mathcal{X}^{symb}(j) = 0$ and $j \equiv \bot$;
   - for all $i \in \{1, \ldots, m_2\}$ such that $i \neq j$ we have:
     - either, there does not exists in $\mathcal{R}$ a rule $\left(q'', g' \xrightarrow{??a} R', q'''\right)$ such that $\mathcal{Q}_2^{symb}(i) = q''$ and $\mathcal{X}_2^{symb}(i) \models g'$, then we have $\mathcal{Q}_2^{symb}(i) = \mathcal{Q}^{symb}(i)$ and $\mathcal{X}_2^{symb}(i) = \mathcal{X}^{symb}(i)$, and, $i \equiv \bot$ iff $i \equiv_2 \bot$, and, $i \equiv \top$ iff $i \equiv_2 \top$;
     - or, there exists in $\mathcal{R}$ a rule of the form $\left(q'', g' \xrightarrow{??a} R', q'''\right)$ such that $\mathcal{Q}_2^{symb}(i) = q''$ and $\mathcal{X}_2^{symb}(i) \models g'$ and $\mathcal{Q}^{symb}(i) = q'''$ and:
       - either $R' = \emptyset$ and $\mathcal{X}^{symb}(i) = \mathcal{X}_2^{symb}(i)$ and $i \equiv \bot$ iff $i \equiv_2 \bot$, and, $i \equiv \top$ iff $i \equiv_2 \top$;
       - or, $R' \neq \emptyset$ and $\mathcal{X}^{symb}(i) = 0$ and $i \equiv \bot$.
   - for all $i, i' \in \{1, \ldots . m\}$, if $i \not\equiv \bot$ and $i' \not\equiv \bot$, then $i \sqsubseteq_2 i'$ iff $i \sqsubseteq i'$.
2. $e = \tau$ and there exists $j \in \{1, \ldots, m_2\}$ such that, if $m_2 = m + 1$, then $j = m + 1$, and such that $\mathcal{Q}_2^{symb}(j) = q$ and $\mathcal{X}_2^{symb}(j) \models g$ and such that the following conditions are satisfied: :
   - if $m_2 = m$, then $\mathcal{Q}^{symb}(j) = q'$ and
     - if $R = \emptyset$ then $\mathcal{X}^{symb}(j) = \mathcal{X}_2^{symb}(j)$ and $j \equiv \bot$ iff $j \equiv_2 \bot$, and, $j \equiv \top$ iff $j \equiv_2 \top$;
     - if $R \neq \emptyset$ then $\mathcal{X}^{symb}(j) = 0$ and $j \equiv \bot$;
   - for all $i \in \{1, \ldots, m_2\}$ such that $i \neq j$, we have $\mathcal{Q}_2^{symb}(i) = \mathcal{Q}^{symb}(i)$ and $\mathcal{X}_2^{symb}(i) = \mathcal{X}^{symb}(i)$, and, $i \equiv \bot$ iff $i \equiv_2 \bot$, and, $i \equiv \top$ iff $i \equiv_2 \top$.
   - for all $i, i' \in \{1, \ldots . m\}$, if $i \not\equiv \bot$ and $i' \not\equiv \bot$, then $i \sqsubseteq_2 i'$ iff $i \sqsubseteq i'$.

The intuition behind the definition of the set $\texttt{Pre}(r, \varphi)$ is that first we consider only rules performing a broadcast or a local action, because the rules labelled with receptions are performed together with a broadcast. Then for a rule $\left(q, g \xrightarrow{e} R, q'\right)$ we need to ensure that, in the set of predecessors, the control state q is present, for this reason, either we add a process to the symbolic configuration (when $m_2 = m+1$) which will perform the broadcast or the internal action, or, we consider that a process present in φ does this action (in that case $m_2 = m$). It is useless to add additional receiver nodes in the symbolic representation of configurations. They will produce redundant configurations.

In the case of a broadcast, we have to ensure that all the processes that could react, have reacted to the broadcast message, whereas in the case of an empty event, we need to ensure that the state of the other processes stays unchanged in the symbolic configuration. Finally, in $\texttt{Pre}(r, \varphi)$, we have to include all the possible symbolic clock mappings which satisfy the conditions of the fired rules, as well as the associated possible reset of the second clocks.

Note that by definition of symbolic configurations, we know that there exists a finite set of symbolic configurations of the form $\left(m, \mathcal{Q}^{symb}, \mathcal{X}^{symb}, \sqsubseteq\right)$ for a fixed m, this allows us to deduce that $\texttt{Pre}(r, \varphi')$ can be computed since it is finite. Furthermore, by definition of $pre(\varphi)$ and by the way we build the set $\texttt{Pre}(r, \varphi)$, we can show that the symbolic configuration $\bigcup_{r \in \mathcal{R}} \texttt{Pre}(r, \varphi)$ represents symbolically all the configuration in $pre_d(\varphi)$. In fact the previous construction covers all the possible cases. This allows us to state the next result.

Lemma 2.

- $\texttt{Pre}(r, \varphi)$ *is computable for all rules* $r \in \mathcal{R}$.
- $pre_d(\varphi) = [\![\bigcup_{r \in \mathcal{R}} \texttt{Pre}(r, \varphi)]\!]$.

Example. We show an example of computation for the set $\texttt{Pre}(r, \varphi)$. For this purpose, we use the protocol given in Figure 1 of Section 3. We take the symbolic configuration $\varphi = (\{1\}, \{1 \mapsto q_f\}, \{1 \mapsto 2\}, \bot \equiv 1 \sqsubseteq \top)$ with a single process whose associated state is q_f and its associated symbolic clock value is 2 and we consider the rule $r = \left(q^{init}, (x > 1) \xrightarrow{!!m_3} \emptyset, q_2\right)$. Then in $\texttt{Pre}(r, \varphi)$, we have the symbolic configuration $(\{1, 2\}, \{1 \mapsto q_4, 2 \mapsto q^{init}\}, \{1 \mapsto 2, 2 \mapsto 2\}, \bot \equiv 1 \sqsubseteq 2 \sqsubseteq \top)$. In fact, it is possible to have predecessors from the symbolic configuration φ considering the rule r but, for this, we need to add a process to the symbolic configurations, which will represent the process performing the broadcast of m_3. Note that on the other hand, if we took the following symbolic configuration $\varphi' = (\{1\}, \{1 \mapsto q_4\}, \{1 \mapsto 2\}, \bot \equiv 1 \sqsubseteq \top)$ instead of φ, then the set $\texttt{Pre}(r, \varphi')$ would have been empty. In fact, it is not possible to have configurations with processes in state q_4 and associated clock value equal to 2 after the transition r has been fired, because the broadcast of the message m_3 would have sent all such processes in state q_f (since we are considering clique connectivity graphs exclusively).

For what concerns the set $pre_t(\varphi)$, the rules of the systems are not taken into account and hence in the case of TAHN computing this set is exactly the same as in Timed Networks, we can consequently reuse the result proved in [2].

Lemma 3. *[2] There exists a computable finite set of symbolic configurations* Φ *such that* $[\![\Phi]\!] = pre_t(\varphi)$.

Sketch of proof We can in fact characterize the symbolic configurations belonging to the set Φ. For a configuration $\varphi = \left(m, \mathcal{Q}^{symb}, \mathcal{X}^{symb}, \sqsubseteq\right)$ a configuration $\varphi_2 = \left(m_2, \mathcal{Q}_2^{symb}, \mathcal{X}_2^{symb}, \sqsubseteq_2\right)$ will belong to the set Φ representing $pre_t(\varphi)$ if it has the same number of process (i.e. $m = m_2$), the state mapping is the same (i.e. $\mathcal{Q}_2^{symb} = \mathcal{Q}^{symb}$) and for what concerns $\mathcal{X}_2^{symb}$ and the relation $\sqsubseteq_2$ over fractional part, they can be deduced from $\mathcal{X}^{symb}$ and $\sqsubseteq$ by iteratively making a rotation in the order $\sqsubseteq$ and in the same time by decreasing the integral part of a process whose fractional part is 0 (i.e. the number of this process is equivalent to $\perp$). Since the details of this construction are exactly the same as in Section 6.2 of [2], we do not provide them here. □

According to the two previous lemmas, for a symbolic configuration φ we can compute a finite set of predecessor symbolic configurations of φ. This is summed up by the next lemma.

Lemma 4. *There exists a computable finite set of symbolic configurations* $\texttt{Pre}(\varphi)$ *such that* $[\![\texttt{Pre}(\varphi)]\!] = pre_d(\varphi) \cup pre_t(\varphi)$.

5.1.3. Solving $\texttt{TAHN}-\texttt{Reach}\,(\texttt{CLIQUE}, 1)$

We now show how the facts that we have a well-quasi-order on the set of symbolic configurations which is related to the inclusion of sets of configuration (see Proposition 2) and that we can reason symbolically to compute the symbolic predecessors are enough to solve $\texttt{TAHN}-\texttt{Reach}\,(\texttt{CLIQUE}, 1)$.

For this purpose, we need one more tool to manipulate sets of symbolic configurations. Given two sets of symbolic configurations $\Phi_1, \Phi_2 \subseteq \mathcal{S}_P$, we define the symbolic union of these two sets $\Phi_1 \sqcup \Phi_2$ as follows: $\varphi \in \Phi_1 \sqcup \Phi_2$ iff $(\varphi \in \Phi_1)$ or $(\varphi \in \Phi_2$ and there does not exist $\varphi' \in \Phi_1$ such that $\varphi' \preceq \varphi)$. Note that there are more than one set respecting these conditions, but each time we do the symbolic union we choose non-deterministically one of them.

From Proposition 2, we deduce the following lemma.

Lemma 5. $[\![\Phi_1 \sqcup \Phi_2]\!] = [\![\Phi_1]\!] \cup [\![\Phi_2]\!]$

Proof. Assume $(G, \gamma) \in [\![\Phi_1 \sqcup \Phi_2]\!]$, then there exists $\varphi \in \Phi_1 \sqcup \Phi_2$ such that $(G, \gamma) \in [\![\varphi]\!]$, and since by definition of $\sqcup$ we have $\varphi \in \Phi_1 \cup \Phi_2$, we deduce that $(G, \gamma) \in [\![\Phi_1]\!] \cup [\![\Phi_2]\!]$.

Assume now $(G, \gamma) \in [\![\Phi_1]\!] \cup [\![\Phi_2]\!]$, then there exists $\varphi \in \Phi_1 \cup \Phi_2$ such that $(G, \gamma) \in [\![\varphi]\!]$. We consider then $\varphi' \in \Phi_1 \sqcup \Phi_2$ such that $\varphi' \preceq \varphi$ (by definition of

$\sqcup$ such a φ' exists). Then thanks to the first item of Proposition 2, we deduce $(G,\gamma) \in [\![\varphi']\!]$. Consequently $(G,\gamma) \in [\![\Phi_1 \sqcup \Phi_2]\!]$. $\square$

We show that if we compute iteratively the symbolic predecessors of a symbolic configuration, then such a computation will converge after a finite number of iterations. We define, for a symbolic configuration $\varphi \in \mathcal{S}_P$, the following sequence of sets of symbolic configurations $(\mathcal{P}_\varphi^i)_{i\in\mathbb{N}}$:

- $\mathcal{P}_\varphi^0 = \{\varphi\}$;
- $\mathcal{P}_\varphi^{i+1} = \mathcal{P}_\varphi^i \sqcup \bigsqcup_{\varphi' \in \mathcal{P}_\varphi^i} \texttt{Pre}(\varphi')$

The next statement shows that the computation of the $\mathcal{P}_\varphi^i$ converges after a finite number of steps and furthermore that the obtained set characterize all the configurations from which it is possible to reach a configuration in $[\![\varphi]\!]$. The second point is quite obvious and the first point is obtained thanks to the fact that $(\mathcal{S}_P, \preceq)$ is a well-quasi-order as said by Proposition 2.

Lemma 6. *There exists $N \in \mathbb{N}$ such that $\mathcal{P}_\varphi^i = \mathcal{P}_\varphi^N$ for all $i \geq N$.*

Proof. We reason by contradiction and suppose that for all $i \in \mathbb{N}$, we have $\mathcal{P}_\varphi^{i+1} \neq \mathcal{P}_\varphi^i$. Since $\mathcal{P}_\varphi^{i+1} = \mathcal{P}_\varphi^i \sqcup \bigsqcup_{\varphi' \in \mathcal{P}_\varphi^i} \texttt{Pre}(\varphi')$, by definition of the operator $\sqcup$, this means that for all $i \in \mathbb{N}$, there exists φ_{i+1} such that $\varphi_{i+1} \in \bigsqcup_{\varphi' \in \mathcal{P}_\varphi^i} \texttt{Pre}(\varphi')$ and for which there does not exist $\varphi'' \in \mathcal{P}_\varphi^i$ such that $\varphi'' \preceq \varphi_{i+1}$. We consider then the infinite sequence $(\varphi_i)_{i\in\mathbb{N}\setminus\{0\}}$ of symbolic configurations in $\mathcal{S}_P$. By construction, this infinite sequence is such that for all $j \geq 1$ there does not exist $i \geq 1$ such that $i < j$ and $\varphi_i \preceq \varphi_j$. This is a contradiction with the claim of Proposition 2 which says that $(\mathcal{S}_P, \preceq)$ is a well-quasi-order. $\square$

In the sequel we will denote $\mathcal{P}_\varphi$ the set $\mathcal{P}_\varphi^N$ defined by the previous lemma. Note that a consequence of this lemma is that such a set is finite and from Lemma 4, we know it can be effectively computed. Furthermore, we have also the following result which states the completeness and soundness of the symbolic reasoning.

Lemma 7.

1. *For all $(G,\gamma) \in [\![\varphi]\!]$, if γ is reachable in $\mathcal{T} = (G,P)$ from the initial configuration γ_0 then $(G,\gamma_0) \in [\![\mathcal{P}_\varphi]\!]$.*
2. *For all $(G,\gamma_0) \in [\![\mathcal{P}_\varphi]\!]$, there exists a reachable configuration γ in $\mathcal{T} = (G,P)$ such that $(G,\gamma) \in [\![\varphi]\!]$.*

Proof. Let $(G,\gamma) \in [\![\varphi]\!]$. Suppose that γ is reachable in $\mathcal{T} = (G,P)$ from the initial configuration γ_0. This means that there exists fromγ_0 a finite path of the form $\gamma_0 \Longrightarrow_\mathcal{T} \gamma_1 \Longrightarrow_\mathcal{T} \cdots \Longrightarrow_\mathcal{T} \gamma_n$ in $\mathcal{T}$ with $\gamma_n = \gamma$. But thanks to Lemma 4, fixing $\varphi_n = \varphi$,we know that for all $i \in \{0,\ldots,n-1\}$, there exists φ_i such that $\varphi_i \in \texttt{Pre}(\varphi_{i+1})$ and $(G,\gamma_i) \in [\![\varphi_i]\!]$. Then using Lemma 5 and the definition of $\mathcal{P}_\varphi$, we deduce that $(G,\gamma_0) \in [\![\mathcal{P}_\varphi]\!]$.

Similarly, if we suppose $(G, \gamma_0) \in [\![\mathcal{P}_\varphi]\!]$, thanks to Lemma 4 and 5 and by definition of $\mathcal{P}_\varphi$, we know that there exists a finite path of the form $\gamma_0 \Longrightarrow_{\mathcal{T}} \gamma_1 \Longrightarrow_{\mathcal{T}} \cdots \Longrightarrow_{\mathcal{T}} \gamma_n$ in $\mathcal{T} = (G, P)$ such that $(G, \gamma_n) \in [\![\varphi]\!]$. □

For a control state $q \in Q$, we build the (finite) set of symbolic configurations Φ_q such that a symbolic configuration $\varphi = \big(m, \mathcal{Q}^{symb}, \mathcal{X}^{symb}, \sqsubseteq\big)$ belongs to this set if and only if $m = 1$ and $\mathcal{Q}^{symb}(1) = q$. And we define $\mathcal{P}_{\Phi_q}$ as the set $\bigcup_{\varphi \in \Phi_q} \mathcal{P}_\varphi$. Using the previous lemma, we can deduce that there exists a TAHN $\mathcal{T} = (G, P)$ with $G \in \texttt{CLIQUE}$ such that q is reachable in $\mathcal{T}$ iff we have a symbolic configuration $\varphi_0 \in \mathcal{P}_{\Phi_q}$ such that $\varphi_0 = \Big(m_0, \mathcal{Q}_0^{symb}, \mathcal{X}_0^{symb}, \sqsubseteq_0\Big)$ verifies the following points:

- $\mathcal{Q}_0^{symb}(i) = q^{init}$ for all $i \in \{1, \ldots, m_0\}$;
- $\mathcal{X}_0^{symb}(i) = 0$ for all $i \in \{1, \ldots, m_0\}$;
- $i \equiv_0 \perp$ for all $i \in \{1, \ldots, m_0\}$.

Note that this last condition can be effectively tested on the set of symbolic configurations $\mathcal{P}_{\Phi_q}$ which is finite and computable. Hence this allows us to state the main result of this section.

Theorem 6. $\texttt{TAHN}-\texttt{Reach}\,(\texttt{CLIQUE}, 1)$ *is decidable.*

5.2. Decidability of $\texttt{TAHN}-\texttt{Reach}\,(\texttt{STAR}(1), 1)$

A similar positive result can be obtained for TAHN with 1 clock restricted to star connectivity graphs of depth 1. The only difference with the previous result is that in a star of depth 1 we have to distinguish the root (the central node) from the leaves. In fact, when the root performs a broadcast, it is transmitted to all the leaf nodes, but when a leaf performs it, only the root can receive it. However the previous proof can be easily adapted to this case. The main trick consists in using symbolic configurations of the form $\big(m, \mathcal{Q}^{symb}, \mathcal{X}^{symb}, \sqsubseteq\big)$, as for the case of cliques, except that this time the index of the processes will go from 0 to m and 0 will be the index of the central node. The rest of the proof is then very similar to the previous construction; the bigger difference being in the computation of symbolic discrete predecessors, where one need to make the difference with a broadcast from the process 0 and one from the other processes. This allows us to state our second decidability result for the reachability problem restricted to protocols equipped with a single clock.

Theorem 7. $\texttt{TAHN}-\texttt{Reach}\,(\texttt{STAR}(1), 1)$ *is decidable.*

We point out the fact, that such a reasoning could not be adapted for star topologies of depth strictly bigger than 1, because in that case we would need to have a relation in the symbolic configurations to know which process is connected to which other processes, and such a relation will break the possibility to have a well-quasi-order on the symbolic configurations (as a matter of fact, we have seen previously that the reachability problem is undecidable when considering protocols with a single clocks and star topologies of depth 2).

6. Decidability with Discrete Time

In this section we consider the state reachability problem for Discrete Time Ad Hoc Networks (DTAHN). In this model clocks range over the natural numbers instead of the reals. When using discrete time, it is enough to consider time steps that advance the clocks one unit per time. Furthermore, we can restrict the valuation of clocks to the finite range $\Omega = \{0, \ldots, \mu\}$ where $\mu = max+1$ and max is the maximum constant used in the protocol rules. This follows from the fact that, as soon as the clock associated to variable x reaches a value greater than or equal to μ, guards of the form $x > c$ [resp. $x < c$] remain enabled [resp. disabled] forever. Therefore, beyond μ we need not distinguish between different values for the same clock [4].

Given a protocol P and a topology $G = (V, E)$, a configuration γ of the associated DTAHN $\mathcal{D}$ is a pair $(\mathcal{Q}, \mathcal{X})$ defined as for TAHN except that the clock valuation mapping is of the form $\mathcal{X} : V \mapsto [X \mapsto \Omega]$. We denote by $\mathcal{C}_\mathcal{D}$ the set of configuration of $\mathcal{D}$. Initial configurations are defined as for TAHN. In the sequel, to simplify the handling of transition guards, without loss of generality we will assume that guards occurring in rules of a protocol $P = \left(Q, X, \Sigma, \mathcal{R}, q^{init}\right)$ have form $\bigwedge_{x \in X} x \geq a_x^g \wedge x \leq b_x^g$ with $a_x, b_x \in \{0, \ldots, max\}$ for all $x \in X$. This normal form is well-defined since clocks have always an explicit lower bound (which can be 0) and in case they do not have an explicit upper bound we set it to the constant μ. Since clock values range in $\{0, \ldots, \mu\}$, the previous restriction on guards does not affect the semantics. Furthermore, it is possible to encode disjunctions and negations by adding multiple rules between the same two states.

The semantics of the DTAHN $\mathcal{D}$ built over a protocol P is given by the transition system $(\mathcal{C}_\mathcal{D}, \Longrightarrow_\mathcal{D})$. The transition relation $\Longrightarrow_\mathcal{D} \subseteq \mathcal{C}_\mathcal{D} \times \mathcal{C}_\mathcal{D}$ is similar to the one of TAHN for the discrete transition and by replacing the time step by a discrete time step. For configurations $\gamma = (\mathcal{Q}, \mathcal{X})$ and $\gamma' = (\mathcal{Q}', \mathcal{X}')$, we write $\gamma \Longrightarrow_\mathcal{D} \gamma'$ iff these two configurations are in relation following the local or broadcast rules defined for TAHN, or via a discrete time step defined as follows: For all $v \in V$ and $x \in X$, the following conditions are satisfied: $\mathcal{Q}(\gamma') = \mathcal{Q}(\gamma)$, $\mathcal{X}'(v)(x) = \mathcal{X}(v)(x) + 1$, if $\mathcal{X}(v)(x) < \mu$ $\mathcal{X}'(v)(x) = \mathcal{X}(v)(x) = \mu$, otherwise.

$$
\begin{array}{cccccccc}
q^{init},0 & q^{init},0 & & q^{init},2 & q^{init},2 & & q_1,2 & q_3,0 \\
 & & \Longrightarrow_\mathcal{D} & & & \Longrightarrow_\mathcal{D} & & & \Longrightarrow_\mathcal{D} \\
q^{init},0 & q^{init},0 & & q^{init},2 & q^{init},2 & & q^{init},2 & q^{init},2 \\
\\
q_1,3 & q_3,2 & & q_1,3 & q_3,2 & & & \\
 & & \Longrightarrow_\mathcal{D} & & & & & \\
q^{init},3 & q^{init},3 & & q^{init},3 & q_4,0 & & &
\end{array}
$$

Figure 11: An example of discrete time execution

Example.. On Figure 11, we present the example of a discrete time execution for the DTAHN composed of the protocol given in Figure 1 and of the graph represented in the Figure 11. As we will see later, it is often convenient to represent the graph together with the configuration. Note that we have labelled the node of the graph with the associated control states and clock value (the protocol of Figure 1 is equipped of a single clock). This run corresponds to the following step: a discrete time step of two units, then a broadcast of message m_1 then a discrete time steps of two units and finally a broadcast of message m_2 . Note that we perform the second time step, some clocks get stucked to the maximal value 3 as described by the operational semantics for DTAHN.

For a topology class *Top* and $K \geq 0$, $\texttt{DTAHN}-\texttt{Reach}\,(\mathit{Top}, K)$ denotes the state reachability problem for the new model. We show next that state reachability is decidable when restricting the topology to the class of bounded path graphs $\texttt{BOUNDED}(N)$ for some $N > 1$.

In the sequel we consider a DTAHN $\mathcal{D}$ built over a protocol P. We first introduce an ordering between the configurations with connectivity graph. For this purpose, it is convenient to embed the connectivity graph G in the representation of a configuration. Specifically, we consider extended configurations defined by triples of the form $\gamma = (G, \mathcal{Q}, \mathcal{X})$. Given two (extended) configurations $\gamma = (G, \mathcal{Q}, \mathcal{X})$ with $G = (V, E)$ and $\gamma' = (G', \mathcal{Q}', \mathcal{X}')$ with $G' = (V', E')$ in $\mathcal{C}_\mathcal{D}$, we will write $\gamma \preceq \gamma'$ iff there exists an injective function $h : V \mapsto V'$ such that: $\forall u, u' \in V$, $(u, u') \in E$ if and only if $(h(u), h(u')) \in E'$, and $\forall u \in V$, $\mathcal{Q}(u) = \mathcal{Q}'(h(u))$ and $\mathcal{X}(u) = \mathcal{X}'(h(u))$.

In the sequel we will restrict ourselves to configurations whose graphs belong to $\texttt{BOUNDED}(N)$ for some $N > 1$. We define $\mathcal{C}_\mathcal{D}^N$ as the set of configurations $\{(G, \mathcal{Q}, \mathcal{X}) \in \mathcal{C}_\mathcal{D} \mid G \in \texttt{BOUNDED}(N)\}$ and $(\mathcal{C}_\mathcal{D}, \preceq)$ as the ordering over the configurations of $\mathcal{D}$. For a set of configuration $S \subseteq \mathcal{C}_\mathcal{D}$ of the DTAHN $\mathcal{D}$, we denote $\mathit{Pre}(S)$ the set $\{\gamma \in \mathcal{C}_\mathcal{D} \mid \gamma \Longrightarrow_\mathcal{D} \gamma',\ \gamma' \in S\}$. The following properties then holds.

Proposition 3. *The following properties hold:*
(1) $(\mathcal{C}_\mathcal{D}^N, \preceq)$ *is a wqo for all* $N > 1$.
(2) For γ *in* $\mathcal{C}_\mathcal{D}$*, we can algorithmically compute a finite set* B *such that* $\uparrow B = \mathit{Pre}(\uparrow\{\gamma\})$.

Property (1) follows from the observation that $\preceq$ is the induced subgraph relation for graphs with finitely many labels and from the wqo property of this relation proved by Ding in [14]. Properties (2) follows from the results for untimed AHN in [11]. To extend the algorithm for computing a basis for $\mathit{Pre}(\uparrow\gamma')$ described in [11] to discrete time steps we observe that, since the range of clocks is restricted to the interval Ω, we just need to collect all configurations obtained by subtracting in the configuration γ' the same constant value $\delta \geq 0$ s.t. the resulting clock values remain all greater or equal than zero.

Example. Consider a configuration of the protocol of Figure 1 containing a single node whose associated control state is q_f and with clock value equal to 2.

$q_4, 2 \qquad q^{init}, 2 \qquad q_4, 2 \qquad q^{init}, 3$

Figure 12: Example of predecessors

To compute predecessors for this configuration, we assume that we are working over graph in $\texttt{BOUNDED}(2)$. To reach q_f, a process needs to receive a message m_3. Therefor we need to extend the configuration (ensuring we remain in the topology $\texttt{BOUNDED}(2)$) with an additional node that corresponds to a process from which this message has been broadcasted. The resulting configurations are shown in Figure .

From proposition 3, we can apply the general results in [3] to decide state reachability via a backward search algorithm working on upward closed sets of extended configurations represented by their finite basis. The following theorem then holds.

Theorem 8. $\texttt{DTAHN}-\texttt{Reach}\,(\texttt{BOUNDED}(N), K)$ *is decidable for* $N \geq 1, K \geq 0$.

7. Related Work

In [20] German and Sistla propose a general framework for parameterized verification of concurrent systems based on counting abstractions and reductions to Petri nets-like formalisms. The German-Sistla model is defined for fully connected topologies, individual processes modelled via finite-state automata and communication based on rendez-vous synchronization. Parameterized verification of concurrent systems in which the underlying communication topology is modelled as a special class of graphs, e.g., rings, have been proposed in [15, 16, 7, 6]. In [15] Emerson and Namjoshi provide small model properties (cutoff properties) for a token-passing protocols in unidirectional rings that can be applied to prove fragments of indexed CTL* properties. The results have been extended by Aminof et al. in [6]. Decidability for token passing protocols for arbitrary graphs have been studied in [7, 6].

Parameterized verification for broadcast communication has been studied in [15, 17]. A forward, possibly non terminating, reachability algorithm has been proposed in [15]. In [17] Esparza, Finkel and Mayr give a reduction of the problem to coverability in an extension of Petri nets with transfer arcs. Coverability is decidable in this model. The property can be proved by applying the general results in [3, 19].

In [11, 12] the authors study decidability issues for parameterized verification of a concurrent model with broadcast communication and communication topology restricted by a graph, called AHN. The model is an untimed abstraction that can be applied to specify protocols used for Ad Hoc Networks. Variations of the model with node and link failures, asynchronous communication, and local mailboxes has been studied in [10, 13, 9]. In [8] Clemente et al. give decidability results for different classes of topologies for systems defined by communicating automata with FIFO and bag channels.

Model checking for timed automata has been applied to verify protocols for ad hoc networks with a fixed number of nodes in [18]. Models with a discrete global clock and lazy exploration of configurations of fixed size has been considered in [24]. Formal specification languages for timed models of ad hoc networks have been proposed, e.g., in [22]. In contrast to these works, we consider here computability issues for verification of timed ad hoc networks with parametric initial configurations.

Decidability of some cases is proved by resorting to an extension of Timed Networks with Transfer. In the untimed case the combination of rendez-vous and transfer is considered in a model called datanets, an untimed extension of Petri nets in which processes have data taken from an ordered domain [21].

This paper extends with detailed proofs the preliminary work presented at FORMATS '11 [1].

8. Conclusions

We have studied local state reachability for Timed Ad Hoc Networks in different classes of topologies and considering the number of clocks of each node as a parameter. Fig. 13 shows a summary of our analysis. We also mention decidability for DTAHN on cliques since, as for bounded paths, it derives from an application of the theory of wsts. Undecidability for DTAHN on graphs with bounded diameter follows instead from the result obtained in the untimed case in [12].

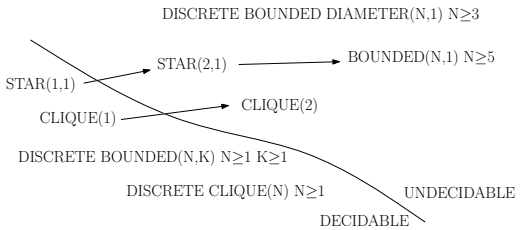


Figure 13: Decidability and undecidability results for TAHN.

[1] P. A. Abdulla, G. Delzanno, O. Rezine, A. Sangnier, and Riccardo Traverso. On the verification of timed ad hoc networks. In *FORMATS*, pages 256–270, 2011.

[2] P. A. Abdulla and B. Jonsson. Model checking of systems with many identical timed processes. *TCS*, 290(1):241–264, 2003.

[3] P.A. Abdulla, K. Cerans, B. Jonsson, and Y.K. Tsay. General decidability theorems for infinite-state systems. In *LICS'96*, pages 313–321. IEEE Computer Society, 1996.

[4] P.A. Abdulla, J. Deneux, and P. Mahata. Multi-clock timed networks. In *LICS'04*, pages 345–354. IEEE Computer Society, 2004.

[5] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[6] B. Aminof, S. Jacobs, A. Khalimov, and S. Rubin. Parameterized model checking of token-passing systems. In *Verification, Model Checking, and Abstract Interpretation - 15th International Conference, VMCAI 2014, San Diego, CA, USA, January 19-21, 2014, Proceedings*, pages 262–281, 2014.

[7] E. M. Clarke, M. Talupur, T. Touili, and H. Veith. Verification by network decomposition. In *CONCUR 2004 - Concurrency Theory, 15th International Conference, London, UK, August 31 - September 3, 2004, Proceedings*, pages 276–291, 2004.

[8] L. Clemente, F. Herbreteau, and G. Sutre. Decidable topologies for communicating automata with FIFO and bag channels. In *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings*, pages 281–296, 2014.

[9] G. Delzanno, A. Sangnier, and R. Traverso. Parameterized verification of broadcast networks of register automata. In *Reachability Problems - 7th International Workshop, RP 2013, Uppsala, Sweden, September 24-26, 2013 Proceedings*, pages 109–121, 2013.

[10] G. Delzanno, A. Sangnier, R. Traverso, and G. Zavattaro. On the complexity of parameterized reachability in reconfigurable broadcast networks. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India*, pages 289–300, 2012.

[11] G. Delzanno, A. Sangnier, and G. Zavattaro. Parameterized verification of ad hoc networks. In *CONCUR'10*, volume 6269 of *LNCS*. Springer, 2010.

[12] 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.

[13] G. Delzanno and R. Traverso. Decidability and complexity results for verification of asynchronous broadcast networks. In *Language and Automata Theory and Applications - 7th International Conference, LATA 2013, Bilbao, Spain, April 2-5, 2013. Proceedings*, pages 238–249, 2013.

[14] G. Ding. Subgraphs and well quasi ordering. *J. of Graph Theory*, 16(5):489–502, 1992.

[15] E. Allen Emerson and K. S. Namjoshi. On model checking for non-deterministic infinite-state systems. In *Thirteenth Annual IEEE Symposium on Logic in Computer Science, Indianapolis, Indiana, USA, June 21-24, 1998*, pages 70–80, 1998.

[16] E. Allen Emerson and K. S. Namjoshi. On reasoning about rings. *Int. J. Found. Comput. Sci.*, 14(4):527–550, 2003.

[17] J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *LICS'99*, pages 352–359. IEEE Computer Society, 1999.

[18] A. Fehnker, L. van Hoesel, and A. Mader. Modelling and verification of the LMAC protocol for wireless sensor networks. In *IFM'07*, volume 4591 of *LNCS*, pages 253–272. Springer, 2007.

[19] A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001.

[20] S. M. German and A. Prasad Sistla. Reasoning about systems with many processes. *J. ACM*, 39(3):675–735, 1992.

[21] R. Lazic, T. Newcomb, J. Ouaknine, A.W. Roscoe, and J. Worrell. Nets with tokens which carry data. *Fund. Inf.*, 88(3):251–274, 2008.

[22] M. Merro, F. F. Ballardin, and E. Sibilio. A timed calculus for wireless systems. In *Proc. of the 3rd Conference on Fundamentals of Software Engineering (FSEN'09)*, volume 5961 of *LNCS*, pages 228–243. Springer, 2010.

[23] M. Saksena, O. Wibling, and B. Jonsson. Graph grammar modeling and verification of Ad Hoc Routing Protocols. In *TACAS'08*, volume 4963 of *LNCS*, pages 18–32. Springer, 2008.

[24] A. Singh, C. R. Ramakrishnan, and S. A. Smolka. Query-based model checking of ad hoc network protocols. In *CONCUR'09*, volume 5710 of *LNCS*, pages 603–619. Springer, 2009.

Distributed local strategies in broadcast networks*

Nathalie Bertrand[1], Paulin Fournier[2], and Arnaud Sangnier[3]

1 **Inria Rennes Bretagne Atlantique**
2 **ENS Rennes, Univ Rennes 1**
3 **LIAFA, Univ Paris Diderot, Sorbonne Paris Cité, CNRS**

Abstract

We study the problems of reaching a specific control state, or converging to a set of target states, in networks with a parameterized number of identical processes communicating via broadcast. To reflect the distributed aspect of such networks, we restrict our attention to executions in which all the processes must follow the same *local strategy* that, given their past performed actions and received messages, provides the next action to be performed. We show that the reachability and target problems under such local strategies are NP-complete, assuming that the set of receivers is chosen non-deterministically at each step. On the other hand, these problems become undecidable when the communication topology is a clique. However, decidability can be regained for reachability under the additional assumption that all processes are bound to receive the broadcast messages.

1998 ACM Subject Classification F.3 Logics and Meanings of Programs, F.1.1 Models of Computation

Keywords and phrases Broadcast Networks, Parameterized Verification, Local strategies

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

Parameterized models for distributed systems. Distributed systems are nowadays ubiquitous and distribution is one of the main paradigms in the conception of computing systems. Conceiving, analyzing, debugging and verifying such systems are tedious tasks which lately received an increased interest from the formal methods community. Considering parametric models with an unknown number of identical processes is a possible approach to tame distributed systems in which all processes share the same code. It has the advantages to allow one to establish the correctness of a system independently of the number of participants, and to ease bugs detection by the possibility to adapt the number of processes on demand.

In their seminal paper on distributed models with many identical entities [14], German and Sistla represent the behavior of a network by finite state machines interacting via 'rendezvous' communications. Variants have then been proposed, to handle different communication means, like broadcast communication [11], token-passing [6, 2], message passing [5] or shared memory [12]. In his nice survey on such parameterized models [10], Esparza shows that minor changes, such as the presence or absence of a controller in the system, can drastically modify the complexity of the verification problems. Another perspective for parametric systems has been proposed by Bollig who studied their expressive power with respect to logics over Message Sequence Charts [4].

* This work is partially supported by the ANR national research program ANR-14-CE28-0002 PACS.

Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 1–15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Broadcast protocols. Among the various parametric models of networks, broadcast protocols, originally studied by Esparza *et al.* [11], have later been analyzed under a new viewpoint, leading to new insights on the verification problems. Specifically, a low level model to represent the main characteristics of ad-hoc networks has been proposed [8]: the network is equipped with a communication topology and processes communicate via broadcast to their neighbors. It was shown that, given a protocol represented by a finite state machine performing internal actions, broadcasts and receptions of messages, the problem of deciding whether there exists an initial communication topology from which one of the processes can reach a specific control state is undecidable. The same holds for the target problem, which asks whether all processes can converge to a set of target states. For both the reachability and the target problems, decidability can however be regained, by considering communication topologies that can change non-deterministically at any moment [7]. Another option to recover decidability of the reachability problem is to restrict the topologies to clique graphs [9], yielding a model equivalent to broadcast protocols.

Local distributed strategies. In this paper, we consider the reachability and target problems under a new perspective, which we believe could also be interesting for other 'many identical processes' models. In such models, the protocol executed by each process is often described by a finite state machine that can be non-deterministic. Therefore it may happen that two processes behave differently, even if they have the same information on what has happened so far in an execution. To forbid such non-truly distributed behaviors, we constrain processes to take the same decisions in case they fired the same sequence of transitions so far. We thus study the reachability and target problems in broadcast protocols restricted to *local strategies*. Interestingly, the notably difficult distributed controller synthesis problem [15] is relatively close to the problem of existence of a local strategy. Indeed a local strategy corresponds to a local controller for the processes executing the protocol and whose role is to resolve the non-deterministic choices.

Our contributions. First we show that the reachability and target problems under local strategies in reconfigurable broadcast networks are NP-complete. To obtain the upper bound, we prove that local strategies can be succinctly represented by a finite tree of polynomial size in the size of the input protocol. This result is particularly interesting, because deciding the existence of a local strategy is intrinsically difficult. Indeed, even with a fixed number of processes, the locality constraint cannot be simply tested on the induced transition system, and *a priori* local strategies may need unbounded memory. From our decidability proofs, we derive an upper bound on the memory needed to implement the local strategies. We also give cutoffs, *i.e.* upper bounds on the minimal number of processes needed to reach or converge to target states. Second we show the two problems to be undecidable when the communication topology is a clique. Moreover, the undecidability proof of the target problem holds even if the locality assumption is dropped. However, the reachability problem under local strategies in clique is decidable (yet non-primitive recursive) for complete protocols, *i.e.* when receptions are always possible from every state.

Due to lack of space, omitted details and proofs can be found in the companion research report [3].

2 Networks of reconfigurable broadcast protocols

In this paper, given $i, j \in \mathbb{N}$ such that $i \leq j$, we let $[i..j] = \{k \mid i \leq k \leq j\}$. For a set E and a natural $\ell > 0$, let E^ℓ be the set of vectors $\mathbf{v}$ of size ℓ over E. For a vector $\mathbf{v} \in E^\ell$ and $i \in [1..\ell]$, $\mathbf{v}[i]$ is the i-th component of $\mathbf{v}$ and $|\mathbf{v}| = \ell$ its size. The notation $\mathcal{V}_E$ stands for

the infinite set $\bigcup_{\ell \in \mathbb{N}\setminus\{0\}} E^\ell$ of all vectors over E. We will use the notation $\mathcal{M}(E)$ to denote the set of multi-sets over E.

2.1 Syntax and semantics

We begin by presenting our model for networks of broadcast protocols. Following [8, 9, 7], we assume that each process in the network executes the same (non-deterministic) broadcast protocol given by a finite state machine where the actions are of three kinds: broadcast of a message m (denoted by $!!m$), reception of a message m (denoted by $??m$) and internal action (denoted by ε).

▶ **Definition 1.** A *broadcast protocol* is a tuple $\mathcal{P} = (Q, q_0, \Sigma, \Delta)$ with Q a finite set of control states; $q_0 \in Q$ the initial control state; Σ a finite message alphabet and $\Delta \subseteq Q \times (\{!!m, ??m \mid m \in \Sigma\} \cup \{\varepsilon\}) \times Q$ a finite set of edges.

We denote by $A(q)$ the set $\{(q, \varepsilon, q') \in \Delta\} \cup \{(q, !!m, q') \in \Delta\}$ containing broadcasts and internal actions (called *active actions*) of $\mathcal{P}$ that start from state q. Furthermore, for each message $m \in \Sigma$, we denote by $R_m(q)$ the set $\{(q, ??m, q') \in \Delta\}$ containing the edges that start in state q and can be taken on reception of message m. We say that a broadcast protocol is *complete* if for every $q \in Q$ and every $m \in \Sigma$, $R_m(q) \neq \emptyset$. Whether protocols are complete or not may change the decidability status of the problems we consider (see Section 4).

We now define the semantics associated with such a protocol. It is common to represent the network topology by an undirected graph describing the communication links [7]. Since the topology may change at any time (such an operation is called reconfiguration), we decide here to simplify the notations by specifying, for each broadcast, a set of possible receivers that is chosen non-deterministically. The semantics of a network built over a broadcast protocol $\mathcal{P} = (Q, q_0, \Sigma, \Delta)$ is given by a transition system $\mathcal{T}_\mathcal{P} = (\Gamma, \Gamma_0, \to)$ where $\Gamma = \mathcal{V}_Q$ is the set of configurations (represented by vectors over Q); $\Gamma_0 = \mathcal{V}_{\{q_0\}}$ is the set of initial configurations and $\to \subseteq \Gamma \times \mathbb{N} \times \Delta \times 2^\mathbb{N} \times \Gamma$ is the transition relation defined as follows: $(\gamma, p, \delta, R, \gamma') \in \to$ (also denoted by $\gamma \xrightarrow{p,\delta,R} \gamma'$) iff $|\gamma| = |\gamma'|$ and $p \in [1..|\gamma|]$ and $R \subseteq [1..|\gamma|] \setminus \{p\}$ and one of the following conditions holds:

- **Internal action:** $\delta = (\gamma[p], \varepsilon, \gamma'[p])$ and $\gamma'[p'] = \gamma[p']$ for all $p' \in [1..|\gamma|] \setminus \{p\}$ (*the p-th process performs an internal action*).
- **Communication:** $\delta = (\gamma[p], !!m, \gamma'[p])$ and $(\gamma[p'], ??m, \gamma'[p']) \in \Delta$ for all $p' \in R$ such that $R_m(\gamma[p']) \neq \emptyset$, and $\gamma'[p''] = \gamma[p'']$ for all $p'' \in [1..|\gamma|] \setminus (R \cup \{p\})$ and for all $p'' \in R$ such that $R_m(\gamma[p'']) = \emptyset$ (*the p-th process broadcasts m to all the processes in the reception set R*).

Obviously, when an internal action is performed, the reception set R is not taken into account. We point out the fact that the hypothesis $|\gamma| = |\gamma'|$ implies that the number of processes remains constant during an execution (there is no creation or deletion of processes). Yet, $\mathcal{T}_\mathcal{P}$ is an infinite state transition system since the number of possible initial configurations is infinite. An *execution* of $\mathcal{P}$ is then a finite sequence of consecutive transitions in $\mathcal{T}_\mathcal{P}$ of the form $\theta = \gamma_0 \xrightarrow{p_0,\delta_0,R_0} \gamma_1 \dots \xrightarrow{p_\ell,\delta_\ell,R_\ell} \gamma_{\ell+1}$ and we denote by $\Theta[\mathcal{P}]$ (or simply Θ when $\mathcal{P}$ is clear from context) the set of all executions of $\mathcal{P}$. Furthermore, we use $nbproc(\theta) = |\gamma_0|$ to represent the number of processes involved in the execution θ.

2.2 Local strategies and clique executions

Our goal is to analyze executions of broadcast protocols under *local strategies*, where each process performs the same choices of edges according to its past history (*i.e.* according to the edges of the protocol it has fired so far).

A *finite path* in $\mathcal{P}$ is either the empty path, denoted by ϵ, or a non-empty finite sequence of edges $\delta_0 \cdots \delta_\ell$ such that δ_0 starts in q_0 and for all $i \in [1..\ell]$, δ_i starts in the state in which δ_{i-1} ends. For convenience, we say that ϵ ends in state q_0. We write $\mathsf{Path}(\mathcal{P})$ for the set of all finite paths in $\mathcal{P}$.

For an execution $\theta \in \Theta[\mathcal{P}]$, we define, for every $p \in [1..nbproc(\theta)]$, the *past* of process p in θ (also referred to as its *history*), written $\pi_p(\theta)$, as the finite path in $\mathcal{P}$ that stores the sequences of edges of $\mathcal{P}$ taken by p along θ. We can now define local strategies which allow us to focus on the executions in which each process performs the same choice according to its past. A *local strategy* σ for $\mathcal{P}$ is a pair (σ_a, σ_r) of functions specifying, given a history, the next active action to be taken, and the reception edge to choose when receiving a message, respectively. Formally $\sigma_a : \mathsf{Path}(\mathcal{P}) \to (Q \times (\{!!m \mid m \in \Sigma\} \cup \{\varepsilon\}) \times Q)$ satisfies, for every $\rho \in \mathsf{Path}(\mathcal{P})$ ending in $q \in Q$, either $A(q) = \emptyset$ or $\sigma_a(\rho) \in A(q)$. Whereas $\sigma_r : \mathsf{Path}(\mathcal{P}) \times \Sigma \to (Q \times \{??m \mid m \in \Sigma\} \times Q)$ satisfies, for every $\rho \in \mathsf{Path}(\mathcal{P})$ ending in $q \in Q$ and every $m \in \Sigma$, either $R_m(q) = \emptyset$ or $\sigma_r(\rho, m) \in R_m(q)$.

Since our aim is to analyze executions where each process behaves according to the same local strategy, we now provide the formal definition of such executions. Given a local strategy σ, we say that a path $\delta_0 \cdots \delta_\ell$ *respects* σ if for all $i \in [0..\ell - 1]$, we have $\delta_{i+1} = \sigma_a(\delta_0 \dots \delta_i)$ or $\delta_{i+1} = \sigma_r(\delta_0 \cdots \delta_i, m)$ for some $m \in \Sigma$. Following this, an execution θ respects σ if for all $p \in [1..nbproc(\theta)]$, we have that $\pi_p(\theta)$ respects σ (i.e. we have that each process behaves as dictated by σ). Finally we define $\Theta_{\mathcal{L}} \subseteq \Theta$ as the set of *local executions* (also called local semantics), that is executions θ respecting a local strategy.

We also consider another set of executions where we assume that every message is broadcast to all the processes of the network (apart from the emitter). Formally, an execution $\theta = \gamma_0 \xrightarrow{p_0,\delta_0,R_0} \dots \xrightarrow{p_\ell,\delta_\ell,R_\ell} \gamma_{\ell+1}$ is said to be a *clique execution* if $R_k = [1, \dots, nbproc(\theta)] \setminus \{p_k\}$ for every $k \in [0..\ell]$. We denote by $\Theta_{\mathcal{C}}$ the set of clique executions (also called clique semantics). Note that clique executions of broadcast networks have been studied in [9] and that such networks correspond to broadcast protocols with no rendez-vous [11]. We will also consider the intersection of these subsets of executions and write $\Theta_{\mathcal{LC}}$ for the set $\Theta_{\mathcal{L}} \cap \Theta_{\mathcal{C}}$ of clique executions which respect a local strategy.

2.3 Verification problems

In this work we study the parameterized verification of the reachability and target properties for broadcast protocols restricted to local strategies. The first one asks whether there exists an execution respecting some local strategy and that eventually reaches a configuration where a given control state appears, whereas the latter problem seeks for an execution respecting some local strategy and that ends in a configuration where all the control states belong to a given target set. We consider several variants of these problems depending on whether we restrict to clique executions or not and to complete protocols or not.

For an execution $\theta = \gamma_0 \xrightarrow{p_0,\delta_0,R_0} \gamma_1 \dots \xrightarrow{p_\ell,\delta_\ell,R_\ell} \gamma_{\ell+1}$, we denote by $\mathsf{End}(\theta) = \{\gamma_{\ell+1}[p] \mid p \in [1..nbproc(\theta)]\}$ the set of states that appear in the last configuration of θ. Reach$[\mathcal{S}]$, the parameterized reachability problem for executions restricted to $\mathcal{S} \in \{\mathcal{L}, \mathcal{C}, \mathcal{LC}\}$ is defined as follows:

Input: A broadcast protocol $\mathcal{P} = (Q, q_0, \Sigma, \Delta)$ and a control state $q_F \in Q$.

Output: Does there exist an execution $\theta \in \Theta_{\mathcal{S}}$ such that $q_F \in \mathsf{End}(\theta)$?
In previous works, the parameterized reachability problem has been studied without the restriction to local strategies; in particular the reachability problem on unconstrained executions is in PTIME [7] and Reach$[\mathcal{C}]$ is decidable and Non-Primitive Recursive (NPR) [9, 11] (it is in fact Ackermann-complete [16]).

Target$[\mathcal{S}]$, the parameterized target problem for executions restricted to $\mathcal{S} \in \{\mathcal{L}, \mathcal{C}, \mathcal{LC}\}$ is defined as follows:
Input: A broadcast protocol $\mathcal{P} = (Q, q_0, \Sigma, \Delta)$ and a set of control states $\mathfrak{T} \subseteq Q$.
Output: Does there exist an execution $\theta \in \Theta_{\mathcal{S}}$ such that $\mathsf{End}(\theta) \subseteq \mathfrak{T}$?
It has been shown that a generalization of the target problem, without restriction to local strategies, can be solved in NP [7]. In this work, we focus on executions under local strategies and we obtain the results presented in the following table:

Reach$[\mathcal{L}]$	Reach$[\mathcal{LC}]$	Target$[\mathcal{L}]$	Target$[\mathcal{LC}]$
NP-complete [Thm. 7]	Undecidable [Thm. 9] Decidable and NPR for complete protocols [Thm. 11]	NP-complete [Thm. 8]	Undecidable [Thm. 9]

Most of the problems listed in the above table are monotone: if, in a network of a given size, an execution satisfying the reachability or target property exists, then, in any bigger network, there also exists an execution satisfying the same property.

▶ **Proposition 2.** *Let θ be an execution in $\Theta_{\mathcal{L}}$ [resp. $\Theta_{\mathcal{LC}}$]. For every $N \geq nbproc(\theta)$, there exists θ' in $\Theta_{\mathcal{L}}$ [resp. $\Theta_{\mathcal{LC}}$] such that $nbproc(\theta') = N$ and $\mathsf{End}(\theta) = \mathsf{End}(\theta')$ [resp. $\mathsf{End}(\theta) \subseteq \mathsf{End}(\theta')$].*

This monotonicity property allows us to look for cutoffs, *i.e.* minimal number of processes such that a local execution with a given property exists. In this work, we provide upper-bounds on these cutoffs for Reach$[\mathcal{L}]$ (Proposition 6) and Target$[\mathcal{L}]$ (Theorem 8.2). For Reach$[\mathcal{LC}]$ restricted to complete protocols, given the complexity of the problem, such an upper-bound would be non-primitive recursive and thus would not be of any practical use.

2.4 Illustrative example

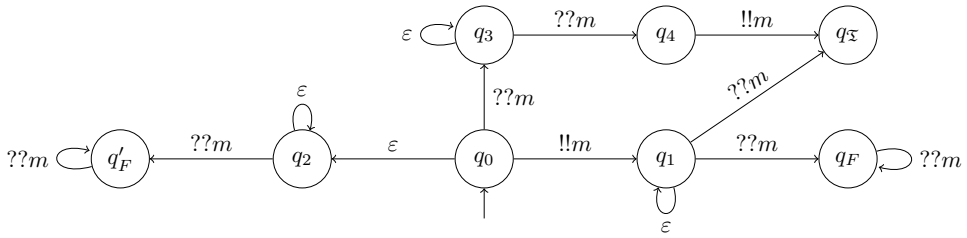


Figure 1 Example of a broadcast protocol.

To illustrate the notions of local strategies and clique executions, we provide an example of a broadcast protocol in Fig. 1. On this protocol no clique execution can reach state q_F: as soon as a process in q_0 sends message m, all the other processes in q_0 receive this message, and move to q_3, because of the clique topology. An example of a clique execution is: $(q_0, q_0, q_0, q_0) \to (q_1, q_3, q_3, q_3)$ (where we omit the labels over $\to$). However, there exists a local execution reaching q_F: $(q_0, q_0) \to (q_1, q_0) \to (q_F, q_1)$. This execution respects a local strategy since, from q_0 with empty past, the first process chooses the edge broadcasting m

with empty reception set and in the next step the second process, also with empty past, performs the same action, broadcasting the message m to the first process. On the other hand, no local strategy permits to reach q'_F. Indeed, intuitively, to reach q'_F, in state q_0 one process with empty past needs to go to q_1 and another one to q_2, which is forbidden by locality. Finally $(q_0, q_0, q_0) \to (q_1, q_0, q_3) \to (q_1, q_1, q_4) \to (q_\mathfrak{T}, q_\mathfrak{T}, q_\mathfrak{T})$ is a local execution that targets the set $\mathfrak{T} = \{q_\mathfrak{T}\}$.

3 Verification problems for local executions

We begin with studying the parameterized reachability and target problems under local executions, *i.e.* we seek for a local strategy ensuring either to reach a specific control state, or to reach a configuration in which all the control states belong to a given set.

3.1 Solving Reach[$\mathcal{L}$]

To obtain an NP-algorithm for Reach[$\mathcal{L}$], we prove that there exists a local strategy to reach a specific control state if and only if there is a local strategy which can be represented thanks to a finite tree of polynomial size; the idea behind such a tree being that the paths in the tree represent past histories and the edges outgoing a specific node represent the decisions of the local strategy. The NP-algorithm will then consist in guessing such finite tree of polynomial size and verifying if it satisfies some conditions needed to reach the specified control state.

Representing strategies with trees. We now define our tree representation of strategies called strategy patterns, which are standard labelled trees with labels on the edges. Intuitively a strategy pattern defines, for some of the paths in the associated protocol, the active action and receptions to perform.

A *strategy pattern* for a broadcast protocol $\mathcal{P} = (Q, q_0, \Sigma, \Delta)$ is a labelled tree $T = (N, n_0, E, \Delta, \mathsf{lab})$ with N a finite set of nodes, $n_0 \in N$ the root, $E \subseteq N \times N$ the edge relation and $\mathsf{lab} : E \to \Delta$ the edge-labelling function. Moreover T is such that if $e_1 \cdots e_\ell$ is a path in T, then $\mathsf{lab}(e_1) \cdots \mathsf{lab}(e_\ell) \in \mathsf{Path}(\mathcal{P})$, and for every node $n \in N$: there is at most one edge $e = (n, n') \in E$ such that $\mathsf{lab}(e)$ is an active action; and, for each message m, there is at most one edge $e = (n, n') \in E$ such that $\mathsf{lab}(e)$ is a reception of m.

Since all labels of edges outgoing a node share a common source state (due to the hypothesis on labelling of paths), the labelling function lab can be consistently extended to nodes by letting $\mathsf{lab}(n_0) = q_0$ and $\mathsf{lab}(n) = q$ for any $(n', n) \in E$ with $\mathsf{lab}((n', n)) = (q', a, q)$.

The strategy pattern represented in Fig. 2, for the broadcast protocol from Fig. 1, illustrates that strategy patterns somehow correspond to under-specified local strategies. For example, from node n_1 (labelled by q_1) no reception of message m is specified, and from node n_5 (labelled by q_4) no reception and no active action are specified.

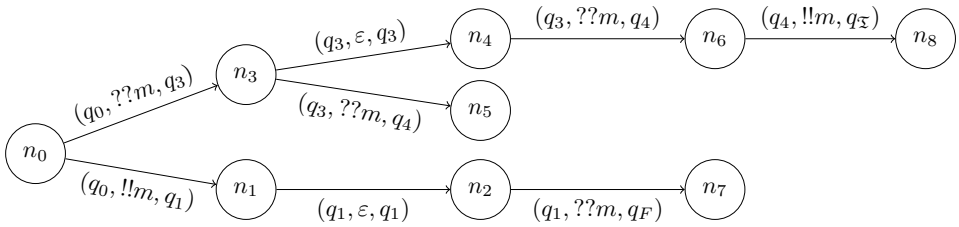


Figure 2 A strategy pattern for the broadcast protocol depicted Fig. 1.

More generally, given $\mathcal{P}$ a broadcast protocol, and T a strategy pattern for $\mathcal{P}$ with edge-labelling function lab, a local strategy $\sigma = (\sigma_a, \sigma_r)$ for $\mathcal{P}$ is said to *follow* T if for every path $e_1 \cdots e_\ell$ in T, the path $\rho = \mathsf{lab}(e_1) \cdots \mathsf{lab}(e_\ell)$ in $\mathcal{P}$ respects σ. Notice that any strategy pattern admits at least one local strategy that follows it.

Reasoning on strategy patterns. We now show that one can test directly on a strategy pattern whether the local strategies following it can yield an execution reaching a specific control state. An *admissible strategy pattern* for $\mathcal{P} = (Q, q_0, \Sigma, \Delta)$ is a pair $(T, \prec)$ where $T = (N, n_0, E, \Delta, \mathsf{lab})$ is a strategy pattern for $\mathcal{P}$ and $\prec \subseteq N \times N$ is a strict total order on the nodes of T such that:

(1) for all $(n, n') \in E$ we have $n \prec n'$;
(2) for all $e = (n, n') \in E$, if $\mathsf{lab}(e) = (\mathsf{lab}(n), ??m, \mathsf{lab}(n'))$ for some $m \in \Sigma$, then there exists $e_1 = (n_1, n_1')$ in E such that $n_1' \prec n'$ and $\mathsf{lab}(e_1) = (\mathsf{lab}(n_1), !!m, \mathsf{lab}(n_1'))$.

In words, (1) states that $\prec$ respects the natural order on the tree and (2) that every node corresponding to a reception of m should be preceded by a node corresponding to a broadcast of m.

The example of strategy pattern on Fig. 2 is admissible with the order $n_i \prec n_j$ if $i < j$, whereas for any order including $n_3 \prec n_1$ it is not admissible (a broadcast of m should precede n_3). In general, given a strategy pattern T and a strict total order $\prec$, checking whether $(T, \prec)$ is admissible can be done in polynomial time (in the size of the pattern).

In order to state the relation between admissible strategy patterns and local strategies, we define $\mathsf{lab}(T) = \{\mathsf{lab}(n) \mid n \in N\}$ as the set of control states labelling nodes of T and $\mathsf{Occur}(\theta) = \{\gamma_i[p] \mid i \in [0..\ell+1] \text{ and } p \in [1..nbproc(\theta)]\}$ as the set of states that appear along an execution $\theta = \gamma_0 \to \cdots \to \gamma_{\ell+1}$. The next proposition tells us that admissible strategy patterns are necessary and sufficient to represent the sets of states that can be reached under local strategies.

▶ **Proposition 3.** *For all $Q' \subseteq Q$, there exists an admissible strategy pattern $(T, \prec)$ such that $\mathsf{lab}(T) = Q'$ iff there exists a local strategy σ and an execution θ such that θ respects σ and $Q' = \mathsf{Occur}(\theta)$, furthermore σ follows T.*

Minimizing admissible strategy patterns. For $(T, \prec)$ an admissible strategy pattern, we denote by $\mathsf{last}(T, \prec)$ the maximal node w.r.t. $\prec$ and we say that $(T, \prec)$ is *q_F-admissible* if $\mathsf{lab}(\mathsf{last}(T, \prec)) = q_F$. We now show that there exist polynomial size witnesses of q_F-admissible strategy patterns. The idea is to keep only relevant edges that either lead to a node labelled by q_F or that permit a broadcast of a new message. Intuitively, a *minimal* strategy pattern guarantees that (1) there is a unique node labelled with q_F, (2) in every subtree there is either a node labelled by q_F or a broadcast of a new message (*i.e.* a broadcast of a message that has not been seen previously with respect to the order $\prec$), and (3) a path starting and ending in two different nodes labelled by the same state, cannot be compressed without losing a new broadcast or a path towards q_F (by compressing we mean replacing the first node on the path by the last one). These hypotheses allow us to seek only for q_F-admissible strategy patterns of polynomial size.

▶ **Proposition 4.** *If there exists a q_F-admissible strategy pattern for $\mathcal{P}$, then there is one of size at most $(2|\Sigma| + 1) \cdot (|Q| - 1)$ and of height at most $(|\Sigma| + 1) \cdot |Q|$.*

By Proposition 3, there exists an execution $\theta \in \Theta_{\mathcal{L}}$ such that $q_F \in \mathsf{Occur}(\theta)$ iff there exists a q_F-admissible strategy pattern and thanks to Proposition 4 it suffices to look only for q_F-admissible strategy patterns of size polynomial in the size of the broadcast protocol. A non-

deterministic polynomial time algorithm for Reach[$\mathcal{L}$] consists then in guessing a strategy pattern of polynomial size and an order and then verifying whether it is q_F-admissible.

▶ **Theorem 5.** *Reach[$\mathcal{L}$] is in* NP.

We can furthermore provide bounds on the minimal number of processes and on the memory needed to implement local strategies. Given a q_F-admissible strategy pattern one can define an execution following the pattern such that each reception edge of the pattern is taken exactly once and active actions may be taken multiple times but in a row. Such an execution needs at most one process per reception edge. Together with the bound on the size of the minimal strategy patterns (see Proposition 4), this yields a cutoff property on the minimal size of network to reach the final state. Moreover the past history of every process in this execution is bounded by the depth of the tree, hence we obtain an upper bound on the size of the memory needed by each process for Reach[$\mathcal{L}$].

▶ **Proposition 6.** *If there exists an execution $\theta \in \Theta_{\mathcal{L}}$ such that $q_F \in \mathsf{Occur}(\theta)$, then there exists an execution $\theta' \in \Theta_{\mathcal{L}}$ such that $q_F \in \mathsf{Occur}(\theta')$ and $nbproc(\theta') \leq (2|\Sigma|+1)\cdot(|Q|-1)$ and $|\pi_p(\theta')| \leq (|\Sigma|+1)\cdot|Q|$ for every $p \in [1..nbproc(\theta')]$.*

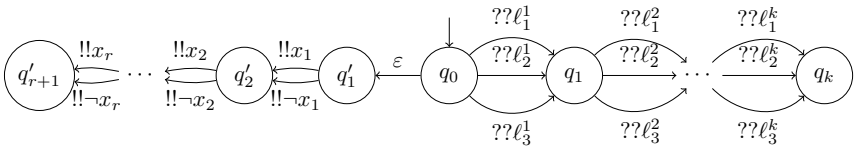


Figure 3 Encoding a 3-SAT formula into a broadcast protocol.

By reducing 3-SAT, one can furthermore show Reach[$\mathcal{L}$] to be NP-hard. Let $\phi = \bigwedge_{1\leq i\leq k}(\ell_1^i \vee \ell_2^i \vee \ell_3^i)$ be a 3-SAT formula such that $\ell_j^i \in \{x_1, \neg x_1, \ldots, x_r, \neg x_r\}$ for all $i \in [1..k]$ and $j \in \{1,2,3\}$. We build from ϕ the broadcast protocol $\mathcal{P}$ depicted at Fig. 3. Under this construction, ϕ is satisfiable iff there is an execution $\theta \in \Theta_{\mathcal{L}}$ such that $q_k \in \mathsf{Occur}(\theta)$. The local strategy hypothesis ensures that even if several processes broadcast a message corresponding to the same variable, all of them must take the same decision so that there cannot be any execution during which both x_i and $\neg x_i$ are broadcast. It is then clear that control state q_k can be reached if and only if each clause is satisfied by the set of broadcast messages. Together with Theorem 5, we obtain the precise complexity of Reach[$\mathcal{L}$].

▶ **Theorem 7.** *Reach[$\mathcal{L}$] is* NP*-complete.*

3.2 Solving Target[$\mathcal{L}$]

Admissible strategy patterns can also be used to obtain an NP-algorithm for Target[$\mathcal{L}$]. As we have seen, given an admissible strategy pattern, one can build an execution where the processes visit all the control states present in the pattern. When considering the target problem, one also needs to ensure that the processes can afterwards be directed to the target set. To guarantee this, it is possible to extend admissible strategy patterns with another order on the nodes which ensures that (a) from any node there exists a path leading to the target set and (b) whenever on this path a reception is performed, the corresponding message can be broadcast by a process that will only later on be able to reach the target.

We formalize now this idea. For $\mathfrak{T} \subseteq Q$ a set of states, a $\mathfrak{T}$*-coadmissible strategy pattern* for $\mathcal{P} = (Q, q_0, \Sigma, \Delta)$ is a pair $(T, \lhd)$ where $T = (N, n_0, E, \Delta, \mathsf{lab})$ is a strategy pattern for

$\mathcal{P}$ and $\lhd \subseteq N \times N$ is a strict total order on the nodes T such that for every node $n \in N$ with $\mathsf{lab}(n) \notin \mathfrak{T}$ there exists an edge $e = (n, n') \in E$ with $n \lhd n'$ and either:

- $\mathsf{lab}(e) = (\mathsf{lab}(n), \varepsilon, \mathsf{lab}(n'))$ or,
- $\mathsf{lab}(e) = (\mathsf{lab}(n), !!m, \mathsf{lab}(n'))$ or,
- $\mathsf{lab}(e) = (\mathsf{lab}(n), ??m, \mathsf{lab}(n'))$ and there exists an edge $e_1 = (n_1, n_1') \in E$ such that $n \lhd n_1$, $n \lhd n_1'$ and $\mathsf{lab}(e_1) = (q_1, !!m, q_1')$.

Intuitively the order $\lhd$ in a $\mathfrak{T}$-coadmissible strategy pattern corresponds to the order in which processes must move along the tree towards the target; the conditions express that any node with label not in $\mathfrak{T}$ has an outgoing edge that is feasible. In particular, a reception of m is only feasible before all edges carrying the corresponding broadcast are disabled.

A strategy pattern T equipped with two orderings $\prec$ and $\lhd$ is said to be $\mathfrak{T}$-*biadmissible* whenever $(T, \prec)$ is admissible and $(T, \lhd)$ is $\mathfrak{T}$-coadmissible. To illustrate the construction

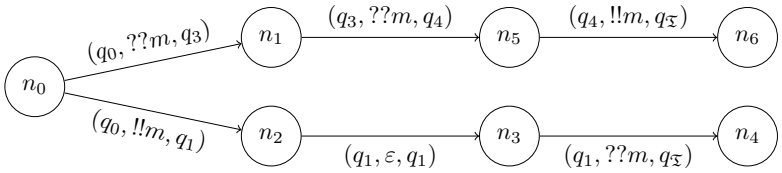


Figure 4 A $\mathfrak{T}$-coadmissible strategy pattern on the example protocol of Fig. 1.

of $\mathfrak{T}$-coadmissble patterns, we give in Fig. 4 an example pattern, that, equipped with the natural order $n_i \lhd n_j$ iff $i < j$, is $\mathfrak{T}$-coadmissible for $\mathfrak{T} = \{q_\mathfrak{T}\}$. Indeed all leaves are labelled with a target state, and the broadcast edge $n_5 \xrightarrow{(q_4, !!m, q_\mathfrak{T})} n_6$ allows all processes to take the corresponding reception edges. This $\mathfrak{T}$-coadmissible pattern is in particular obtained from the execution $(q_0, q_0, q_0) \to (q_1, q_3, q_0) \to (q_1, q_3, q_0) \to (q_\mathfrak{T}, q_4, q_1) \to (q_\mathfrak{T}, q_4, q_1) \to (q_\mathfrak{T}, q_\mathfrak{T}, q_\mathfrak{T})$. Notice that $\lhd$ is not an admissible order, because $n_1 \lhd n_2$, however there are admissible orders for this pattern, for example the order $n_0 \prec n_2 \prec n_3 \prec n_4 \prec n_1 \prec n_5 \prec n_6$.

As for Reach$[\mathcal{L}]$, one can show polynomial size witnesses of $\mathfrak{T}$-*biadmissible* strategy patterns exist, yielding an NP-algorithm for Target$[\mathcal{L}]$. Also, the size of minimal $\mathfrak{T}$-*biadmissible* strategy patterns gives here also a cutoff on the number of processes needed to satisfy the target condition, as well as an upper bound on the memory size.

▶ **Theorem 8.** *1.* Target$[\mathcal{L}]$ *is* NP*-complete.*

2. If there exists an execution $\theta \in \Theta_\mathcal{L}$ such that $\mathsf{End}(\theta) \subseteq \mathfrak{T}$, then there exists an execution $\theta' \in \Theta_\mathcal{L}$ such that $\mathsf{End}(\theta') \subseteq \mathfrak{T}$ and $nbproc(\theta') \leq 16|\Sigma| \cdot |Q| + 4|\Sigma| \cdot (|Q| - |\mathfrak{T}| + 1)$ and $|\pi_p(\theta')| \leq 4|\Sigma| \cdot |Q| + 2(|Q| - |\mathfrak{T}|) + 1$ for every $p \leq nbproc(\theta')$.

▶ Remark. The NP-hardness derives from the fact that the target problem is harder than the reachability problem. To reduce Reach$[\mathcal{L}]$ to Target$[\mathcal{L}]$, one can add the broadcast of a new message from q_F, and its reception from any state to q_F.

Another consequence of this simple reduction is that Target$[\mathcal{L}]$ in NP yields another proof that Reach$[\mathcal{L}]$ is in NP, yet the two proofs of NP-membership allowed us to give an incremental presentation, starting with admissible strategy patterns, and proceeding with co-admissible strategy patterns.

4 Verification problems for local clique executions

4.1 Undecidability of Reach[$\mathcal{LC}$] and Target[$\mathcal{LC}$]

Reach[$\mathcal{LC}$] and Target[$\mathcal{LC}$] happen to be undecidable and for the latter, even in the case of complete protocols. The proofs of these two results are based on a reduction from the halting problem of a two counter Minsky machine (a finite program equipped with two integer variables which can be incremented, decremented and tested to zero). The main idea consists in both cases in isolating some processes to simulate the behavior of the machine while the other processes encode the values of the counters.

Thanks to the clique semantics we can in fact isolate one process. This is achieved by setting the first transition to be the broadcast of a message *start* whose reception makes all the other process change their state. Hence, thanks to the clique semantics, there is only one process that sends the message *start*, such process, called the controller, will be in charge of simulating the transitions of the Minsky machine. The clique semantics is also used to correctly simulate the increment and decrement of counters. For instance to increment a counter, the controller asks whether a process simulating the counter can be moved from state 0 to state 1 and if it is possible, relying on the clique topology only one such process changes its state (the value of the counter is then the number of processes in state 1). In fact, all the processes will receive the request, but the first one answering it, will force the other processes to come back to their original state, ensuring that only one process will move from state 0 to 1.

The main difficulty is that broadcast protocols (even under the clique semantics) cannot test the absence of processes in a certain state (which would be needed to simulate a test to 0 of one of the counters). Here is how we overcome this issue for Target[$\mathcal{LC}$]: the controller, when simulating a zero-test, sends all the processes with value 1 into a sink error state and the target problem allows to check for the reachability of a configuration with no process in this error state (and thus to test whether the controller has 'cheated', *i.e.* has taken a zero-test transition whereas the value of the associated counter was not 0). We point out that in this case, restricting to local executions is not necessary, we get in fact as well that Target[$\mathcal{C}$] is undecidable.

For Reach[$\mathcal{LC}$], the reduction is more tricky since we cannot rely on a target set of states to check that zero-test were faithfully simulated. Here in fact we will use two controllers. Basically, before sending a *start* message, some processes will be able to go to a waiting state (thanks to an internal transition) from which they can become controller and in which they will not receive any messages (this is where the protocol needs to be incomplete). Then we will use the locality hypothesis to ensure that two different controllers will simulate exactly the same run of the Minsky machine twice and with exactly the same number of processes encoding the counters. Restricting to local strategies guarantees the two runs to be identical, and the correctness derives from the fact that if in the first simulation the controller 'cheats' while performing a zero-test (and sending as before some processes encoding a counter value into a sink state), then in the second simulation, the number of processes encoding the counters will be smaller (due to the processes blocked in the sink state), so that the simulation will fail (because there will not be enough processes to simulate faithfully the counter values).

▶ **Theorem 9.** Reach*[$\mathcal{LC}$] is undecidable and* Target*[$\mathcal{LC}$] restricted to complete protocol is undecidable.*

The undecidability proof for Reach[$\mathcal{LC}$] strongly relies on the protocol being incomplete.

Indeed, in the absence of specified receptions, the processes ignore broadcast messages and keep the same history, thus allowing to perform twice the same simulation of the run. In contrast, for complete protocols, all the processes are aware of all broadcast messages, therefore one cannot force the two runs to be identical. In fact, the reachability problem is decidable for complete protocols, as we shall see in the next section.

4.2 Decidability of Reach[$\mathcal{LC}$] for complete protocols

To prove the decidability of Reach[$\mathcal{LC}$] for complete protocols, we abstract the behavior of a protocol under local clique semantics by counting the possible number of different histories in each control state.

We identify two cases when the history of processes can differ (under local clique semantics): (1) When a process p performs a broadcast, its history is unique for ever (since all the other processes must receive the emitted message); (2) A set of processes sharing the same history can be split when some of them perform a sequence of internal actions and the others perform only a prefix of that sequence.

From a complete broadcast protocol $\mathcal{P} = (Q, q_0, \Sigma, \Delta)$ we build an abstract transition system $\mathcal{T}_{\mathcal{P}}^{\mathcal{LC}} = (\Lambda, \lambda_0, \Rightarrow)$ where configurations count the number of different histories in each control state. More precisely the set of abstract configurations is $\Lambda = \mathcal{M}(Q \times \{\mathbf{m}, \mathbf{s}\} \times \{!!_{ok}, !!_{no}\}) \times \{\varepsilon, !!\}$. Abstract configurations are thus pairs where the first element is a multiset and the second element is a flag in $\{\varepsilon, !!\}$. The latter indicates the type of the next actions to be simulated (sequence of internal actions or broadcast): it prevents to simulate consecutively two incoherent sequences of internal actions (with respect to the local strategy hypothesis). For the former, an element $(q, \mathbf{s}, !!_{ok})$ in the multiset represents a single process (flag $\mathbf{s}$) in state q with a unique history which is allowed to perform a broadcast (flag $!!_{ok}$). An element $(q, \mathbf{m}, !!_{no})$ represents many processes (flag $\mathbf{m}$) in state q, all sharing the same unique history and none of them is allowed to perform a broadcast (flag $!!_{no}$). The initial abstract configuration λ_0 is then $(\{\{(q_0, \mathbf{m}, !!_{ok})\}\}, \varepsilon)$. In the sequel we will write $\mathbf{HM}$ for the set $\mathcal{M}(Q \times \{\mathbf{m}, \mathbf{s}\} \times \{!!_{ok}, !!_{no}\})$ of history multisets, so that $\Lambda = \mathbf{HM} \times \{\varepsilon, !!\}$, and typical elements of $\mathbf{HM}$ are denoted $\mathbb{M}$, $\mathbb{M}'$, etc.

In order to provide the definition of the abstract transition relation $\Rightarrow$, we need to introduce new notions, and notations. An ε-path ρ in $\mathcal{P}$ from q to q' is either the empty path (and in that case $q = q'$) or it is a non-empty finite path $\delta_0 \cdots \delta_n$ that starts in q, ends in q' and such that all the δ_i's are internal transitions.

An ε-path ρ in $\mathcal{P}$ is said to be a *prefix* of an ε-path ρ' if $\rho \neq \rho'$ and either ρ is the empty path or $\rho = \delta_0 \cdots \delta_n$ and $\rho' = \delta_0 \cdots \delta_n \delta_{n+1} \ldots \delta_{n+m}$ for some $m > 0$. Since we will handle multisets, let us give some convenient notations. Given E a set, and $\mathbb{M}$ a multiset over E, we write $\mathbb{M}(e)$ for the number of occurrences of element $e \in E$ in $\mathbb{M}$. Moreover, $\mathit{card}(\mathbb{M})$ stands for the cardinality of $\mathbb{M}$: $\mathit{card}(\mathbb{M}) = \sum_{e \in E} \mathbb{M}(e)$. Last, we will write $\oplus$ for the addition on multisets: $\mathbb{M} \oplus \mathbb{M}'$ is such that for all $e \in E$, $(\mathbb{M} \oplus \mathbb{M}')(e) = \mathbb{M}(e) + \mathbb{M}'(e)$.

The abstract transition relation $\Rightarrow \in \Lambda \times \Lambda$ is composed of two transitions relations: one simulates the broadcast of messages and the other one sequences of internal transitions. This will guarantee an alternation between abstract configurations flagged with ε and the ones flagged with $!!$. Let us first define $\Rightarrow_{!!} \subseteq (\mathbf{HM} \times \{!!\}) \times (\mathbf{HM} \times \{\varepsilon\})$ which simulates a broadcast. We have $(\mathbb{M}, !!) \Rightarrow_{!!} (\mathbb{M}', \varepsilon)$ iff there exists $(q_1, !!m, q_2) \in \Delta$ and $\mathit{fl}_1 \in \{\mathbf{s}, \mathbf{m}\}$ such that

1. $\mathbb{M}(q_1, \mathit{fl}_1, !!_{ok}) > 0$

2. there exists a family of functions G indexed by $(q, \mathit{fl}, b) \in Q \times \{\mathbf{m}, \mathbf{s}\} \times \{!!_{ok}, !!_{no}\}$, such that $G_{(q,\mathit{fl},b)} : [1..\mathbb{M}(q, \mathit{fl}, b)] \to \mathbf{HM}$, and:

$$\mathbb{M}' = \{\{q_2, \mathbf{s}, !!_{ok}\}\} \oplus \bigoplus_{\{(q,\mathit{fl},b) | \mathbb{M}(q,\mathit{fl},b) \neq 0\}} \bigoplus_{i \in [1..\mathbb{M}(q,\mathit{fl},b)]} G_{(q,\mathit{fl},b)}(i)$$

and such that for each (q, fl, b) verifying $\mathbb{M}(q, \mathit{fl}, b) \neq 0$, for all $i \in [1..\mathbb{M}(q, \mathit{fl}, b)]$, the following conditions are satisfied:

- **a.** if $\mathit{fl}_1 = \mathbf{s}$, $\mathit{card}(G_{(q_1,\mathit{fl}_1,!!_{ok})}(1)) = 0$ and if $\mathit{fl}_1 = \mathbf{m}$, then there exists $q' \in Q$ such that $G_{(q_1,\mathit{fl}_1,!!_{ok})}(1) = \{\{(q', \mathit{fl}_1, !!_{ok})\}\}$ and such that $(q, ??m, q') \in \Delta$;
- **b.** if $(q, \mathit{fl}, b) \neq (q_1, \mathit{fl}_1, !!_{ok})$ or $i \neq 1$, then there exists $q' \in Q$ such that $G_{(q,\mathit{fl},b)}(i) = \{\{(q', \mathit{fl}, !!_{ok})\}\}$ and such that $(q, ??m, q') \in \Delta$.

Intuitively to provide the broadcast, we need to find a process which is 'allowed' to perform a broadcast and which is hence associated with an element $(q_1, \mathit{fl}_1, !!_{ok})$ in $\mathbb{M}$. The transition $(q_1, !!m, q_2)$ tells us which broadcast is simulated. Then the functions $G_{(q,\mathit{fl},b)}$ associate with each element of the multiset $\mathbb{M}$ of the form (q, fl, b) a single element which can be reached thanks to a reception of the message m. Of course this might not hold for an element of the shape $(q_1, \mathbf{s}, !!_{ok})$ if it is the one chosen to do the broadcast since it represents a single process, and hence this element moves to q_2. Note however that if $\mathit{fl}_1 = \mathbf{m}$, then $(q_1, \mathbf{m}, !!_{ok})$ represents many processes, hence the one which performs the broadcast is isolated, but the many other ones have to be treated for reception of the message. Note also that we use here the fact that since an element $(q, \mathbf{m}, b)$ represents many processes with the same history, all these processes will behave the same way on reception of the message m.

We now define $\Rightarrow_\varepsilon \subseteq (\mathbf{HM} \times \{\varepsilon\}) \times (\mathbf{HM} \times \{!!\})$ which simulates the firing of sequences of ε-transitions. We have $(\mathbb{M}, \varepsilon) \Rightarrow_\varepsilon (\mathbb{M}', !!)$ iff there exists a family of functions F indexed by $(q, \mathit{fl}, b) \in Q \times \{\mathbf{m}, \mathbf{s}\} \times \{!!_{ok}, !!_{no}\}$, such that $F_{(q,\mathit{fl},b)} : [1..\mathbb{M}(q, \mathit{fl}, b)] \to \mathbf{HM}$, and

$$\mathbb{M}' = \bigoplus_{\{(q,\mathit{fl},b) | \mathbb{M}(q,\mathit{fl},b) \neq 0\}} \bigoplus_{i \in [1..\mathbb{M}(q,\mathit{fl},b)]} F_{(q,\mathit{fl},b)}(i)$$

and such that for each (q, fl, b) verifying $\mathbb{M}(q, \mathit{fl}, b) \neq 0$, for all $i \in [1..\mathbb{M}(q, \mathit{fl}, b)]$, we have:

1. $\mathit{card}(F_{(q,\mathit{fl},b)}(i)) \geq 1$ and if $\mathit{fl} = \mathbf{s}$, $\mathit{card}(F_{(q,\mathit{fl},b)}(i)) = 1$;
2. If $F_{(q,\mathit{fl},b)}(i)(q', \mathit{fl}', b') \neq 0$, then $\mathit{fl}' = \mathit{fl}$;
3. There exists a pair $(q_{!!}, \mathit{fl}_{!!}) \in Q \times \{\mathbf{m}, \mathbf{s}\}$ such that:
- $F_{(q,\mathit{fl},b)}(i)(q_{!!}, \mathit{fl}_{!!}, !!_{ok}) = 1$
- for all $(q', \mathit{fl}') \neq (q_{!!}, \mathit{fl}_{!!})$ $F_{(q,\mathit{fl},b)}(i)(q', \mathit{fl}', !!_{ok}) = 0$;
- There exists a ε-path $\rho_{!!}$ from q to $q_{!!}$.

4. For all (q', fl') such that $F_{(q,\mathit{fl},b)}(i)(q', \mathit{fl}', !!_{no}) = k > 0$, there exists k different ε-paths (strict) prefix of $\rho_{!!}$ from q to q'.

Intuitively the functions $F_{(q,\mathit{fl},b)}$ associate with each element (q, fl, b) of the multiset $\mathbb{M}$ a set of elements that can be reached via internal transitions. We recall that each such element represents a set (or a singleton if $\mathit{fl} = \mathbf{s}$) of processes sharing the same history. Condition 1. states that if there are multiple processes ($\mathit{fl} = \mathbf{m}$) then they can be matched to more states in the protocol, but if it is single ($\mathit{fl} = \mathbf{s}$) it should be matched by an unique state. Condition 2. expresses that if an element in $\mathbb{M}$ represents many processes, then all its images represent as well many processes. Conditions 3. and 4. deal with the locality assumption. Precisely, condition 3. states that among all the elements of $\mathbb{M}'$ associated with an element of $\mathbb{M}$, one

and only one should be at the end of a ε-path, and only one process associated with this element will be allowed to perform a broadcast. This justifies the use of the flag $!!_{ok}$. Last, condition 4. concerns all the other elements associated to this element of $\mathbb{M}$: their flag is set to $!!_{no}$ (they cannot perform a broadcast, because the local strategy will force them to take an internal transition), and their state should be on the previously mentioned ε-path.

As announced, we define the abstract transitive relation by $\Rightarrow = \Rightarrow_\varepsilon \cup \Rightarrow_{!!}$. Note that by definition we have a strict alternation of transitions of the type $\Rightarrow_\varepsilon$ and of the type $\Rightarrow_{!!}$. An *abstract local clique execution* of $\mathcal{P}$ is then a finite sequence of consecutive transitions in $\mathcal{T}_{\mathcal{P}}^{\mathcal{LC}}$ of the shape $\xi = \lambda_0 \Rightarrow \lambda_1 \cdots \Rightarrow \lambda_{\ell+1}$. As for concrete executions, if $\lambda_{\ell+1} = (\mathbb{M}_{\ell+1}, t_{\ell+1})$ we denote by $\mathsf{End}(\xi) = \{q \mid \exists fl \in \{\mathbf{m}, \mathbf{s}\}. \exists b \in \{!!_{ok}, !!_{no}\}. \mathbb{M}_{\ell+1}(q, fl, b) > 0\}$ the set of states that appear in the end configuration of ξ.

As an example, a possible abstract execution of the broadcast protocol from Fig. 1 is: $(\{\{(q_0, \mathbf{m}, !!_{ok})\}\}, \varepsilon) \Rightarrow (\{\{(q_0, \mathbf{m}, !!_{no}), (q_2, \mathbf{m}, !!_{no}), (q_2, \mathbf{m}, !!_{ok})\}\}, !!)$. This single-step execution represents that among the processes in q_0, some processes will take an internal action to q_2 and loop there with another internal action (they are represented by the element $(q_2, \mathbf{m}, !!_{ok})$), others will only move to q_2 taking a single internal action (they are represented by $(q_2, \mathbf{m}, !!_{no})$), and finally some processes will stay in q_0 (they are represented by $(q_0, \mathbf{m}, !!_{no})$); note that these processes are not able to perform a broadcast, because due to the local strategy hypothesis, they committed to firing the internal action leading to q_2.

Another example of an abstract execution is: $(\{\{(q_0, \mathbf{m}, !!_{ok})\}\}, \varepsilon) \Rightarrow (\{\{(q_0, \mathbf{m}, !!_{ok})\}\}, !!)$ $\Rightarrow (\{\{(q_1, \mathbf{s}, !!_{ok}), (q_3, \mathbf{m}, !!_{ok})\}\}, \varepsilon) \Rightarrow (\{\{(q_1, \mathbf{s}, !!_{ok}), (q_3, \mathbf{m}, !!_{no}), (q_3, \mathbf{m}, !!_{ok})\}\}, \varepsilon)$. Here in the first step, no process performs internal actions, in the second step one of the processes in q_0 broadcasts m, moves to q_1 and we know that no other process will ever share the same history, it is hence represented by $(q_1, \mathbf{s}, !!_{ok})$; then all the other processes with the same history represented by $(q_0, \mathbf{m}, !!_{ok})$ must receive m and move to q_3, they are hence represented by $(q_3, \mathbf{m}, !!_{ok})$. The last step represents that some processes perform the internal action loop on q_3.

The definition of the abstract transition system $\mathcal{T}_{\mathcal{P}}^{\mathcal{LC}}$ ensures a correspondence between abstract local clique executions and local clique executions in $\mathcal{P}$. Formally:

▶ **Lemma 10.** *Let $q_F \in Q$. There exists an abstract local clique execution ξ of $\mathcal{P}$ such that $q_F \in \mathsf{End}(\xi)$ iff there exists a local clique execution $\theta \in \Theta_{\mathcal{LC}}$ such that $q_F \in \mathsf{End}(\theta)$.*

Given the abstract transition system $\mathcal{T}_{\mathcal{P}}^{\mathcal{LC}}$, in order to show that Reach$[\mathcal{LC}]$ is decidable, we then rely on the theory of well-structured transition systems [1, 13]. Indeed, the natural order on abstract configurations is a well-quasi-order compatible with the transition relation $\Rightarrow$ of $\mathcal{T}_{\mathcal{P}}^{\mathcal{LC}}$ (bigger abstract configurations simulate smaller ones) and one can compute predecessors of upward-closed sets of configurations. This allows us to conclude that, in $\mathcal{T}_{\mathcal{P}}^{\mathcal{LC}}$, the set of all predecessors of a configuration where q_F appears is effectively computable, so that we can decide whether q_F is reachable in $\mathcal{T}_{\mathcal{P}}^{\mathcal{LC}}$, hence, thanks to the previous lemma, in $\mathcal{P}$.

We also show that Reach$[\mathcal{LC}]$ is non-primitive recursive thanks to a PTIME reduction from Reach$[\mathcal{C}]$ (which is Ackermann-complete [16]) to Reach$[\mathcal{LC}]$. We exploit the fact that the only difference between the semantics $\mathcal{C}$ and $\mathcal{LC}$ is that in the latter, processes with the same history take the same decision. We simulate this in $\mathcal{C}$ with a gadget which assigns a different history to each individual process at the beginning of the protocol making hence the reachability problem for $\mathcal{C}$ equivalent to the one with $\mathcal{LC}$ semantics.

▶ **Theorem 11.** Reach*$[\mathcal{LC}]$ restricted to complete protocols is decidable and* NPR.

5 Conclusion

We considered reconfigurable broadcast networks under local strategies that rule out executions in which processes with identical local history behave differently. Under this natural assumption for distributed protocols, the reachability and target problems are NP-complete. Moreover, we gave polynomial bounds on the cutoff and on the memory needed by strategies. When the communication topology is a clique, both problems become undecidable. Decidability is recovered for reachability if we further assume that protocols are complete.

To the best of our knowledge, this is the first attempt to take into account the local viewpoint of the processes in parameterized distributed systems. It could be interesting to study how the method we propose in this work can be adapted to parameterized networks equipped with other means of communication (such as rendez-vous [14] or shared memory [12]). In the future we also plan to deal with properties beyond simple reachability objectives, as for example linear or branching time properties.

References

1 Parosh A. Abdulla, Karlis Cerans, Bengt Jonsson, and Yih-Kuen Tsay. Algorithmic analysis of programs with well quasi-ordered domains. *Inf. Comput.*, 160(1-2):109–127, 2000.

2 Benjamin Aminof, Swen Jacobs, Ayrat Khalimov, and Sasha Rubin. Parameterized model checking of token-passing systems. In *Proc. of VMCAI'14*, volume 8318 of *LNCS*, pages 262–281, 2014.

3 Nathalie Bertrand, Paulin Fournier, and Arnaud Sangnier. Distributed local strategies in broadcast networks. Research report, HAL, CNRS, France, July 2015. https://hal.inria.fr/hal-01170796.

4 Benedikt Bollig. Logic for communicating automata with parameterized topology. In *Proc. of CSL-LICS'14*, page 18. ACM, 2014.

5 Benedikt Bollig, Paul Gastin, and Jana Schubert. Parameterized verification of communicating automata under context bounds. In *Proc. of RP'14*, volume 8762 of *LNCS*, pages 45–57, 2014.

6 Edmund M. Clarke, Muralidhar Talupur, Tayssir Touili, and Helmut Veith. Verification by network decomposition. In *Proc. of CONCUR'04*, volume 3170 of *LNCS*, pages 276–291, 2004.

7 Giorgio Delzanno, Arnaud Sangnier, Riccardo Traverso, and Gianluigi Zavattaro. On the complexity of parameterized reachability in reconfigurable broadcast networks. In *Proc. of FSTTCS'12*, volume 18 of *LIPIcs*, pages 289–300. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

8 Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. Parameterized verification of ad hoc networks. In *Proc. of CONCUR'10*, volume 6269 of *LNCS*, pages 313–327. Springer, 2010.

9 Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. On the power of cliques in the parameterized verification of ad hoc networks. In *Proc. of FoSSaCS'11*, volume 6604 of *LNCS*, pages 441–455. Springer, 2011.

10 Javier Esparza. Keeping a crowd safe: On the complexity of parameterized verification (invited talk). In *Proc. of STACS'14*, volume 25 of *LIPIcs*, pages 1–10. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014.

11 Javier Esparza, Alain Finkel, and Richard Mayr. On the verification of broadcast protocols. In *Proc. of LICS'99*, pages 352–359. IEEE Computer Society, 1999.

12 Javier Esparza, Pierre Ganty, and Rupak Majumdar. Parameterized verification of asynchronous shared-memory systems. In *Proc. of CAV'13*, volume 8044 of *LNCS*, pages 124–140, 2013.

13 Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001.

14 Steven M. German and A. Prasad Sistla. Reasoning about systems with many processes. *J. ACM*, 39(3):675–735, 1992.

15 Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *Proc. of FOCS'90*, pages 746–757. IEEE Computer Society, 1990.

16 Sylvain Schmitz and Philippe Schnoebelen. The power of well-structured systems. In *Proc. of CONCUR'13*, volume 8052 of *LNCS*, pages 5–24. Springer, 2013.

Reachability in Networks of Register Protocols under Stochastic Schedulers*†

Patricia Bouyer[1], Nicolas Markey[1], Mickael Randour[‡2], Arnaud Sangnier[3], and Daniel Stan[1]

1 **LSV – CNRS, ENS Cachan & University Paris-Saclay – France**
2 **Computer Science Department – Université Libre de Bruxelles – Belgium**
3 **IRIF – University Paris Diderot & CNRS – France**

Abstract

We study the almost-sure reachability problem in a distributed system obtained as the asynchronous composition of N copies (called processes) of the same automaton (called protocol), that can communicate via a shared register with finite domain. The automaton has two types of transitions: write-transitions update the value of the register, while read-transitions move to a new state depending on the content of the register. Non-determinism is resolved by a stochastic scheduler. Given a protocol, we focus on almost-sure reachability of a target state by one of the processes. The answer to this problem naturally depends on the number N of processes. However, we prove that our setting has a cut-off property: the answer to the almost-sure reachability problem is constant when N is large enough; we then develop an EXPSPACE algorithm deciding whether this constant answer is positive or negative.

1998 ACM Subject Classification F.1.1 Models of Computation, F.3.1 Specifying and Verifying and Reasoning about Programs, C.2.2 Network Protocols

Keywords and phrases Networks of Processes, Parametrized Systems, Stochastic Scheduler, Almost-sure Reachability, Cut-Off Property

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

Verification of systems with many identical processes. It is a classical pattern in distributed systems to have a large number of identical components running concurrently (a.k.a. networks of processes). In order to verify the correctness of such systems, a naive option consists in fixing an upper bound on the number of processes, and applying classical verification techniques on the resulting system. This has several drawbacks, and in particular it gives no information whatsoever about larger systems. Another option is to use parameterized-verification techniques, taking as a parameter the number of copies of the protocol in the system being considered. In such a setting, the natural question is to find and characterize, if it exists, an infinite set of parameter values for which the system is correct. Not only the latter approach is more general, but it might also turn out to be easier and more efficient, since it involves orthogonal techniques.

* A full version of the paper is available on Arxiv [7]
† This work has been partly supported by ERC Starting grant EQualIS (FP7-308087) and by European FET project Cassting (FP7-601148) by the ANR research program PACS (ANR-14-CE28-0002).
‡ F.R.S.-FNRS Postdoctoral Researcher.

Different means of communication lead to different models. A seminal paper on parameterized verification of such distributed systems is the work of German and Sistla [17]. In this work, the authors consider networks of processes all following the same finite-state automaton; the communication between processes is performed thanks to *rendez-vous* communication. Various related settings have been proposed and studied since then, which mainly differ by the way the processes communicate. Among those, let us mention broadcast communication [15, 10], token-passing [8, 2], message passing [6], shared register with ring topologies [1], or shared memory [16]. In his nice survey on such parameterized models [14], Esparza shows that minor changes in the setting, such as the presence of a controller in the system, might drastically change the complexity of the verification problems. The relative expressiveness of some of those models has been studied recently in [3], yielding several reductions of the verification problems for some of those classes of models.

Asynchronous shared-memory systems. We consider a communication model where the processes asynchronously access a shared register, and where read and write operations on this register are performed non-atomically. A similar model has been proposed by Hague in [18], where the behavior of processes is defined by a pushdown automaton. The complexity of some reachability and liveness problems for shared-memory models have then been established in [16] and [11], respectively. These works consider networks in which a specific process, called the leader, runs a different program, and address the problem whether, for some number of processes, the leader can satisfy a given reachability or liveness property. In the case where there is no leader, and where processes are finite-state, the parameterized control-state reachability problem (asking whether one of the processes can reach a given control state) can be solved in polynomial time, by adapting the approach of [9] for lossy broadcast protocols.

Fairness and cut-off properties. In this work, we further insert fairness assumptions in the model of parameterized networks with asynchronous shared memory, and consider reachability problems in this setting. There are different ways to include fairness in parameterized models. One approach is to enforce fairness expressed as a temporal-logic properties on the executions (e.g., any action that is available infinitely often must be performed infinitely often); this is the option chosen for parameterized networks with rendez-vous [17] and for systems with disjunctive guards (where processes can query the states of other processes) in [4]. We follow another choice, by equipping our networks with a stochastic scheduler that, at each step of the execution, assigns the same probability to the available actions of all the processes. From a high-level perspective, both forms of fairness are similar. However, expressing fairness via temporal logic allows for very regular patterns (e.g., round-robin execution of the processes), whereas the stochastic approach leads to consider all possible interleavings with probability 1. Under this stochastic scheduler assumption, we focus on almost-sure reachability of a given control state by any of the processes of the system. More specifically, as in [4], we are interested in determining the existence of a *cut-off*, i.e., an integer k such that networks with more than k processes almost-surely reach the target state. Deciding the existence and computing such cut-offs is important for at least two aspects: first, it ensures that the system is correct for arbitrarily large networks; second, if we are able to derive a bound on the cut-off, then using classical verification techniques we can find the exact value of the cut-off and exactly characterize the sizes of the networks for which the behavior is correct.

Our contributions. We prove that for finite-state asynchronous shared-memory protocols with a stochastic scheduler, and for almost-sure reachability of some control state by some process of the network, there always exists a positive or negative cut-off; positive cut-offs are those above which the target state is reached with probability 1, while negative cut-offs are those above which the target state is reached with probability strictly less than 1. Notice

that both cut-offs are not complement of one another, so that our result is not trivial.

We then prove that the "sign" (positive or negative) of a cut-off can be decided in EXPSPACE, and that this problem is PSPACE-hard. Finally, we provide lower and upper bounds on the values of the cut-offs, exhibiting in particular protocols with exponential (negative) cut-off. Notice how these results contrast with classical results in related areas: in the absence of fairness, reachability can be decided in polynomial time, and in most settings, when cut-offs exist, they generally have polynomial size [4, 13, 12].

2 Presentation of the model and of the considered problem

2.1 Preliminaries.

Let S be a finite set. A multiset over S is a mapping $\mu\colon S \to \mathbb{N}$. The cardinality of a multiset μ is $|\mu| = \sum_{s\in S} \mu(s)$. The support $\overline{\mu}$ of μ is the subset $\nu \subseteq S$ s.t. for all $s \in S$, it holds $s \in \nu$ if, and only if, $\mu(s) > 0$. For $k \in \mathbb{N}$, we write $\mathbb{N}_k^S$ for the set of multisets of cardinality k over S, and $\mathbb{N}^S$ for the set of all multisets over S. For any $s \in S$ and $k \in \mathbb{N}$, we write s^k for the multiset where $s^k(s) = k$ and $s^k(s') = 0$ for all $s' \neq s$. We may write s instead of s^1 when no ambiguity may arise. A multiset μ is included in a multiset μ', written $\mu \sqsubseteq \mu'$, if $\mu(s) \leq \mu'(s)$ for all $s \in S'$. Given two multisets μ and μ', their union $\mu \oplus \mu'$ is still a multiset s.t. $(\mu \oplus \mu')(s) = \mu(s) + \mu'(s)$ for all $s \in S$. Assuming $\mu \sqsubseteq \mu'$, the difference $\mu' \ominus \mu$ is still a multiset s.t. $(\mu' \ominus \mu)(s) = \mu'(s) - \mu(s)$.

A quasi-order $\langle A, \preceq\rangle$ is a *well quasi-order* (wqo for short) if for every infinite sequence of elements $a_1, a_2, \ldots$ in A, there exist two indices $i < j$ such that $a_i \preceq a_j$. For instance, for $n > 0$, $\langle \mathbb{N}^n, \leq\rangle$ (with lexicographic order) is a wqo. Given a set A with an ordering $\preceq$ and a subset $B \subseteq A$, the set B is said to be *upward closed* in A if for all $a_1 \in B$ and $a_2 \in A$, in case $a_1 \preceq a_2$, then $a_2 \in B$. The *upward-closure* of a set B (for the ordering $\preceq$), denoted by $\uparrow_{\preceq}(B)$ (or sometimes $\uparrow(B)$ when the ordering is clear from the context), is the set $\{a \in A \mid \exists b \in B \text{ s.t. } b \preceq a\}$. If $\langle A, \preceq\rangle$ is a wqo and B is an upward closed set in A, there exists a finite set of minimal elements $\{b_1, \ldots, b_k\}$ such that $B = \uparrow\{b_1, \ldots, b_k\}$.

2.2 Register protocols and associated distributed system.

We focus on systems that are defined as the (asynchronous) product of several copies of the same protocol. Each copy communicates with the others through a single register that can store values from a finite alphabet.

▶ **Definition 1.** A *register protocol* is given by $\mathcal{P} = \langle Q, D, q_0, T\rangle$

- Q is a finite set of control locations;
- D is a finite alphabet of data for the shared register;
- $q_0 \in Q$ is an initial location;
- $T \subseteq Q \times \{R, W\} \times D \times Q$ is the set of transitions of the protocol. Here R means *read* the content of the shared register, while W means *write* in the register.

In order to avoid deadlocks, it is required that each location has at least one outgoing transition. We also require that whenever some R-transition (q, R, d, q') appears in T, then for all $d \in D$, there exists at least one $q_d \in Q$ such that $(q, R, d, q_d) \in T$. The size of the protocol $\mathcal{P}$ is given by $|Q| + |T|$.

▶ **Example 1.a.** *Figure 1 displays a small register protocol with four locations, over an alphabet of data $D = \{0, 1, 2\}$. In this figure (and in the sequel), omitted R-transitions (e.g., transitions $R(1)$ and $R(2)$ from q_0) are assumed to be self-loops. When the register*

contains 0*, this protocol may move from initial location* q_0 *to location* q_1*. From there it can write* 1 *in the register, and then move to* q_2*. From* q_2*, as long as the register contains* 1*, the process can either stay in* q_2 *(with the omitted self-loop* $R(1)$*), or write* 2 *in the register and jump back to* q_1*. It is easily seen that if this process executes alone, it cannot reach state* q_f*.*

We now present the semantics of distributed systems associated with our register protocols. We consider the *asynchronous* composition of several copies of the protocol (the number of copies is not fixed a priori and can be seen as a parameter). We are interested in the behavior of such a composition under a fair scheduler. Such distributed systems involve two sources of non-determinism: first, register protocols may be non-deterministic; second, in any configuration, all protocols have at least one available transition, and non-determinism arises from the asynchronous semantics. In the semantics associated with a register protocol, non-determinism will be solved by a randomized scheduler, whose role is to select at each step which process will perform a transition, and which transition it will perform among the available ones. Because we will consider qualitative objectives (almost-sure reachability), the exact probability distributions will not really matter, and we will pick the uniform one (arbitrary choice). Note that we assume non-atomic read/write operations on the register, as in [18, 16, 11]. More precisely, when one process performs a transition, then all the processes that are in the same state are allowed to also perform the same transition just after, in fact write are always possible, and if a process performs a read of a specific value, since this read does not alter the value of the register, all processes in the same state can perform the same read (until one process performs a write). We will see later that dropping this hypothesis has a consequence on our results. We now give the formal definition of such a system.

The configurations of the distributed system built on register protocol $\mathcal{P} = \langle Q, D, q_0, T\rangle$ belong to the set $\Gamma = \mathbb{N}^Q \times D$. The first component of a configuration is a multiset characterizing the number of processes in each state of Q, whereas the second component provides the content of the register. For a configuration $\gamma = \langle \mu, d\rangle$, we denote by $st(\gamma)$ the multiset μ in $\mathbb{N}^Q$ and by $data(\gamma)$ the data d in D. We overload the operators defined over multisets; in particular, for a multiset δ over Q, we write $\gamma \oplus \delta$ for the configuration $\langle \mu \oplus \delta, d\rangle$. Similarly, we write $\overline{\gamma}$ for the support of $st(\gamma)$.

A configuration $\gamma' = \langle \mu', d'\rangle$ is a *successor* of a configuration $\gamma = \langle \mu, d\rangle$ if, and only if, there is a transition $(q, \mathsf{op}, d'', q') \in T$ such that $\mu(q) > 0$, $\mu' = \mu \ominus q \oplus q'$ and either $\mathsf{op} = R$ and $d = d' = d''$, or $\mathsf{op} = W$ and $d' = d''$. In that case, we write $\gamma \to \gamma'$. Note that since $\mu(q) > 0$ and $\mu' = \mu \ominus q \oplus q'$, we have necessarily $|\mu| = |\mu'|$. In our system, we assume that there is no creation or deletion of processes during an execution, hence the size of configurations (i.e., $|st(\gamma)|$) remains constant along transitions. We write Γ_k for the set of configurations of size k. For any configuration $\gamma \in \Gamma_k$, we denote by $\mathrm{Post}(\gamma) \subseteq \Gamma_k$ the set of successors of γ, and point out that such a set is finite and non-empty.

Now, the *distributed system* $\mathcal{S}_\mathcal{P}$ associated with a register protocol $\mathcal{P}$ is a discrete-time Markov chain $\langle \Gamma, Pr\rangle$ where $Pr\colon \Gamma \times \Gamma \to [0,1]$ is the transition probability matrix defined as follows: for all γ and $\gamma' \in \Gamma$, we have $Pr(\gamma, \gamma') = \frac{1}{|\mathrm{Post}(\gamma)|}$ if $\gamma \to \gamma'$, and $Pr(\gamma, \gamma') = 0$ otherwise. Note that Pr is well defined: by the restriction imposed on the transition

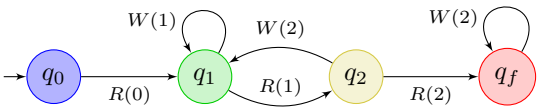


Figure 1 Example of a register protocol with $D = \{0, 1, 2\}$.

relation T of the protocol, we have $0 < |\mathrm{Post}(\gamma)| < \infty$ for all configuration γ, and hence we also get $\Sigma_{\gamma' \in \Gamma} Pr(\gamma, \gamma') = 1$. For a fixed integer k, we define the distributed system of size k associated with $\mathcal{P}$ as the finite-state discrete-time Markov chain $\mathcal{S}^k_{\mathcal{P}} = \langle \Gamma_k, \Pr_k \rangle$, where $\Pr_k$ is the restriction of Pr to $\Gamma_k \times \Gamma_k$.

We are interested in analyzing the behavior of the distributed system for a large number of participants. More precisely, we are interested in determining whether almost-sure reachability of a specific control state holds when the number of processes involved is large. We are therefore seeking a *cut-off* property, which we formalize in the following.

A finite path in the system $\mathcal{S}_{\mathcal{P}}$ is a finite sequence of configurations $\gamma_0 \to \gamma_1 \ldots \to \gamma_k$. In such a case, we say that the path starts in γ_0 and ends in γ_k. We furthermore write $\gamma \to^* \gamma'$ if, and only if, there exists a path that starts in γ and ends in γ'. Given a location q_f, we denote by $[\![\Diamond q_f]\!]$ the set of paths of the form $\gamma_0 \to \gamma_1 \ldots \to \gamma_k$ for which there is $i \in [0; k]$ such that $st(\gamma_i)(q_f) > 0$. Given a configuration γ, we denote by $\mathbb{P}(\gamma, [\![\Diamond q_f]\!])$ the probability that some paths starting in γ belong to $[\![\Diamond q_f]\!]$ in $\mathcal{S}_{\mathcal{P}}$. This probability is well-defined since the set of such paths is measurable (see e.g., [5]). Given a register protocol $\mathcal{P} = \langle Q, D, q_0, T \rangle$, an initial register value d_0, and a target location $q_f \in Q$, we say that q_f is almost-surely reachable for k processes if $\mathbb{P}(\langle q_0^k, d_0 \rangle, [\![\Diamond q_f]\!]) = 1$.

▶ **Example 1.b.** *Consider again the protocol depicted in Fig. 1, with initial register content* 0*. As we explained already, for* $k = 1$*, the final state is not reachable at all, for any scheduler (here as* $k = 1$*, the scheduler only has to solve non-determinism in the protocol).*

When $k = 2$*, one easily sees that the final state is reachable: it suffices that both processes go to* q_2 *together, from where one process may write value* 2 *in the register, which the other process can read and go to* q_f*. Notice that this does not ensure that* q_f *is reachable almost-surely for this* k *(and actually, it is not; see Example 1.c).*

We aim here at finding cut-offs for almost-sure reachability, i.e., we seek the existence of a threshold such that almost-sure reachability (or its negation) holds for all larger values.

▶ **Definition 2.** Fix a protocol $\mathcal{P} = \langle Q, D, q_0, T \rangle$, $d_0 \in D$, and $q_f \in Q$. An integer $k \in \mathbb{N}$ is a *cut-off for almost-sure reachability* (shortly a *cut-off*) for $\mathcal{P}$, d_0 and q_f if one of the following two properties holds:

- for all $h \geq k$, we have $\mathbb{P}(\langle q_0^h, d_0 \rangle, [\![\Diamond q_f]\!]) = 1$. In this case k is a *positive* cut-off;
- for all $h \geq k$, we have $\mathbb{P}(\langle q_0^h, d_0 \rangle, [\![\Diamond q_f]\!]) < 1$. Then k is a *negative* cut-off.

An integer k is a *tight* cut-off if it is a cut-off and $k - 1$ is not.

Notice that from the definition, cut-offs need not exist for a given distributed system. Our main result precisely states that cut-offs do always exist, and that we can decide their nature.

▶ **Theorem 3.** *For any protocol* $\mathcal{P}$*, any initial register value* d_0 *and any target location* q_f*, there always exists a cut-off for almost-sure reachability, whose value is at most doubly-exponential in the size of* $\mathcal{P}$*. Whether it is a positive or a negative cut-off can be decided in* EXPSPACE*, and is* PSPACE*-hard.*

▶ Remark. When dropping the condition on non-atomic read/write operations , and allowing transitions with atomic read/write operations (i.e. one process is ensured to perform a read and a write operation without to be interrupted by another process), the existence of a cut-off (Theorem 3) is not ensured. This is demonstrated with the protocol of Fig. 2 : one easily checks (e.g., inductively on the number of processes, since processes that end up in q_2 play no role anymore) that state q_f is reached with probability 1 if, and only if, the number of processes is odd.

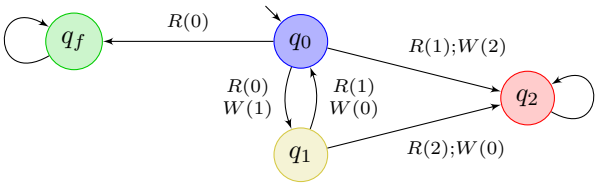


Figure 2 Example of a register protocol with atomic read/write operations.

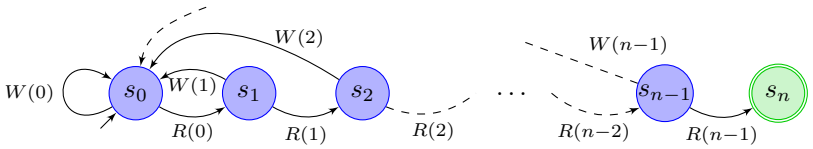


Figure 3 A "filter" protocol $\mathcal{F}_n$ for $n > 0$.

3 Properties of register protocols

3.1 Example of a register protocol

We illustrate our model with a family of register protocols $(\mathcal{F}_n)_{n>0}$, depicted in Fig. 3. For a fixed n, protocol $\mathcal{F}_n$ has $n+1$ states and n different data; intuitively, in order to move from s_i to s_{i+1}, two processes are needed: one writes i in the register and goes back to s_0, and the second process can proceed to s_{i+1} by reading i. Since backward transitions to s_0 are always possible and since states can always exit s_0 by writing a 0 and reading it afterwards, no deadlock can ever occur so the main question remains to determine if s_n is reachable by one of the processes as we increase the number of initial processes. As shown in Lemma 4, the answer is positive: $\mathcal{F}_n$ has a tight linear positive cut-off; it actually behaves like a "filter", that can test if at least n processes are running together. We exploit this property later in Section 4.4.

▶ **Lemma 4.** *Fix $n \in \mathbb{N}$. The "filter" protocol $\mathcal{F}_n$, depicted in Fig. 3, with initial register value 0 and target location s_n, has a tight positive cut-off equal to n.*

3.2 Basic results

In this section, we consider a register protocol $\mathcal{P} = \langle Q, D, q_0, T\rangle$, its associated distributed system $\mathcal{S}_\mathcal{P} = \langle \Gamma, Pr\rangle$, an initial register value $d_0 \in D$ and a target state $q_f \in Q$. We define a partial order $\preceq$ over the set Γ of configurations as follows: $\langle \mu, d\rangle \preceq \langle \mu', d'\rangle$ if, and only if, $d = d'$ and $\overline{\mu} = \overline{\mu'}$ and $\mu \sqsubseteq \mu'$. Note that with respect to the classical order over multisets, we require here that the supports of μ and μ' be the same (we add in fact a finite information to hold for the comparison). We know from Dickson's lemma that $\langle \mathbb{N}^Q, \sqsubseteq\rangle$ is a wqo and since Q, D and the supports of multisets in $\mathbb{N}^Q$ are finite, we can deduce the following lemma.

▶ **Lemma 5.** *$\langle \Gamma, \preceq\rangle$ is a wqo.*

We will give some properties of register protocols, but first we introduce some further notations. Given a set of configuration $\Delta \subseteq \Gamma$, we define $\mathrm{Pre}^*(\Delta)$ and $\mathrm{Post}^*(\Delta)$ as follows:

$$\mathrm{Pre}^*(\Delta) = \{\gamma \in \Gamma \mid \exists \gamma' \in \Delta. \gamma \to^* \gamma'\} \qquad \mathrm{Post}^*(\Delta) = \{\gamma' \in \Gamma \mid \exists \gamma \in \Delta. \gamma \to^* \gamma'\}$$

We also define the set $[\![q_f]\!]$ of configurations we aim to reach as $\{\gamma \in \Gamma \mid st(\gamma)(q_f) > 0\}$. It holds that $\gamma \in \mathrm{Pre}^*([\![q_f]\!])$ if, and only if, there exists a path in $[\![\Diamond q_f]\!]$ starting in γ.

As already mentioned, when $\langle \mu, d\rangle \to \langle \mu', d'\rangle$ in $\mathcal{S}_\mathcal{P}$, then $|\mu| = |\mu'|$, i.e., the multisets μ and μ' have the same cardinality. This implies that given $k > 0$, the set $\mathrm{Post}^*(\{\langle q_0^k, d_0\rangle\})$ is finite (remember that Q and D are finite). As a consequence, for a fixed k, checking whether $\mathbb{P}(\langle q_0^k, d_0\rangle, [\![\Diamond q_f]\!]) = 1$ can be easily achieved by analyzing the finite-state discrete-time Markov chain $\mathcal{S}_\mathcal{P}^k$ [5].

▶ **Lemma 6.** *Let $k > 0$. We have $\mathbb{P}(\langle q_0^k, d_0\rangle, [\![\Diamond q_f]\!]) = 1$ if, and only if, $\mathrm{Post}^*(\{\langle q_0^k, d_0\rangle\}) \subseteq \mathrm{Pre}^*([\![q_f]\!])$.*

The difficulty here precisely lies in finding such a k and in proving that, once we have found one correct value for k, all larger values are correct as well (to get the cut-off property). Characteristics of register protocols provide us with some tools to solve this problem. We base our analysis on reasoning on the set of configurations reachable from initial configurations in $\uparrow\{\langle q_0, d_0\rangle\}$ (the upward closure of $\{\langle q_0, d_0\rangle\}$ w.r.t. $\preceq$), remember that since the order $\langle \Gamma, \preceq\rangle$ requires equality of support for elements to be comparable, we have that $\uparrow\{\langle q_0, d_0\rangle\} = \bigcup_{k\in\mathbb{N}\setminus\{0\}}\{\langle q_0^k, d_0\rangle\}$. We begin by showing that this set of reachable configurations and the set of configurations from which $[\![q_f]\!]$ is reachable are both upward-closed. Thanks to Lemma 5, they can be represented as upward closures of finite sets. To show that $\mathrm{Post}^*(\uparrow\{\langle q_0, d_0\rangle\})$ is upward-closed, we prove that register protocols enjoy the following monotonicity property. A similar property is given in [11] and derives from the non-atomicity of operations.

▶ **Lemma 7.** *Let γ_1, γ_2, and γ_2' be configurations in Γ. If $\gamma_1 \to^* \gamma_2$ and $\gamma_2 \preceq \gamma_2'$, then there exists $\gamma_1' \in \Gamma$ such that $\gamma_1' \to^* \gamma_2'$ and $\gamma_1 \preceq \gamma_1'$.*

We point out that $\mathrm{Pre}^*([\![q_f]\!])$ is clearly upward-closed, since if $[\![q_f]\!]$ can be reached from some configuration γ, it can also be reached by a larger configuration by keeping the extra copies idle. As a corollary:

▶ **Lemma 8.** $\mathrm{Post}^*(\uparrow\{\langle q_0, d_0\rangle\})$ *and* $\mathrm{Pre}^*([\![q_f]\!])$ *are upward-closed sets in* $\langle \Gamma, \preceq\rangle$.

3.3 Existence of a cut-off

From Lemma 8, and from the fact that $\langle \Gamma, \preceq\rangle$ is a wqo, there must exist two finite sequences of configurations $(\theta_i)_{1\le i\le n}$ and $(\eta_i)_{1\le i\le m}$ such that $\mathrm{Post}^*(\uparrow\{\langle q_0, d_0\rangle\}) = \uparrow\{\theta_1, \ldots, \theta_n\}$ and $\mathrm{Pre}^*([\![q_f]\!]) = \uparrow\{\eta_1, \ldots, \eta_m\}$. By analyzing these two sequences, we now prove that any register protocol has a cut-off (for any initial register value and any target location).

We let $\Delta, \Delta' \subseteq \Gamma$ be two upward-closed sets (for $\preceq$). We say that Δ *is included in* Δ' *modulo single-state incrementation* whenever for every $\gamma \in \Delta$, for every $q \in \overline{\gamma}$, there is some $k \in \mathbb{N}$ such that $\gamma \oplus q^k \in \Delta'$. Note that this condition can be checked using only comparisons between minimal elements of Δ and Δ'. In particular, we have the following lemma.

▶ **Lemma 9.** $\mathrm{Post}^*(\uparrow\{\langle q_0, d_0\rangle\})$ *is included in* $\mathrm{Pre}^*([\![q_f]\!])$ *modulo single-state incrementation if, and only if, for all $i \in [1; n]$, and for all $q \in \overline{\theta_i}$, there exists $j \in [1; m]$ such that $data(\theta_i) = data(\eta_j)$ and $\overline{\theta_i} = \overline{\eta_j}$ and $st(\eta_j)(q') \le st(\theta_i)(q')$ for all $q' \in Q \setminus \{q\}$.*

Using the previous characterization of inclusion modulo single-state incrementation for $\mathrm{Post}^*(\uparrow\{\langle q_0, d_0\rangle\})$ and $\mathrm{Pre}^*([\![q_f]\!])$ together with the result of Lemma 6, we are able to provide a first characterization of the existence of a negative cut-off.

▶ **Lemma 10.** *If* $\mathrm{Post}^*(\uparrow\{\langle q_0, d_0\rangle\})$ *is not included in* $\mathrm{Pre}^*([\![q_f]\!])$ *modulo single-state incrementation, then* $\max_{1\leq i\leq n}(|st(\theta_i)|)$ *is a negative cut-off.*

We now prove that if the condition of Lemma 10 fails to hold, then there is a positive cut-off.In order to make our claim precise, for every $i \in [1; n]$ and for any $q \in \overline{\theta_i}$, we let $d_{i,q} = \max\{(|st(\eta_j)(q) - st(\theta_i)(q)|) \mid 1 \leq j \leq m \text{ and } \overline{\theta_i} = \overline{\eta_j}\}$.

▶ **Lemma 11.** *If* $\mathrm{Post}^*(\uparrow\{\langle q_0, d_0\rangle\})$ *is included in* $\mathrm{Pre}^*([\![q_f]\!])$ *modulo single-state incrementation, then* $\max_{1\leq i\leq n}(|st(\theta_i)| + \sum_{q\in\overline{\theta_i}} d_{i,q})$ *is a positive cut-off.*

The last two lemmas entail our first result:

▶ **Theorem 12.** *Any register protocol admits a cut-off (for any given initial register value and target state).*

4 Detecting negative cut-offs

We develop an algorithm for deciding whether a distributed system associated with a register protocol has a negative cut-off. Thanks to Theorem 12, this can also be used to detect the existence of a positive cut-off. Our algorithm relies on the construction and study of a *symbolic graph*, as we define below: for any given protocol $\mathcal{P}$, the symbolic graph has bounded size, but can be used to reason about *arbitrarily large* distributed systems built from $\mathcal{P}$. It will store sufficient information to decide the existence of a negative cut-off.

4.1 k-bounded symbolic graph

In this section, we consider a register protocol $\mathcal{P} = \langle Q, D, q_0, T\rangle$, its associated distributed system $\mathcal{S}_\mathcal{P} = \langle \Gamma, Pr\rangle$, an initial register value $d_0 \in D$, and a target location $q_f \in Q$ of $\mathcal{P}$. With $\mathcal{P}$, we associate a finite-state graph, called *symbolic graph of index* k, which for k large enough contains enough information to decide the existence of a negative cut-off.

▶ **Definition 13.** Let k be an integer. The *symbolic graph of index* k associated with $\mathcal{P}$ and d_0 is the transition system $\mathcal{G} = \langle V, v_0, E\rangle$ where

- $V = \mathbb{N}_k^Q \times 2^Q \times D$ contains triples made of a multiset of states of Q of size k, a subset of Q, and the content of the register; the multiset (called *concrete part* hereafter) is used to exactly keep track of a fixed set of k processes, while the subset of Q (the *abstract part*) encodes the support of the arbitrarily many remaining processes;
- $v_0 = \langle q_0^k, \{q_0\}, \{d_0\}\rangle$;
- transitions are of two types, depending whether they involve a process in the concrete part or a process in the abstract part. Formally, there is a transition $\langle \mu, S, d\rangle \to \langle \mu', S', d'\rangle$ whenever there is a transition $(q, O, d'', q') \in T$ such that $d = d' = d''$ if $O = R$ and $d' = d''$ if $O = W$, and one of the following two conditions holds:
  - either $S' = S$ and $q \sqsubseteq \mu$ (that is, $\mu(q) > 0$) and $\mu' = \mu \ominus q \oplus q'$;
  - or $\mu = \mu'$ and $q \in S$ and $S' \in \{S \setminus \{q\} \cup \{q'\}, S \cup \{q'\}\}$.

The symbolic graph of index k can be used as an abstraction of distributed systems made of at least $k + 1$ copies of $\mathcal{P}$: it keeps full information of the states of k processes, and only gives the support of the states of the other processes. In particular, the symbolic graph of index 0 provides only the states appearing in each configuration of the system.

▶ **Example 1.c.** *Consider the protocol depicted in Fig. 1. Its symbolic graph of index* 0 *is depicted in Fig. 4 (where self-loops have been omitted). Notice that the final state (representing all configurations containing* q_f*) is reachable from any state of this symbolic graph. However, our original protocol* $\mathcal{P}$ *of Fig. 1 does not have a positive cut-off (assuming initial register value* 0*): indeed, with positive probability, a single process will go to* q_1 *and immediately writes* 1 *in the register, thus preventing any other process to leave* q_0*; then one may check that the process in* q_1 *alone cannot reach* q_f*, so that the probability of reaching* q_f *from* q_0^k *is strictly less than* 1*, for any* $k > 0$*. This livelock is not taken into account in the symbolic graph of index* 0*, because from any configuration with support* $\{q_0, q_1\}$ *and register data equal to* 1*, the symbolic graph has a transition to the configuration with support* $\{q_0, q_1, q_2\}$*, which only exists in the concrete system when there are at least two processes in* q_1*. As we prove in the following, analyzing the symbolic graph for a sufficiently large index guarantees to detect such a situation.*

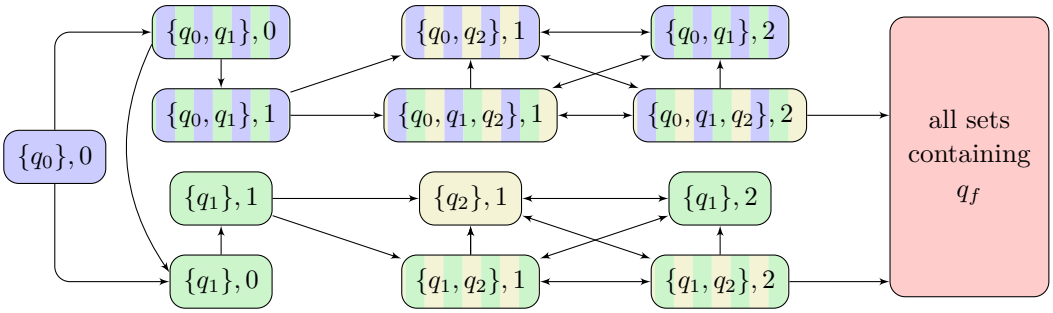


Figure 4 Symbolic graph (of index 0) of the protocol of Fig. 1 (self-loops omitted).

For any index k, the symbolic graph achieves the following correspondence:

▶ **Lemma 14.** *Given two states* $\langle \mu, S, d\rangle$ *and* $\langle \mu', S', d'\rangle$*, there is a transition from* $\langle \mu, S, d\rangle$ *to* $\langle \mu', S', d'\rangle$ *in the symbolic graph* $\mathcal{G}$ *of index* k *if, and only if, there exist multisets* δ *and* δ' *with respective supports* S *and* S'*, and such that* $\langle \mu \oplus \delta, d\rangle \to \langle \mu' \oplus \delta', d'\rangle$ *in* $\mathcal{S}_{\mathcal{P}}$*.*

4.2 Deciding the existence of a negative cut-off

We now explain how the symbolic graph can be used to decide the existence of a negative cut-off. As said in Lemma 8, the set $\mathrm{Pre}^*(\llbracket q_f \rrbracket)$ is upward-closed in $\langle \Gamma, \preceq\rangle$ and there is a finite set of configurations $\{\eta_i = \langle \mu_i, d_i\rangle \mid 1 \le i \le m\}$ such that $\mathrm{Pre}^*(\llbracket q_f \rrbracket) = \uparrow\{\eta_i \mid 1 \le i \le m\}$. We let $K = \max\{st(\eta_i)(q) \mid q \in Q,\ 1 \le i \le m\}$. We show in this part that for our purpose, it is enough to consider the symbolic graph of index $K \cdot |Q|$ and in the next section, we provide a bound on K.

▶ **Lemma 15.** *There is a negative cut-off for* $\mathcal{P}$*,* d_0 *and* q_f *if, and only if, there is a node in the symbolic graph of index* $K \cdot |Q|$ *that is reachable from* $\langle q_0^{K\cdot|Q|}, \{q_0\}, d_0\rangle$ *but from which no configuration involving* q_f *is reachable.*

Proof. We begin with the converse implication, assuming that there is a state $\langle \mu, S, d\rangle$ in the symbolic graph of index $K \cdot |Q|$ that is reachable from $(q_0^{K\cdot|Q|}, \{q_0\}, d_0)$ and from which no configuration in $\llbracket q_f \rrbracket$ is reachable. Applying Lemma 14, there exist multisets $\delta_0 = q_0^N$ and δ, with respective supports $\{q_0\}$ and S, such that $\langle \mu \oplus \delta, d\rangle$ is reachable from $\langle q_0^{K\cdot|Q|} \oplus \delta_0, d_0\rangle$. If location q_f was reachable from $\langle \mu \oplus \delta, d\rangle$ in the distributed system, then there would exist

a path from $\langle \mu, S, d\rangle$ to a state involving q_f in the symbolic graph, which contradicts our hypothesis. By Lemma 7, it follows that such a configuration $\langle \mu \oplus \delta', d\rangle$ — which cannot reach q_f — can be reached from $\langle q_0^{K\cdot|Q|} \oplus q_0^{N'}, d_0\rangle$ for any $N' \geq N$: hence it cannot be the case that q_f is reachable almost-surely for any $N' \geq N$. Therefore there cannot be a positive cut-off, which implies that there is a negative one (from Theorem 12).

Conversely, assume that there is a negative cut-off: then for some $N > K \cdot |Q|$, the distributed system $\mathcal{S}_{\mathcal{P}}^N$ with N processes has probability less than 1 of reaching $[\![q_f]\!]$ from q_0^N. This system being finite, there must exist a reachable configuration $\langle \mu, d\rangle$ from which q_f is not reachable [5]. Hence $\langle \mu, d\rangle \notin \mathrm{Pre}^*([\![q_f]\!])$, entailing that for all $i \leq m$, there is a location q^i such that $\mu(q^i) < \mu_i(q^i) \leq K$. Then there must exist a reachable state $\langle \kappa, S, d\rangle$ of the symbolic graph of index $K \cdot |Q|$ for which $\kappa(q^i) = \mu(q^i)$ and $q^i \notin S$, for all $1 \leq i \leq m$: it indeed suffices to follow the path from $\langle q_0^N, d_0\rangle$ to $\langle \mu, d\rangle$ while keeping track of the processes that end up in some q^i in the concrete part; this is possible because the concrete part has size at least $K \cdot |Q|$.

It remains to be proved that no state involving q_f is reachable from $\langle \kappa, S, d\rangle$ in the symbolic graph. If it were the case, then by Lemma 14, there would exist δ with support S such that $[\![q_f]\!]$ is reachable from $\langle \kappa \oplus \delta, d\rangle$ in the distributed system. Then $\langle \kappa \oplus \delta, d\rangle \in \mathrm{Pre}^*([\![q_f]\!])$, so that for some $1 \leq i \leq m$, $(\kappa \oplus \delta)(q^i) \geq \mu_i(q^i)$, which is not possible as $\kappa(q^i) < \mu_i(q^i)$ and q^i is not in the support S of δ. This contradiction concludes the proof. ◀

▶ Remark. Besides the existence of a negative cut-off, this proof also provides us with an upper bound on the tight cut-off, as we shall see in Section 5.

4.3 Complexity of the algorithm

We now consider the complexity of the algorithm that can be deduced from Lemma 15. Using results by Rackoff on the coverability problem in Vector Addition Systems [19], we can bound K – and consequently the size of the needed symbolic graph – by a *double-exponential* in the size of the protocol. Therefore, it suffices to solve a reachability problem in NLOGSPACE [20] on this doubly-exponential graph: this boils down to NEXPSPACE with regard to the protocol's size, hence EXPSPACE by Savitch's theorem [20].

▶ **Theorem 16.** *Deciding the existence of a negative cut-off is in* EXPSPACE.

4.4 PSPACE-hardness for deciding cut-offs

Our proof is based on the encoding of a linear-bounded Turing machine [20]: we build a register protocol for which there is a negative cut-off if, and only if, the machine reaches its final state q_{halt} with the tape head reading the last cell of the tape.

▶ **Theorem 17.** *Deciding the existence of a negative cut-off is* PSPACE*-hard.*

Write n for the size of the tape of the Turing machine. We assume (without loss of generality) that the machine is deterministic, and that it accepts only if it ends in its halting state q_{halt} while reading the last cell of the tape. Our reduction works as follows: some processes of our network will first be assigned an index i in $[1; n]$ indicating the cell of the tape they shall encode during the simulation. The other processes are stuck in the initial location, and will play no role. The state q and position j of the head of the Turing machine are stored in the register. During the simulation phase, when a process is scheduled to play, it checks in the register whether the tape head is on the cell it encodes, and in that case it performs the transition of the Turing machine. If the tape head is not on the cell it encodes,

the process moves to the target location (which we consider as the target for the almost-sure reachability problem). Finally, upon seeing (q_{halt}, n) in the register, all processes move to a $(n+1)$-filter protocol $\mathcal{F}_{n+1}$ (similar to that of Fig. 3) whose last location s_{n+1} is the aforementioned target location.

If the Turing machine halts, then the corresponding run can be mimicked with exactly one process per cell, thus giving rise to a finite run of the distributed system where n processes end up in the $(n+1)$-filter (and the other processes are stuck in the initial location); from there s_{n+1} cannot be reached. If the Turing machine does not halt, then assume that there is an infinite run of the distributed system never reaching the target location. This run cannot get stuck in the simulation phase forever, because it would end up in a strongly connected component from which the target location is reachable. Thus this run eventually reaches the $(n+1)$-filter, which requires that at least $n+1$ processes participate in the simulation (because with n processes it would simulate the exact run of the machine, and would not reach q_{halt}, while with fewer processes the tape head could not go over cells that are not handled by a process). Thus at least $n+1$ processes would end up in the $(n+1)$-filter, and with probability 1 the target location should be reached.

5 Bounds on cut-offs

5.1 Existence of exponential tight negative cut-offs

We exhibit a family of register protocols that admits negative cut-off exponential in the size of the protocol. The construction reuses ideas from the PSPACE-hardness proof. Our register protocol has two parts: one part simulates a counter over n bits, and requires a *token* (a special value in the register) to perform each step of the simulation. The second part is used to generate the tokens (i.e., writing 1 in the register). Figure 5 depicts our construction. We claim that this protocol, with # as initial register value and q_f as target location, admits a negative tight cut-off larger than 2^n: in other terms, there exists $N > 2^n$ such that the final state will be reached with probability strictly less than 1 in the distributed system made of at least N processes (starting with # in the register), while the distributed system with 2^n processes will reach the final state almost-surely. In order to justify this claim, we explain now the intuition behind this protocol.

We first focus on the first part of the protocol, containing nodes named a_i, b_i, c_i, d_i and s_i. This part can be divided into three phases: the initialization phase lasts as long as the register contains #; the counting phase starts when the register first contains *halt*; the simulation phase is the intermediate phase.

During the initialization phase, processes move to locations a_i and *tok*, until some process in *tok*writes 1 in the register (or until some process reaches q_f, using a transition from a_i to q_f while reading #).

Write γ_0 for the configuration reached when entering the simulation phase (i.e., when 1 is written in the register for the first time). We assume that $st(\gamma_0)(a_i) > 0$ for some i, as otherwise all the processes are in *tok*, and they all will eventually reach q_f. Now, we notice that if $st(\gamma_0)(a_i) = 0$ for some i, then location d_n cannot be reached, so that no process can reach the counting phase. In that case, some process (and actually all of them) will eventually reach q_f. We now consider the case where $st(\gamma_0)(a_i) \geq 1$ for all i. One can prove (inductively) that d_i is reachable when $st(\gamma_0)(tok) \geq 2^i$. Hence d_n, and thus also s_0, can be reached when $st(\gamma_0)(tok) \geq 2^n$. Assuming q_f is not reached, the counting phase must never contain more than n processes, hence we actually have that $st(\gamma_0)(a_i) = 1$. With this new condition, s_0 is reached if, and only if, $st(\gamma_0)(tok) \geq 2^n$. When the latter condition

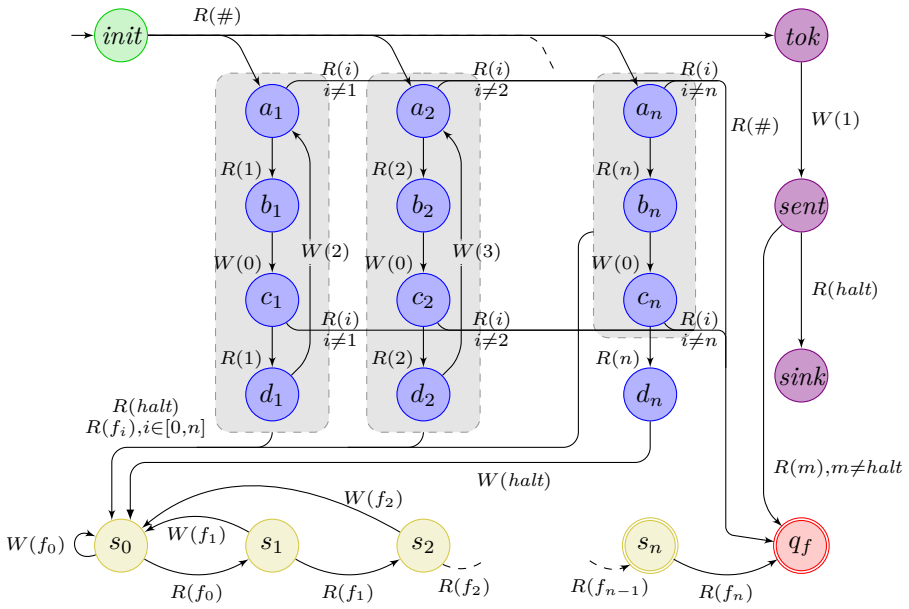


Figure 5 Simulating an exponential counter: grey boxes contain the nodes used to encode the bits of the counter; yellow nodes at the bottom correspond to the filter module from Fig. 3; purple nodes *tok*, *sent*and *sink*correspond to the second part of the protocol, and are used to produce tokens. Missing *read* edges are assumed to be self-loops.

is not true, q_f will be reached almost-surely, which proves the second part of our claim: the final location is reached almost-surely in systems with strictly less than $n + 2^n$ copies of the protocol.

We now consider the case of systems with at least $n + 2^n$ processes. We exhibit a finite execution of those systems from which no continuation can reach q_f, thus proving that q_f is reached with probability strictly less than 1 in those systems. The execution is as follows: during initialization, for each i, one process enters a_i; all other processes move to *tok*, and one of them write 1 in the register. The n processes in the simulation phase then simulate the consecutive incrementations of the counter, consuming one token at each step, until reaching d_n. At that time, all the processes in *tok*move to *sent*, and the process in d_n writes *halt* in the register and enters s_0. The processes in the simulation phase can then enter s_0, and those in *sent* can move to *sink*. We now have n processes in s_0, and the other ones in *sink*. According to Lemma 4, location q_f cannot be reached from this configuration, which concludes our proof.

▶ **Theorem 18.** *There exists a family of register protocols which, equipped with an initial register value and a target location, admit negative tight cut-offs whose size are exponential in the size of the protocol.*

▶ Remark. The question whether there exists protocols with exponential *positive* cut-offs remains open. The family of *filter* protocols described at Section 3.1 is an example of protocols with a linear positive cut-off.

5.2 Upper bounds on tight cut-offs

The results (and proofs) of Section 4 can be used to derive upper bounds on tight cut-offs. We make this explicit in the following theorem.

▶ **Theorem 19.** *For a protocol $\mathcal{P} = \langle Q, D, q_0, T\rangle$ equipped with an initial register value $d_0 \in D$ and a target location $q_f \in Q$, the tight cut-off is at most doubly-exponential in $|\mathcal{P}|$.*

6 Conclusions and future works

We have shown that in networks of identical finite-state automata communicating (non-atomically) through a single register and equipped with a fair stochastic scheduler, there always exists a cut-off on the number of processes which either witnesses almost-sure reachability of a specific control-state (positive cut-off) or its negation (negative cut-off). This cut-off determinacy essentially relies on the monotonicity induced by our model, which allows to use well-quasi order techniques. By analyzing a well-chosen symbolic graph, one can decide in EXPSPACE whether that cut-off is positive, or negative, and we proved this decision problem to be PSPACE-hard. This approach allows us to deduce some doubly-exponential bounds on the value of the cut-offs. Finally, we gave an example of a network in which there is a negative cut-off, which is exponential in the size of the underlying protocol. Note however that no such lower-bound is known yet for positive cut-offs.

We have several further directions of research. First, it would be nice to fill the gap between the PSPACE lower bound and the EXPSPACE upper bound for deciding the nature of the cut-off. We would like also to investigate further atomic read/write operations, which generate non-monotonic transition systems, but for which we would like to decide whether there is a cut-off or not. Finally, we believe that our techniques could be extended to more general classes of properties, for instance, universal reachability (all processes should enter a distinguished state), or liveness properties.

References

1 C. Aiswarya, Benedikt Bollig, and Paul Gastin. An automata-theoretic approach to the verification of distributed algorithms. In Luca Aceto and David de Frutos-Escrig, editors, *Proceedings of the 26th International Conference on Concurrency Theory (CONCUR'15)*, volume 42 of *Leibniz International Proceedings in Informatics*, pages 340–353. Leibniz-Zentrum für Informatik, September 2015. doi:{10.4230/LIPIcs.CONCUR.2015.340}.

2 Benjamin Aminof, Swen Jacobs, Ayrat Khalimov, and Sasha Rubin. Parametrized model checking of token-passing systems. In Kenneth L. McMillan and Xavier Rival, editors, *Proceedings of the 15th International Workshop on Verification, Model Checking, and Abstract Interpretation (VMCAI'14)*, volume 8318 of *Lecture Notes in Computer Science*, pages 262–281. Springer-Verlag, January 2014. doi:{10.1007/978-3-642-54013-4_15}.

3 Benjamin Aminof, Sasha Rubin, and Florian Zuleger. On the expressive power of communication primitives in parameterised systems. In Martin Davis, Ansgar Fehnker, Annabelle K. McIver, and Andrei Voronkov, editors, *Proceedings of the 20th International Conference Logic Programming and Automated Reasoning (LPAR'15)*, volume 9450 of *Lecture Notes in Computer Science*, pages 313–328. Springer-Verlag, November 2015.

4 Simon Außerlechner, Swen Jacobs, and Ayrat Khalimov. Tight cutoffs for guarded protocols with fairness. In Barbara Jobstmann and K. Rustan M. Leino, editors, *Proceedings of the 17th International Workshop on Verification, Model Checking, and Abstract Interpretation (VMCAI'16)*, volume 9583 of *Lecture Notes in Computer Science*, pages 476–494. Springer-Verlag, January 2016. doi:{10.1007/978-3-662-49122-5_23}.

5 Christel Baier and Joost-Pieter Katoen. *Principles of Model-Checking*. MIT Press, May 2008.

6 Benedikt Bollig, Paul Gastin, and Len Schubert. Parameterized verification of communicating automata under context bounds. In Joël Ouaknine, Igor Potapov, and James Worrell, editors, *Proceedings of the 8th Workshop on Reachability Problems in Computational Models (RP'14)*, volume 8762 of *Lecture Notes in Computer Science*, pages 45–57. Springer-Verlag, September 2014. doi:{10.1007/978-3-319-11439-2_4}.

7 Patricia Bouyer, Nicolas Markey, Mickael Randour, Arnaud Sangnier, and Daniel Stan. Reachability in networks of register protocols under stochastic schedulers. *CoRR*, abs/1602.05928, 2016. URL: http://arxiv.org/abs/1602.05928.

8 Edmund M. Clarke, Muralidhar Talupur, Tayssir Touili, and Helmut Veith. Verification by network decomposition. In Philippa Gardner and Nobuko Yoshida, editors, *Proceedings of the 15th International Conference on Concurrency Theory (CONCUR'04)*, volume 3170 of *Lecture Notes in Computer Science*, pages 276–291. Springer-Verlag, August-September 2004. doi:{10.1007/978-3-540-28644-8_18}.

9 Giorgio Delzanno, Arnaud Sangnier, Riccardo Traverso, and Gianluigi Zavattaro. On the complexity of parameterized reachability in reconfigurable broadcast networks. In Deepak D'Souza, Telikepalli Kavitha, and Jaikumar Radhakrishnan, editors, *Proceedings of the 32nd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'12)*, volume 18 of *Leibniz International Proceedings in Informatics*, pages 289–300. Leibniz-Zentrum für Informatik, December 2012. doi:LIPIcs.FSTTCS.2012.289.

10 Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. Parameterized verification of ad hoc networks. In Paul Gastin and François Laroussinie, editors, *Proceedings of the 21st International Conference on Concurrency Theory (CONCUR'10)*, volume 6269 of *Lecture Notes in Computer Science*, pages 313–327. Springer-Verlag, September 2010. doi:{10.1007/978-3-642-15375-4_22}.

11 Antoine Durand-Gasselin, Javier Esparza, Pierre Ganty, and Rupak Majumdar. Model checking parameterized asynchronous shared-memory systems. In Daniel Kroening and Corina S. Pasareanu, editors, *Proceedings of the 27th International Conference on Computer Aided Verification (CAV'15)*, volume 9206 of *Lecture Notes in Computer Science*, pages 67–84. Springer-Verlag, July 2015. doi:{10.1007/978-3-319-21690-4_5}.

12 E. Allen Emerson and Vineet Kahlon. Reducing model checking of the many to the few. In David McAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction (CADE'00)*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 236–254. Springer-Verlag, June 2000. doi:{10.1007/10721959_19}.

13 E. Allen Emerson and Kedar Namjoshi. On reasoning about rings. *International Journal of Foundations of Computer Science*, 14(4):527–550, August 2003. doi:{10.1142/S0129054103001881}.

14 Javier Esparza. Keeping a crowd safe: On the complexity of parameterized verification (invited talk). In Ernst W. Mayr and Natacha Portier, editors, *Proceedings of the 31st Symposium on Theoretical Aspects of Computer Science (STACS'14)*, volume 25 of *Leibniz International Proceedings in Informatics*, pages 1–10. Leibniz-Zentrum für Informatik, March 2014. doi:{10.4230/LIPIcs.STACS.2014.1}.

15 Javier Esparza, Alain Finkel, and Richard Mayr. On the verification of broadcast protocols. In *Proceedings of the 14th Annual Symposium on Logic in Computer Science (LICS'99)*, pages 352–359. IEEE Comp. Soc. Press, July 1999. doi:{10.1109/LICS.1999.782630}.

16 Javier Esparza, Pierre Ganty, and Rupak Majumdar. Parameterized verification of asynchronous shared-memory systems. In Natasha Sharygina and Helmut Veith, editors, *Proceedings of the 25th International Conference on Computer Aided Verification (CAV'13)*,

volume 8044 of *Lecture Notes in Computer Science*, pages 124–140. Springer-Verlag, July 2013. doi:{10.1007/978-3-642-39799-8_8}.

17 Steven M. German and A. Prasad Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 39(3):675–735, July 1992.

18 Matthew Hague. Parameterised pushdown systems with non-atomic writes. In Supratik Chakraborty and Amit Kumar, editors, *Proceedings of the 31st Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'11)*, volume 13 of *Leibniz International Proceedings in Informatics*, pages 457–468. Leibniz-Zentrum für Informatik, December 2011. doi:{10.4230/LIPIcs.FSTTCS.2011.457}.

19 Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6:223–231, 1978. doi:{10.1016/0304-3975(78)90036-1}.

20 Michael Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997.

Regular transformations of data words through origin information *

Antoine Durand-Gasselin[1]** and Peter Habermehl[2]

[1] Aix Marseille Université, CNRS & Centrale Marseille
Antoine.Durand-Gasselin@centrale-marseille.fr
[2] IRIF, Univ. Paris Diderot & CNRS
Peter.Habermehl@liafa.univ-paris-diderot.fr

Abstract. We introduce a class of transformations of finite data words generalizing the well-known class of regular finite string transformations described by MSO-definable transductions of finite strings. These transformations map input words to output words whereas our transformations handle data words where each position has a letter from a finite alphabet and a data value. Each data value appearing in the output has as origin a data value in the input. As is the case for regular transformations we show that our class of transformations has equivalent characterizations in terms of deterministic two-way and streaming string transducers.

1 Introduction

The theory of transformations of strings (or words) over a finite alphabet has attracted a lot of interest recently. Courcelle [8] defined finite string transformations in a logical way using Monadic second-order definable graph transductions. Then, a breakthrough was achieved in [9] where it was shown that these transformations are equivalent to those definable by deterministic two-way finite transducers on finite words. In [1] deterministic streaming string transducers (SST) on finite words were introduced. This model is one-way but it is equipped with string variables allowing to store words. It is equivalent [1] to the deterministic two-way finite transducers and to MSO-definable transformations. Interestingly, the motivation behind SST was the more powerful SDST model [2]. SDST work on data words, i.e. words composed of couples of letters from a finite alphabet and an infinite data domain. However, they do not have the same nice theoretical properties as SST, for example they are not closed under composition because SDST have data variables allowing to store data values and *compare* data values with each other. Furthermore, there is no equivalent logical characterization.

In this paper, analogously to the case of string transformations of finite words, we obtain a class of transformations of finite data words which has an MSO characterization as well as equivalent characterizations in terms of deterministic

* This work was supported in part by the VECOLIB project (ANR-14-CE28-0018) and by the PACS project (ANR-14-CE28-0002).

** Part of this work was done while this author was at Technical University Munich.

two-way transducers and streaming transducers. To achieve this, we allow storing of data values in the transducers but not comparison.

As an example we consider the transformation taking a finite input word over $\{\#, a, b\}$ starting and finishing with a $\#$, together with associated data values from the integers, like $\binom{\#ab\#abb\#}{1\,24\,5\,671\,4}$ and produces as output the word where (1) $\#$'s are left unchanged, and between successive $\#$'s (2) words w in $\{a, b\}^*$ are transformed into $w^R w$ where w^R denotes the reverse image of w, and (3) the data value associated to each a is the value of the first $\#$ and the value for b's is the value of the second $\#$. So, the output for the example word is $\binom{\#baab\#bbaabb\#}{1\,5115\,5\,4455441\,4}$.

It is clear how a deterministic two-way transducer with the ability of storing data values can realize this transformation: it stores the first data value (1) in a *data variable* while outputting $\binom{\#}{1}$, then goes to the second $\#$, stores the corresponding data value (5) in a second data variable, and goes back one by one while producing $\binom{ba}{51}$. Then, it turns around at the first $\#$, goes again to the second $\#$ producing $\binom{ab}{15}$ and restarts the same process.

Now, to realize this transformation with a deterministic streaming string transducer one has to make with the fact that they can only go through the input word once from left to right. Nevertheless we will introduce a model which can realize the described transformation: in between two $\#'s$ it stores the so-far read string and its reverse in a *data word variable*. As the data value of the second $\#'$ is not known in the beginning it uses a *data parameter* p instead. For example, before the second $\#$, the stored word will be $\binom{baab}{p11p}$. When reading the second $\#$, it then replaces p everywhere by 5 and stores the result in another data word variable. The same repeats for the rest of the word until the end is reached and the output contained in a data word variable.

The same transformation can also be described logically. To define transformations on data words, a natural choice would be to use transducers with origin information and their corresponding MSO transductions studied in [6]. Basically, their semantics also takes into account information about the origin of a letter in the output, i.e. the position in the input from which it originates. Obviously, this can be generalized to data values by defining the data value of each output position as the data value in the input position from where the output originated. This definition is however not expressive enough to handle our example, since an input position can only be the origin of a bounded number of output positions but the data values attached to (unboundedly many) a's and b's between two successive $\#$'s come from the same two input positions.

Therefore, in this paper, we first introduce a logical characterization of word transformations with generalized origin information. Our logical characterization is an extension of classical MSO transductions with an additional functional MSO defined relation that maps each element of the interpretation (symbols of the output word) to an element of the interpreted structure (symbols of the input word). This generalization naturally defines transformations of data words; the data value at a given position of the output is the data value carried at the corresponding position in the input. This suffices to define the previously described example transformation.

Interestingly, our class of transformations is captured by a natural model of deterministic two-way transducers extended with data variables whose values can neither be tested nor compared. By adding data word variables (as in streaming string transducers) containing data values and parameters, we then manage, while preserving determinism, to restrict that model to a one-way model. Data parameters are placeholders for data values which can be stored in data word variables and later replaced by concrete data values. We show that this one-way model can be captured by MSO to achieve the equivalence of all three models.

2 MSO interpretations with MSO origin relation

2.1 Words, strings and data words

For S a set of symbols, we denote by S^* the set of finite words (i.e. the set of finite sequences of elements of S) over S. Given a word w, we can refer to its length ($|w|$), its first symbol ($w[0]$), the symbol at some position $i < |w|$ in the word ($w[i]$), some of its subwords (e.g. $w[i{:}j]$ with $0 \leq i \leq j < |w|$, the subword between positions i and j) etc. In this paper, we only consider finite words.

An alphabet (typically denoted Σ or Γ) is a finite set of symbols. Furthermore, we use a (potentially infinite) set of *data values* called Δ. In the sequel, we use *string* to refer to a (finite) word over a finite alphabet and *data word* to refer to a word over the cartesian product of a finite alphabet and a set of data values. Since symbols of data words consist of a letter (from the finite alphabet) and a data value, we can naturally refer to the data value at some position in a data word, or the string corresponding to some data word. Conversely a string together with a mapping from its position to Δ forms a data word.

A string w (over alphabet Σ) is naturally seen as a directed node-labeled graph (rather than considering edges only connecting two successive positions, we take the transitive closure: thus the graph is a finite linear order). The graph is then seen as an interpreted relational structure whose domain is the positions of w, with a binary edge predicate $<$, and a monadic predicate for each letter of Σ. We denote $\mathcal{S}_\Sigma$ the signature consisting of $<_{/2}$ and $\sigma_{/1}$ for each $\sigma \in \Sigma$.

Any string over alphabet Σ is an interpretation over a finite domain of $\mathcal{S}_\Sigma$, conversely any interpretation of $\mathcal{S}_\Sigma$ is a string if (1) its domain is finite, (2) $<$ defines a linear order and (3) at every position exactly one monadic predicate holds. We remark that (2) and (3) can be expressed as monadic second order (MSO) sentences. Two interpretations are isomorphic iff they are the same string.

With this logic based approach we have a very simple classical characterization of regular languages: a language L over alphabet Σ is regular iff there exists an MSO sentence φ (over signature $\mathcal{S}_\Sigma$) such that the set of interpretations of $\mathcal{S}_\Sigma$ with finite domain satisfying φ is the set of strings in L.

2.2 MSO interpretations

Using this model theoretic characterization of strings, we can define a class of transformations of strings. For the sake of clarity we consider transformations

of strings over alphabet Σ to strings over alphabet Γ. We now define an MSO interpretation of $\mathcal{S}_\Gamma$ in $\mathcal{S}_\Sigma$, as $|\Gamma|+2$ MSO formulas over signature $\mathcal{S}_\Sigma$: $\varphi_<$ with two free first-order variables and φ_{dom} and $(\varphi_\gamma)_{\gamma\in\Gamma}$ with one free first-order variable. Any interpretation $\mathcal{I}_\Sigma$ (of the signature $\mathcal{S}_\Sigma$) defines an interpretation of the structure $\mathcal{S}_\Gamma$: its domain is the set of elements of the domain of $\mathcal{I}_\Sigma$ satisfying φ_{dom} in $\mathcal{I}_\Sigma$, and the interpretation of the predicates over that domain is given by the truth value of each of the other MSO formulas.

An important remark is that if the interpretation $\mathcal{I}_\Sigma$ has finite domain, then so will the constructed interpretation of $\mathcal{S}_\Gamma$. Also, since we can express in MSO (over the signature $\mathcal{S}_\Gamma$) that the output structure is a string (with (2) and (3)), we can also express in MSO over the signature $\mathcal{S}_\Sigma$ that the output structure is a string, hence we can decide whether for any input string our MSO interpretation produces a string.

Above, we presented the core idea of Courcelle's definition [8] of MSO graph transductions. Courcelle further introduces the idea of interpreting the output structure in several copies of the input structures. To define such a transduction, we need to fix a *finite* set C of copies, the domain of the output structure will thus be a subset of the cartesian product of the domain of the input structure with the set of copies. The transduction consists of $|C|^2+(|\Gamma|+1)|C|+1$ MSO formulas over the input structure:

- the sentence φ_{indom} that states whether the input structure is in the input domain of the transduction,
- formulas φ^c_{dom} (with one free first-order variable) for each c in C, each stating whether a node x in copy c is a node of the output structure,
- formulas φ^c_γ (also with one free first-order variable), for each $c\in C$ and each $\alpha\in\Gamma$ which states whether a node x in copy c is labelled by α,
- and formulas $\varphi^{c,d}_<$ (with two free first-order variables, namely x,y) that states whether there exists an edge from x in copy c to y in copy d.

The semantics of these transformations naturally provides a notion of origin: by definition a node of the output structure is a position x in the copy c of the input structure (such that $\varphi^c_{dom}(x)$ is true).

2.3 Transduction of data words

Data words cannot be represented as finite structures (over a finite signature) but they can be seen as strings together with a mapping of positions to data values.

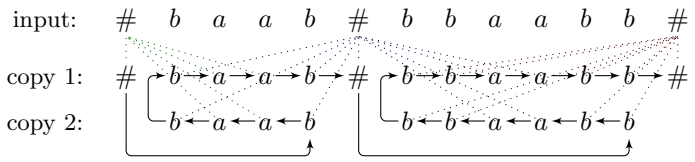
To define a data word transduction, we take a string transduction that we extend with an MSO relation between positions in the input word and positions in the output word. Formally we extend the definition of MSO transduction with $|C|$ MSO formulas (with two free first-order variables) $\varphi^c_{orig}(x,y)$, which we call the origin formulas, stating that position x in copy c (in the output string) corresponds to position y in the input string. We further impose that for any input word in the domain of the transformation and any x and $c\in C$ such that

x in copy c is in the output of the transformation, there exists exactly one y that validates $\varphi^c(x, y)$. We remark that this restriction can be ensured in MSO (over the input structure). Then, the data value at each output position is defined to be the data value at the corresponding input position.

We call MSOT the class of string transformations defined as MSO interpretations, and MSOT+O the class of data word transformation defined as MSO interpretations together with origin formulas. We remark that this definition of origin captures the usual origin information in the sense of [6] by fixing $\varphi^c_{orig}(x, y) \equiv (x = y)$.

2.4 The running example

Two copies suffice to define the transformation for the running example. For clarity, we do not represent the ordering relation $<$, but rather the successor relation.



φ_{indom} states the input word starts and ends with a #. $\varphi^1_{dom}(x)$ is true (every node in the first copy is part of the output), while $\varphi^2_{dom}(x) = \neg\#(x)$ tests the letter in the input at that position is not a #. The labeling formulas are the identity ($\varphi^1_a(x) = a(x)$,...) —the behaviour of the formula outside the output domain is considered irrelevant. $\varphi^{1,1}_<(x, y) = x < y$, and $\varphi^{2,2}_<(x, y)$ checks if there is a #-labeled position between position x and y (in the input): if so it ensures that $x < y$, if not it ensures $x > y$. $\varphi^{1,2}(x, y)$ and $\varphi^{1,2}(x, y)$ also distinguish cases whether there is a #-labeled position between x and y or not.

The origin information MSO formulas happen here to be the same for the two copies $\varphi^i(x, y)$ making cases on the letter x: if it is an a (resp. a b) it ensures y is the first #-labeled position before (resp. after) position x.

2.5 Properties

Defining word transformations through MSO interpretations yields some nice properties:

Theorem 1. *MSOT+O is closed under composition.*

Proof. MSOT is naturally closed under composition: given 2 mso transductions T_1 and T_2, (using C_1 and C_2 copies) we can define $T_1 \circ T_2$ as the MSO-interpretation T_1 of the MSO-interpretation T_2 of the original structure, which is an MSO-interpretation over $C_1 \times C_2$ copies.

In order to show the compositional closure of MSOT+O, it now suffices to define the origin information for the composition of two transductions T_1 and T_2

in MSOT+O. It is clear how to define formulas φ^c_{orig} that relate a position in the output with a position in the input, from the origin formulas of T_1 and T_2. We just need to show these origin formulas are functional; a fact that we easily derive from the functionality of the origin formulas of T_1 and T_2.

The (MSO)-typechecking problem of a transformation is defined as follows:

INPUT: Two MSO sentences $\varphi_{pre}, \varphi_{post}$ and an MSOT+O transformation T
OUTPUT: Does $w \models \varphi_{pre}$ imply that $T(w) \models \varphi_{post}$?

It consists in checking whether some property on the input implies a property on the output, those properties are here expressed in MSO.

Theorem 2. *MSO-typechecking of MSOT+O is decidable.*

Proof. An MSO formula can not reason about data values. Therefore it is sufficient to show that MSO-typechecking of MSOT is decidable. Since the output is defined as an MSO interpretation of the input, it is easy to convert an MSO formula on the output into an MSO formula on the input. We just need to check whether the input property implies that converted output property, on any input word, which is checking the universality of an MSO formula over finite strings.

2.6 MSO k-types

Since we present a generalisation of the classical MSO string transductions, the machine models that are expressively equivalent to our logical definition will be extensions of the classical machine models.

To show later that these logical transformations are captured by finite state machines, we use the notion of MSO k-types. We crucially use this notion (more precisely Theorem 3) to prove in Section 3 that we only need a finite number of data variables (Lemma 1) to store data values originating from the input.

Given a string w, we define its k-type as the set of MSO sentences of quantifier depth at most k (i.e. the maximum nesting of quantifiers is at most k) that hold for w. A crucial property is that the set of MSO k-types (which we denote Θ_k) is finite and defines an equivalence relation over strings which is a monoïd congruence of finite index. We refer the reader to [11] for more details.

These k-indexed congruences satisfy the following property: two k-equivalent strings will satisfy the same quantifier depth k MSO sentence.

We can extend this notion to MSO formulas with free first-order variables.

Theorem 3. *Given two strings w_1 and w_2 each with two distinguished positions x_1, y_1 and x_2, y_2. $(w_1, (x_1, y_1))$ and $(w_2, (x_2, y_2))$ satisfy the same MSO formulas with quantifier depth at most k and two free first order variables if:*

- *$w_1[x_1] = w_2[x_2]$ and $w_1[y_1] = w_2[y_2]$*
- *x_1, y_1 and x_2, y_2 occur in the same order in w_1 and w_2 (with the special case that if $x_1 = y_1$, then $x_2 = y_2$).*
- *The k-types of the two (strict) prefixes are the same, and the k-types of the two (strict) suffixes are the same, as well as the k-types of the two (strict) subwords between the two positions.*

Proof. Immediate with Ehrenfeucht-Fraïssé games.

3 Two-way transducers on data words

Two-way deterministic transducers on strings are known to be equivalent to MSO string transductions [9]. Since we process data words and output data words, we will naturally extend this model with a finite set of *data variables*. Notice that the data values in the input word do not influence the finite string part of the output. Therefore the transition function of the transducer may not perform any test on the values of those data variables. However the output word will contain some (if not all) data values of the input word, therefore the model may store some data value appearing in the input word in some variable, and when an output symbol is produced, this is done (deterministically) by combining some letter of the output alphabet together with the data value contained in some data variable.

We start by defining the classical two-way deterministic finite-state transducers (2dft) (with input alphabet Σ and output alphabet Γ) as a deterministic two-way automaton whose transitions are labeled by strings over Γ. The image of a string w by a 2dft A is defined (if w admits a run) as the concatenation of all the labels of the transitions along that run.

Definition 1. *A 2dft is a tuple $(\Sigma, \Gamma, Q, q_0, F, \delta)$ where:*

- *Σ and Γ are respectively the finite input and output alphabets ($\vdash, \dashv \notin \Sigma$)*
- *Q is the finite set of states, q_0 the initial state, and F the set of accepting states*
- *$\delta : Q \times (\Sigma \cup \{\vdash, \dashv\}) \to Q \times \{+1, -1\} \times \Gamma^*$ is the (two-way, Γ^*-labeled) transition function*

A (finite) run of a 2dft A over some string w is a finite sequence ρ of pairs of control states Q and positions in $[-1, |w|]$ (where -1 is supposed to be labeled by $\vdash$ and $|w|$ by $\dashv$), such that: $\rho(0) = (0, q_0)$, $\rho(|\rho|-1) \in \mathbb{N} \times F$ and at any position $k < |\rho| - 1$ in the run, if we denote $\rho(k) = (i_k, q_k)$ and $\rho(k+1) = i_{k+1}, q_{k+1}$, we have that $\delta(q_k, w(i_k)) = (q_{k+1}, i_{k+1} - i_k, u_{k+1})$ for some $u_{k+1} \in \Gamma^*$. Informally $+1$ corresponds to moving to the right in the input string and -1 to moving to the left. The output of A over w is simply the string $u_1 u_2 \dots u_{|\rho|-1}$. We denote $T(A)$ the (partial) transduction from Σ^* to Γ^* defined by A.

Notice that not every input string admits a finite run (since the transducer might get stuck or loop), but if w admits a finite run, it is unique and has length at most $|Q|(|w|+2)$, as this run visits any position at most $|Q|$ times. Therefore a run can also be defined as a mapping from positions of $\vdash w \dashv$ to $Q^{\leq |Q|}$ (sequences of states of length at most $|Q|$).

The next theorem states the equivalence between transformations defined by this two-way machine model and the logical definition of string transformations.

Theorem 4. *[9] Any string transformation from Σ^* to Γ^* defined by a 2dft can be defined as an MSO interpretation of Γ^* in Σ^* and vice versa.*

Now we define our two-way machine model, *two-way deterministic finite-state transducer with data variables* (2dftv) for data word transformations. We simply extend the 2dft by adding some data variables whose values are deterministically updated at each step of the machine.

Definition 2. *A 2dftv is a tuple* $(\Sigma, \Gamma, \Delta, Q, q_0, F, V, \mu, \delta)$ *where:*

- Σ *and* Γ *are respectively the input and output alphabets* $(\vdash, \dashv \notin \Sigma)$,
- Δ *is the (infinite) data domain,*
- Q *is the finite set of states,* q_0 *the initial state, and* F *the set of accepting states,*
- V *a finite set of data variables with a designated variable* $curr \in V$,
- $\mu : Q \times \Sigma \times (V \setminus \{curr\}) \to V$ *is the data variable update function,*
- $\delta : Q \times (\Sigma \cup \{\vdash, \dashv\}) \to Q \times \{+1, -1\} \times (\Gamma \times V)^*$ *is the (two-way,* $(\Gamma \times V)^*$*-labeled) transition function.*

We can define the semantics of a 2dftv like the semantics of an 2dft by extending the notion of run. Here, a run is labeled by positions and states but also by a valuation of the variables, i.e. a partial function β which assigns to variables from V values from Δ. This partial function is updated in each step (while reading a symbol different from the endmarkers $\vdash$ or $\dashv$) according to μ and additionally to the variable *curr* the current data value in the input is assigned. The output is obtained by substituting the data variables appearing in the label of the transition relation by their value according to β which we suppose to be always defined (this can be checked easily). Then, naturally a 2dftv defines a transduction from words over $\Sigma \times \Delta$ to words over $\Gamma \times \Delta$.

We call 2DFTV the class of all data word transductions definable by a 2dftv.

Theorem 5. *MSOT+O is included in 2DFTV.*

The challenge to show the theorem is to be able to extend the MSOT to 2DFT proof from [9], so as to be able to also carry in data variables all the necessary data values needed in the output.

We recall the key features in the proof of [9]. First, 2dft's are explicitly shown to be composable [7], which gives regular look-around (the ability to test if the words to the left and to the right of the reading head are in some regular languages) for free: a first pass rewrites the input right-to-left and adds the regular look-ahead, and the second pass re-reverses that word while adding the regular look-back. It is then possible (by reading that regular look-around) to implement MSO-jumps. Given an MSO formula φ with 2 free variables, an MSO-jump φ from position x consists in directly going to a position y such that $\varphi(x, y)$ holds. Using MSO-jumps 2dft can then simulate MSO transformations.

We show thereafter how to extend such a 2dft that takes as input the (look-around enriched) string and produces its image, to a 2dftv. The proof is then in three steps: first we show that a finite number of data variables is needed, then we briefly describe how to update those data variables: each transition of the 2dft being possibly replaced by a "fetching" of exactly one data variable,

and finally it is easy to see how to compose the preprocessing 2dft with that produced 2dftv.

To store only a finite number of data values, we will only store those which originate from a position on one side of the currently processed position and that are used on the other side of the currently processed position. The following lemma ensures a bound on the number of data variables.

Lemma 1. *Let w be a data word, x a position in w, and T a transducer. Denote k the quantifier depth of origin formulas. There are at most $|\Sigma||\Theta_k|^2$ positions $z > x$ such that there exists a position $y < x$ in some copy c such that $\varphi^c_{orig}(y, z)$ holds, i.e. that the data value carried by y in copy c is that of z.*

Proof. By contradiction, we use the pigeon hole principle. We can find two distinct positions z and z' such that the type of the subword between x and z and x and z' is the same, and the type of the suffix from z is the same as the type of the suffix from z'.

Let y a position, left of x where the data value of z is used, thus $\varphi^c_{orig}(y, z)$ holds. We apply Theorem 3 to $(w, (y, z))$ and $(w, (y, z'))$ and therefore $\varphi^c_{orig}(y, z')$ also holds, which contradicts the functionality of the relation φ_{orig}. □

It seems appropriate to name our data variables using MSO types. The data variables are thus $\Sigma \times \Theta_k \times \Theta_k \times \{l, r\}$, $(\sigma, \tau_1, \tau_2, l)$ denoting the data variable containing the data value from the position y (in the input word which is labeled by σ), left (l) of current positions, such that the prefix up to y has type τ_1, and the subword between y and current position has type τ_2.

With an appropriate value of k' (greater than k) the knowledge of the k'-types of the prefix and suffix of the word from the currently processed position, informs us for each data variable whether it contains a value or not, whether it is used at the current position and most importantly to which data variable the value should be transfered when a transition to the right (or the left) is taken.

Notice that when the 2dft performs a transition to the right, four things can happen (only 2 and 3 are mutually exclusive):

1. A data value from a previous position was used for the last time and should be discarded
2. The current data value has been used earlier and will not be used later (and should be discarded)
3. The current data value may be used later and was not used before (and thus should be stored)
4. A data value from a next position is first used (and thus should be stored)

The challenging part is the case (4), as we would need to fetch the data value which we suddenly need to track. The new value is easily fetched through an MSO jump (to the right) which is a feature introduced by [9] allowing to jump to a position in the input specified by an MSO formula. In turn this jump is implemented (thanks to the look-around carried by the input word) as a one-way automaton that goes to the right until it reaches the position where the data

value is, and a one-way automaton that goes (left) from that position back to the original position. The challenge is to be able to return to the current position. Thanks to our definition, we can also describe an MSO jump that allows the return: if we had to fetch a new data value, it is because it was *first* used at the position we want to jump back to. Such a position can easily be expressed uniquely with an MSO formula from the position we fetched our data value. We remark that we cannot fetch data values on a per-needed basis (an MSO jump to the position where the data is, is possible, but going back with an MSO-jump is not), which indicates we need data variables.

In the 2dft, any transition for which case (4) happens (this information is contained in the look-around) is replaced by two automata that go fetch (and back) that newly needed data value.

Finally we present how this conversion should work on our example. We need to consider 1-types. Θ_1 is 2^Σ: each characterizing exactly which letters are present in the word. This means hundreds of data variables, but at any point for this transformation, no more than 2 data values will be stored. So long as we read a's we should not have fetched the data value of the following #-labeled position. When a b is read, we fetch that data value and then we can return back to our original position: it is the first position (after the last #) in the word that contains a b.

4 One-way transducers

4.1 Streaming string transducers with data variables and parameters

We first define sstvp, i.e. streaming string transducers with data variables and data parameters. They have the features of streaming string transducers [1,2] extended with *data variables* and *data parameters*. Notice that in contrast to the streaming data-string transducers from [2] sstvp can not compare data values with each other.

Intuitively, sstvp read deterministically data words and compute an output data word. They are equipped with data variables which store data values, parameters which are placeholders for data values and data word variables containing data words which in addition to data values can also contain data parameters. These data parameters can be replaced by data values subsequently.

Definition 3. *A sstvp is a tuple* $(\Sigma, \Delta, \Gamma, Q, X, V, P, q_0, v_0, \delta, \Omega)$ *where:*

- Σ *and* Γ *are respectively the input and output alphabets,*
- Δ *is the (infinite) data domain,*
- Q *is the finite set of states and* $q_0 \in Q$ *the initial state,*
- X *is the finite set of data word variables,*
- V *is the finite set of data variables with a designated variable* $curr \in V$*,*
- P *is the finite set of data parameters* ($P \cap \Delta = \emptyset$)*,*
- $v_0 : X \to (\Gamma \times P)^*$ *is a function representing the initial valuation of the data word variables.*

- δ *is a (deterministic) transition function:* $\delta(q, \sigma) = (q', \mu_V, \mu_X, \mu_P)$ *where:*
  - $\mu_V : (V \setminus \{curr\}) \to V$ *is the update function of data variables,*
  - $\mu_X : X \to (X \cup (\Gamma \times (V \cup P)))^*$*, is the update function of data word variables,*
  - $\mu_P : P \times X \to P \cup V$ *is the parameter assignment function (dependent on the data word variable).*
- $\Omega : Q \to ((\Gamma \times V) \cup X)^*$ *is the partial output function.*

The streaming string transducers of [1,2] were defined by restricting updates to be *copyless*, i.e. each data word variable can appear only once in an update μ_X. Here, we relax this syntactic restriction along the lines of [5] by considering only 1-bounded sstvp's: informally, at any position the content of some data word variable may only occur once in the output. This allows to duplicate the value of some data word variable in two distinct data word variables, but the value of these variables can not be later combined. It is clear that this condition can be checked and a 1-bounded sstvp can be transformed into a syntactically copyless sstvp one [5].

Now, we define the semantics of sstvp. A valuation of data variables β_V for an sstvp is a partial function assigning data values to data variables. A valuation of data word variables β_X is a function assigning words over $\Gamma \times (\Delta \cup P)$ to data word variables. Then, a *configuration* of an sstvp consists of a control state and a valuation of data and word variables (β_V, β_X). The initial configuration is (q_0, β_V^0, v_0), where β_V^0 is the empty function. When processing a position i in the input word in some state q, first *curr* is set to the data value at that position in the input, then the data word variables are updated according to μ_X, then the data words contained in data word variables are substituted according to μ_P and finally data variables are updated according to μ_V.

Formally, if $\delta(q, a) = (q', \mu_V, \mu_X, \mu_P)$, then from (q, β_V, β_X) at position i with a letter (a, d) one goes to $(q', \beta_V'', \beta_X''')$ where:

- $\beta_V'' = \beta_V' \cdot \mu_V$, where $\beta_V' = \beta_V[curr \mapsto d]$.
- $\beta_X' = \beta_X \cdot \mu_X$
  $\beta_X''(x) = \beta_X'(x)[v \leftarrow \beta_V'(v)]_{v \in V}$
  $\beta_X'''(x) = \beta_X''(x) \left[p \leftarrow \begin{cases} \mu_P(x,p) \text{ if } \mu_P(x,p) \in P \\ \beta_V'(\mu_P(x,p)) \text{ if } \mu_P(x,p) \in V \end{cases} \right]$

For each two data word variables x, x', we say that x at position i flows to x' at position $i+1$ if $x \in \mu_X(x')$. The notion of flow can be easily extended by transitivity, the copylessness restriction forbids that the value of some data word variable at some position i flows more than once to some data word variable at position $j > i$. When reaching the end of the input word in a configuration (q, β), a sstvp produces $\beta(\Omega(q))$ if $\Omega(q)$ is defined. Then, naturally a sstvp S defines a transduction from words in $\Sigma \times \Delta$ to words in $\Gamma \times \Delta$.

The sstvp for our running example is given in Figure 1. All data word variables are initialized with the empty word. By convention, a variable which is not explicitly updated is unchanged. We omit these updates for readability.

Theorem 6. *Equivalence of two sstvp is decidable.*

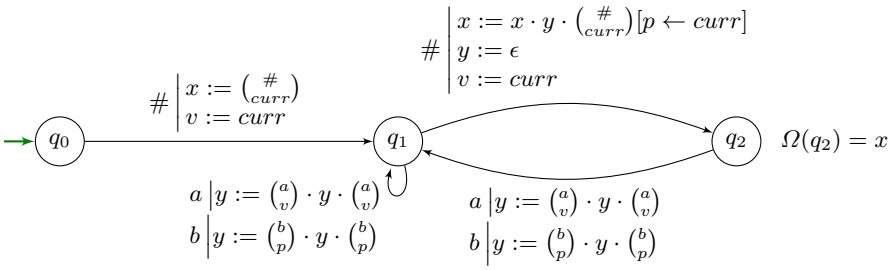


Fig. 1. The sstvp for the running example

To prove this theorem we can generalize the proof of decidability of equivalence of SST [2], a reduction to reachability in a non-deterministic one-counter machine. Given two transducers we choose non-deterministically an input string, and one conflicting position in each of the two images (of the transducers): either they are labeled by different letters, or with attached data value originating from two distinct positions in the input word. We keep track in the counter of the difference between the number of produced symbols which will be part of each output before the corresponding conflicting position. Therefore, if the counter reaches 0 at the last letter of the input, the two transducers are different.

We call SSTVP the class of all data word transductions definable by a sstvp.

4.2 From two-way to one-way transducers

Theorem 7. *2DFTV is included in SSTVP.*

Proof. (Sketch) We use ideas of [1] (based itself on Shepherdson's translation from 2DFA to DFA [10]) where two-way transducers are translated into streaming string transducers. As they translate two-way transducers to copyless streaming string transducers they have to go through an intermediate model called heap-based transducers. Since we relax the copylessness restriction to 1-boundedness we can directly translate 2dftv to sstvp. Furthermore, we have to take care of the contents of data variables of the 2dftv. For that purpose we use data variables and data parameters of the sstvp.

Since an sstvp does only one left-to-right pass on the input word, we cannot revisit any position. As we process a position we need to store all relevant information about that position possibly being later reprocessed by the two-way transducer. The two-way transducer may process a position multiple times (each time in a different state) each time with a different valuation of data variables and producing some different word: for each state, we need to store in an appropriate data word variable the corresponding production, the valuation of data variables being abstracted with data parameters. Notice that not all these data word variables will be used in the output. Given a 2dftv $A = (\Sigma, \Gamma, \Delta, Q, q_0, F, V, \mu, \delta)$, over which we assume all accepting runs end on the last symbol, we define an sstvp $B = (\Sigma, \Gamma, \Delta, Q', X, V', P, q'_0, v_0, \delta', \Omega)$ as follows:

- $Q' = Q \times [Q \to (Q \times 2^V)]$
  A state of the one-way transducer consists of a state of the two-way transducer and a partial mapping from states to a pair of a state and a set of variables. As a position $i+1$ is processed, the state of B contains the following information: in which state A first reaches position i and for each state q of A what would be the state of A when it reaches for the first time position $i+1$ had it started processing position i from state q: this part is the standard Shepherdson's construction. The function is partial, as from position i from some states A might never reach position $i+1$ (getting stuck).
  We remark that along the subrun from position i (in state q) to position $i+1$, the A might store some data values in some data variables. The set of data variables denotes the set of data variables the two-way transducer has updated along that run.
- $X = x_l \cup \{x_q \mid q \in Q\}$
  At position i, variable x_l will store the word produced by A until it first reaches position i. Variable x_q will store the word produced from position i in state q until position $i+1$ is first reached.
- $V' = V \cup \{v_q \mid v \in V, q \in Q\}$
  At position $i+1$, data variable v will contain the value of variable v of A as it first reaches position $i+1$. Assume that B reaches position i in some state (q, f) with $f(q') = (q'', W)$, and $v \in W$. Then variable $v_{q'}$ will contain the last value stored in v when A processes from position i in state q' until it first reaches position $i+1$.
- $P = \{p_{v,q} \mid v \in V, q \in Q\}$
  At position i, parameter $p_{v,q}$ will be present only in data word variable x_q, representing that along the run of A the data value from data variable v at position i in state q was output before $i+1$ was first reached. Such a symbol needs to be present in x_q, but the data value is not yet known, hence it is abstracted by the data parameter $p_{v,q}$.

It is then easy to see how to define q'_0 and δ' so as to preserve these invariants. As B can not see $\dashv$, B must maintain the possible output in an extra variable, where it is supposed that the next symbol would be $\dashv$.

We now detail an example (see Fig. 2) so as to give an intuition how $\delta'(q, \sigma)$ is built: we will specifically focus on the value of x_{q_1}. We denote f the second component of q and we assume that $f(q_2) = (q_3, \{v_1, v_2\})$, $f(q_4) = (q_5, \{v_2, v_3\})$. Furthermore, we assume that in A, $\delta(q_1, \sigma) = (q_2, -1, (\gamma, v_2))$ and $\delta(q_3, \sigma) = (q_4, -1, (\gamma', v_2)(\gamma'', v_3))$ and finally that $\delta(q_5, \sigma) = (q_6, +1, (\gamma''', v_2))$. Also reading σ in q_1 and q_3 assigns the current data value to v_1 (i.e. $\mu(q_1, \sigma, v_1) = \mu(q_3, \sigma, v_1) = curr$), other data variables are not modified (i.e. $\mu(q_1, \sigma, v_i) = \mu(q_3, \sigma, v_i) = v_i$).

By the aforementioned invariants, from state q_1, A will first reach the following position in state q_6 (from the σ-labeled position in state q_1, it first goes left, reaches it again in state q_3, goes left again, arrives in state q_5 and then moves to the right in state q_6).

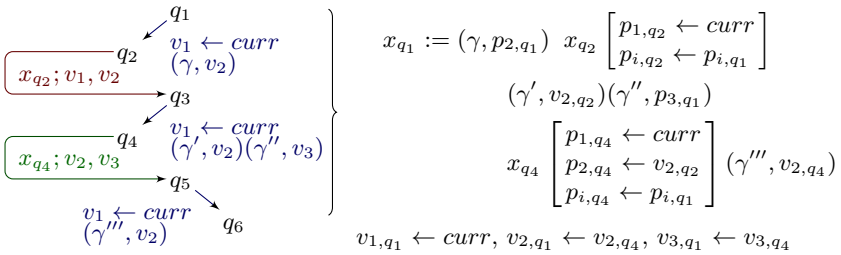


Fig. 2. An example to illustrate the transformation from A to B.

If we abstract the data values, the content of the data word variable x_{q_1} will thus be $(\gamma, ?)x_{q_2}(\gamma', ?)(\gamma'', ?)x_{q_4}(\gamma''', ?)$. Now we detail data attached to the produced letters, and the parameter assignments in the data word variables:

γ will be given the data parameter p_{2,q_1}.

In x_{q_2}: since a data value is assigned to v_1 between q_1 and q_2, p_{1,q_1} should be substituted by that data value (which is $curr$) in x_{q_2}. Other parameters in x_{q_2} (which are all of the form p_{i,q_2}) are substituted by the corresponding p_{i,q_1}.

γ' will be given the data value v_{2,q_2} and (because v_3 has not been assigned a data value since q_1) γ'' will be assigned the data parameter p_{3,q_1}.

In x_{q_4}: as a data value was assigned to v_2 between q_2 to q_3, parameter p_{2,q_4} will be substituted by that value i.e. v_{2,q_2}; parameter p_{1,q_4} will be substituted by $curr$ and all other parameters (which are of the form p_{i,q_4}) will be assigned the corresponding data parameters p_{i,q_1}.

γ''' should be assigned data value v_{2,q_4}.

Therefore by reading a σ in B, we reach a state whose second component maps q_1 to $(q_6, \{v_1, v_2, v_3\})$, $v_{1,q_1} \leftarrow curr$, $v_{2,q_1} \leftarrow v_{2,q_4}$, $v_{3,q_1} \leftarrow v_{3,q_4}$.

4.3 From one-way transducers to MSO

In order to conclude that the three models of data word transformations are equivalent, it remains to show that our MSO transductions with MSO origin information capture all transformations defined by the one-way model.

Theorem 8. *SSTVP is included in MSOT+O.*

The proof is very similar to that of encoding finite state automata in MSO. Usually to show that MSO captures string transformations defined by a one-way model one defines an output graph with Γ-labeled edges and ϵ-edges. We directly give a proof that builds a (string) graph whose nodes are Γ-labeled.

Given an sstvp S we fix the set of copies C as the set of occurrences of symbols of Γ in the variable update function.

Since S is deterministic, we will write an MSO sentence φ that characterizes a run of a word in S. This formula will be of the form $\exists X_1, \dots X_n \psi$, such that given a word w (in the domain of the transformation), there exists a unique assignment of the X_i such that ψ holds. These second order variables consist of:

- X_q for $q \in Q$: position $i \in X_q$ iff processing position i yielded state q.
- X_r for every word variable r: position $i \in X_r$ iff the content of variable r will flow in the output
- X_{r_1,r_2} for every pair of distinct word variables r_1, r_2: position $i \in X_{r_1,r_2}$ iff the content of variable r_1 will flow in the output before the content of the variable r_2 that will also flow in the output.

Our sequential machine model allows easily to write such a formula ψ. With the formula ψ, we can write formulas φ_{indom}, $(\varphi^c_{dom})_{c\in C}$, $(\varphi^c_\gamma)_{c\in C}$, and $(\varphi^{c,d}_<)_{c,d\in C}$. We remark that second order variables X_{r_1,r_2} have a unique valid assignment because of the (semantic) copylessness of sstvp. These variables are typically used to define $\varphi^{c,d}_<$.

To hint how to build formula $\varphi^c_{orig}(x, y)$ we state the following simple lemma about runs of sstvps.

Lemma 2. *Given an sstvp S, an input word w and position x that produces a symbol $\gamma \in \Gamma$ that will be part of the output.*

- *Either γ is produced with a data variable (namely v):*
  In this case, there exists a unique position $y \leq x$ where the data value curr was stored in some data variable and that data variable flows to data variable v at position x.
- *or γ is produced with a data parameter (namely p):*
  In this case, there exists a unique position z such that the data parameter attached to γ is some p_m at position z and that p_m is assigned a variable v_m (or curr) at position z. There exists a unique position $y \leq q$ such that at position y the data value curr was put in some data variable, which flows to a data variable v_m at position z.

The notion of "flow" is easily expressed with ψ and second order existential quantification. The copyless semantics of sstvps ensures that to each (output) symbols, exactly one data value (or equivalently a unique position from the input word) is assigned to. This allows to build MSO formulas φ^c_{orig} that have the desired functional property.

5 Conclusion

Finite string transformation have been generalized to infinite string transformations [5] and tree transformations [3,4]. It would be interesting to extend our results to these settings by adding data values and defining transformations via origin information. Furthermore, it would be interesting to study the pre-post condition checking problem along the lines of [2], i.e. the problem to check that given a transducer is it the case that each input satisfying a pre-condition defined via some automata-model is transformed into an output satisfying a similarly defined post-condition.

References

1. Alur, R., Černý, P.: Expressiveness of streaming string transducers. In: FSTTCS. vol. 8, pp. 1–12 (2010)
2. Alur, R., Černý, P.: Streaming transducers for algorithmic verification of single-pass list-processing programs. In: POPL. pp. 599–610 (2011)
3. Alur, R., D'Antoni, L.: Streaming tree transducers. In: Automata, Languages, and Programming - 39th International Colloquium, ICALP. vol. 7392, pp. 42–53. Springer (2012)
4. Alur, R., Durand-Gasselin, A., Trivedi, A.: From monadic second-order definable string transformations to transducers. In: 28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS. pp. 458–467. IEEE Computer Society (2013)
5. Alur, R., Filiot, E., Trivedi, A.: Regular transformations of infinite strings. In: Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS. pp. 65–74. IEEE Computer Society (2012)
6. Bojanczyk, M.: Transducers with origin information. In: Automata, Languages, and Programming - 41st International Colloquium, ICALP Proceedings, Part II. vol. 8573, pp. 26–37. Springer (2014)
7. Chytil, M., Jákl, V.: Serial composition of 2-way finite-state transducers and simple programs on strings. In: Proceedings of the Fourth Colloquium on Automata, Languages and Programming. pp. 135–147. Springer-Verlag, London, UK, UK (1977)
8. Courcelle, B.: Monadic second-order definable graph transductions: A survey. Theor. Comput. Sci. 126(1), 53–75 (1994)
9. Engelfriet, J., Hoogeboom, H.J.: MSO definable string transductions and two-way finite-state transducers. ACM Trans. Comput. Logic 2, 216–254 (2001)
10. Shepherdson, J.C.: The reduction of two-way automata to one-way automata. IBM J. Res. Dev. 3(2), 198–200 (Apr 1959)
11. Thomas, W.: Ehrenfeucht games, the composition method, and the monadic theory of ordinal words. In: In Structures in Logic and Computer Science: A Selection of Essays in Honor of A. Ehrenfeucht, Lecture. pp. 118–143. Springer-Verlag (1997)

Discrete Parameters in Petri Nets*

Nicolas David[1], Claude Jard[1], Didier Lime[2], and Olivier H. Roux[2]

[1] University of Nantes, LINA
nicolas.david1@univ-nantes.fr
claude.jard@univ-nantes.fr
[2] École Centrale de Nantes, IRCCyN
didier.lime@ec-nantes.fr
olivier-h.roux@irccyn.ec-nantes.fr

Abstract. With the aim of significantly increasing the modeling capability of Petri nets, we suggest that models involve parameters to represent the weights of arcs, or the number of tokens in places. We consider the property of coverability of markings. Two general questions arise: "Is there a parameter value for which the property is satisfied?" and "Does the property hold for all possible values of the parameters?". We show that these issues are undecidable in the general case. Therefore, we also define subclasses of parameterised networks, depending on whether the parameters are used on places, input or output arcs of transitions. For some subclasses, we prove that certain problems become decidable, making these subclasses more usable in practice.

Keywords: Petri net, Parameters, Coverability

1 Introduction

The introduction of parameters in models aims to improve genericity. It also allows the designer to leave unspecified aspects, such as those related to the modeling of the environment. This increase in modeling power usually results in greater complexity in the analysis and verification of the model. Beyond verification of properties, the existence of parameters opens the way to very relevant issues in design, such as the computation of the parameters values ensuring satisfaction of the expected properties.

We chose to explore the subject on concurrent models whose archetype is that of Petri nets. We consider discrete parameterisation of markings (the number of tokens in the places of the net) or weight of arcs connecting the input or output places to transitions. We call these Petri nets parameterised nets or PPNs.

We consider the general properties of coverability and, to a lesser extent, reachability (that are often the basis for the verification of more specific properties).

First issues are:

*Work partially supported by ANR project PACS (ANR-14-CE28-0002) and Pays de la Loire research project AFSEC.

– Is there a value of the parameters such that the property is satisfied?
– Is the property satisfied for all possible values of the parameters?

Given the modeling power offered by PPNs, we first study the decidability of these issues. Since in the general case, they are undecidable, we then examine decidable subclasses.

Related work There is not much work on Petri nets with parameters. One example is regular model checking [3] for algorithmic verification of several classes of infinite-state systems whose configurations can be modeled as words over a finite alphabet. The main idea is to use regular languages as the representation of sets of configurations, and finite-state transducers to describe transition relations. This is only possible for particular examples including parameterised systems consisting of an arbitrary number of homogeneous finite-state processes connected in a regular topology, and systems that operate on linear data structures. Parameters are also introduced in models such as predicate Petri nets [8], in the aim to have more concise models, in particular to take into account symmetries in the model [4]. Domains of values are generally finite. Parameterised verification on timed systems has also been studied in several papers since its introduction by Alur et al. in [1]. Parameterisation of time uses continuous parameters. In this paper, we focus on discrete parameters on untimed Petri nets.

The remainder of the paper is structured as follows: Section 2 re-visites the semantics of Petri Nets and gives the basic definitions related to the formalism of Parametric Petri Nets. Section 3 presents the undecidability results. Section 4 introduces subclasses of our parameterised models. Section 5 answers decidability results over those subclasses and underlines issues encountered with reachability. Section 6 concludes and points to future work.

2 Definitions

Notations

$\mathbb{N}$ is the set of natural numbers. $\mathbb{N}^*$ is the set of positive natural numbers and $\mathbb{N}_\omega$ is the classic union $\mathbb{N} \cup \{\omega\}$ where for each $n \in \mathbb{N}$, $n + \omega = \omega$, $\omega - n = \omega$, $n < \omega$ and $\omega \leq \omega$. $\mathbb{Z}$ is the set of integers. Let X be a finite set. 2^X denotes the powerset of X and $|X|$ the size of X. Let $V \subseteq \mathbb{N}$, a V-valuation for X is a function from X to V. We therefore denote $V^X$the set of V-valuations on X. Given an alphabet Σ, we denote as Σ_ϵ the union $\Sigma \cup \{\epsilon\}$ where ϵ is the silent action. Given a set X, let $k \in \mathbb{Z}$ and $x \in X$, we define a linear expression on X by the following grammar: $\lambda ::= k \mid k * x \mid \lambda + \lambda$. Given a linear expression λ on X and a $\mathbb{N}$-valuation ν for X, $\nu(\lambda)$ is the integer obtained when replacing each element x in X from λ, by the corresponding value $\nu(x)$.

2.1 Petri Nets and Marked Petri Nets

Definition 1 (Petri Net) *A Petri Net is a 4-tuple* $\mathcal{N} = (P, T, Pre, Post)$ *where* P *is a finite set of places,* T *is a finite set of transitions,* Pre *and*

$Post \in \mathbb{N}^{|P|\times|T|}$ are the backward and forward incidence matrices, such that $Pre(p,t) = n$ with $n > 0$ when there is an arc from place p to transition t with weight n and $Post(p,t) = n$ with $n > 0$ when there is an arc from transition t to place p with weight n.

Given a Petri Net $\mathcal{N} = (P, T, Pre, Post)$, we denote $Pre(\bullet, t)$ (also written ${}^\bullet t$) as the vector $(Pre(p_1,t), Pre(p_2,t), ..., Pre(p_{|P|},t))$ *i.e.* the t^{th} column of the matrix Pre. The same notation is used for $Post(\bullet, t)$ (or $t^\bullet$).

Definition 2 (Marking) *A marking of a Petri Net $\mathcal{N} = (P, T, Pre, Post)$ is a vector $m \in \mathbb{N}^{|P|}$.*

If $m \in \mathbb{N}^{|P|}$ is a marking, $m(p_i)$ is the number of tokens in place p_i. We can define a partial order over markings.

Definition 3 (Partial Order) *Let $\mathcal{N}$ be a Petri Net such that $\mathcal{N} = (P, T, Pre, Post)$, let m and m' be two markings of $\mathcal{N}$. We define $\leq$ as a binary relation such that $\leq$ is a subset of $\mathbb{N}^{|P|} \times \mathbb{N}^{|P|}$ defined by:*

$$m \leq m' \Leftrightarrow \forall p \in P, m(p) \leq m'(p) \tag{1}$$

Definition 4 (Marked Petri Net) *A marked Petri Net (PN) is a couple $\mathcal{S} = (\mathcal{N}, m_0)$ where $\mathcal{N}$ is a Petri Net and m_0 is a marking of $\mathcal{N}$ called the initial marking of the system.*

An example of *marked Petri Net* is given in Figure 1.

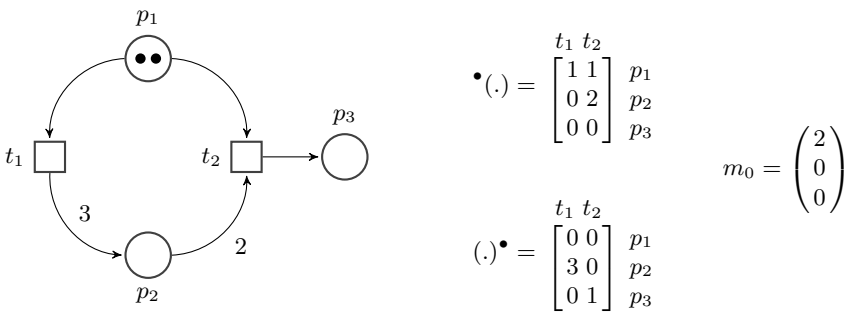


Fig. 1. A Marked Petri Net

2.2 Operational Semantics

Augmenting Petri Nets with markings leads to the notion of enabled-transitions and firing of transitions. Given a marked Petri Net $\mathcal{S}$, a transition $t \in T$ is said *enabled by a marking m* when $m \geq Pre(\bullet, t)$.

Definition 5 (PN Semantics) *The semantics of PN is a transition system* $\mathcal{S}_{\mathcal{T}} = (Q, q_0, \rightarrow)$ *where,* $Q = \mathbb{N}^{|P|}$, $q_0 = m_0$, $\rightarrow \in Q \times T \times Q$ *such that,*

$$m \xrightarrow{t_i} m' \Leftrightarrow \begin{cases} m \geq {}^\bullet t_i \\ m' = m - {}^\bullet t_i + t_i^\bullet \end{cases} \tag{2}$$

This relation holds for sequences of transitions:

- $m \xrightarrow{w} m'$ if w is the empty word and $m = m'$
- $m \xrightarrow{wt} m'$ if $\exists m'', m \xrightarrow{w} m'' \wedge m'' \xrightarrow{t} m'$ where $w \in T^*$ and $t \in T$.

Definition 6 (Reachability set) *Given a PN,* $\mathcal{S} = (\mathcal{N}, m_0)$*, the reachability set of* $\mathcal{S}$*,* $RS(\mathcal{S})$ *is the set of all reachable markings of* $\mathcal{S}$ *i.e.* $RS(\mathcal{S}) = \{m \mid \exists w \in T^*, m_0 \xrightarrow{w} m\}$

2.3 Parametric Petri Nets

We would like to use less rigid modeling in order to model systems where some data are not known *a priori*. Therefore, in this subsection, we extend the previous definitions by adding a set of parameters Par. Working with Petri nets and discrete parameters leads to consider two main situations: the first one involves parameters on markings, by replacing the number of tokens in some places by parameters, the second one involves parameters as weights. The same parameter can be used in both situations. Using parameters on markings can be easily understood as modeling an unfixed amount of resources that one may want to optimise. Let us consider a concrete example to illustrate parameterised weights. In a production line, we consider two operations: first, to supply raw material, we need to unpack some boxes containing an amount λ_1 of resources, as depicted in Figure 2, and at the end, we need to pack end products in boxes of capacity λ_2, as in Figure 3. This is part of a whole packaging process that one may want to optimise. The level of abstraction induced by parameters permits to leave those values unspecified in order to perform an early analysis.

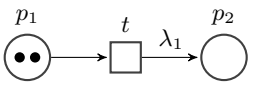


Fig. 2. Unpacking raw material

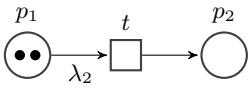
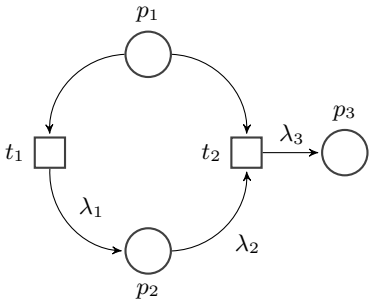


Fig. 3. Packing end products

Definition 7 (Parametric Petri Net) *A parametric Petri Net,* $\mathcal{NP}$ *is a* 5*-tuple* $\mathcal{NP} = (P, T, Pre, Post, Par)$ *such that* P *is a finite set of places of* $\mathcal{NP}$*,* T *is a finite set of transitions of* $\mathcal{NP}$*,* Par *is a finite set of parameters of* $\mathcal{NP}$*,* Pre *and* $Post \in (\mathbb{N} \cup Par)^{|P| \times |T|}$



$$\bullet(.) = \begin{array}{c} \begin{matrix} t_1 & t_2 \end{matrix} \\ \begin{bmatrix} 1 & 1 \\ 0 & \lambda_2 \\ 0 & 0 \end{bmatrix} \end{array} \begin{matrix} p_1 \\ p_2 \\ p_3 \end{matrix}$$

$$(.)^\bullet = \begin{array}{c} \begin{matrix} t_1 & t_2 \end{matrix} \\ \begin{bmatrix} 0 & 0 \\ \lambda_1 & 0 \\ 0 & \lambda_3 \end{bmatrix} \end{array} \begin{matrix} p_1 \\ p_2 \\ p_3 \end{matrix}$$

Fig. 4. A Parametric Petri Net

Intuitively, a parametric Petri net is a Petri net where the number of tokens involved in a transition is parameterised as depicted in Figure 4.

Definition 8 (Parametric marking) *Given a parametric Petri Net* $\mathcal{NP} = (P, T, Pre, Post, Par)$*, a parametric marking is a* $|P|$*-dimensional vector* μ *of linear expressions on* $\mathbb{N} \cup Par$*.*

Modeling with parameters means using parameters over weights and markings rather than setting numeric values everywhere. Therefore we may also use a parametric initial marking.

Definition 9 (parametric PN or PPN) *A parametric marked Petri Net (PPN) is a couple,* $\mathcal{SP} = (\mathcal{NP}, \mu_0)$ *where* $\mathcal{NP}$ *is a Parametric Petri Net and* μ_0 *is the parametric initial marking of* $\mathcal{NP}$*.*

PPNs can be used to design systems where some parts have not been analysed or where we need to keep flexibility. We now need to define a way to instantiate classic Petri nets from our parametric marked Petri Nets, in order to define a semantics.

Definition 10 (Parametric Semantics) *Let* $\mathcal{SP} = (P, T, Pre, Post, Par, \mu_0)$ *be a PPN, we consider the set of valuations* $\mathbb{N}^{Par}$*. Let* $\nu \in \mathbb{N}^{Par}$*, we define* $\nu(\mathcal{SP})$ *as the PN obtained from* $\mathcal{SP}$ *by replacing each parameter* $\lambda \in Par$ *by* $\nu(\lambda)$*, its valuation by* ν*, i.e.* $\nu(\mathcal{SP}) = (P, T, Pre', Post', m_0)$ *where* $\forall i \in [\![1, |P|]\!], \forall j \in [\![1, |T|]\!]$*,*

$$Pre'(i,j) = \begin{cases} Pre(i,j) & \text{if } \ Pre(i,j) \in \mathbb{N} \\ \nu(Pre(i,j)) & \text{if } Pre(i,j) \in Par \end{cases} \tag{3}$$

$$Post'(i,j) = \begin{cases} Post(i,j) & \text{if } \ Post(i,j) \in \mathbb{N} \\ \nu(Post(i,j)) & \text{if } Post(i,j) \in Par \end{cases} \tag{4}$$

$$m_0(i) = \begin{cases} \mu_0(i) & \text{if } \qquad \mu_0(i) \in \mathbb{N} \\ \nu(\mu_0(i)) & \text{if } \mu_0(i) \text{ is a linear expression on } Par \end{cases} \tag{5}$$

A marked Petri Net is an *instance* of a parametric marked Petri net.

2.4 Parametric problems

We can define several interesting parametric problems on PPNs. In fact, the behaviour of a PPN is described by the behaviours of all the PNs obtained by considering all possible valuations of the parameters. It seems therefore obvious to ask, in a first time, if *there exists valuations for the parameters such that a property holds for the corresponding instance* and its dual, *i.e.*, if *every instance of the parametric marked Petri Net satisfies the property*. Given a class of problem $\mathcal{P}$ (coverability, reachability,...), $\mathcal{SP}$ a PPN and ϕ is an instance of $\mathcal{P}$, parameterised problems are written as follows:

Definition 11 ($\mathcal{P}$-Existence problem) *($\mathscr{E}$-$\mathcal{P}$): Is there a valuation $\nu \in \mathbb{N}^{Par}$ s.t. $\nu(\mathcal{SP})$ satisfies the property ϕ ?*

Definition 12 ($\mathcal{P}$-Universality problem) *($\mathscr{U}$-$\mathcal{P}$): Does $\nu(\mathcal{SP})$ satisfies the property ϕ for each $\nu \in \mathbb{N}^{Par}$?*

This paper focuses on *reachability* and *coverability* issues.

Definition 13 (Reachability) *Let $\mathcal{S} = (\mathcal{N}, m_0) = (P, T, Pre, Post, m_0)$ and m a marking of $\mathcal{S}$, $\mathcal{S}$ reaches m iff $m \in RS(\mathcal{S})$.*

Definition 14 (Coverability) *Let $\mathcal{S} = (\mathcal{N}, m_0) = (P, T, Pre, Post, m_0)$ and m a marking of $\mathcal{S}$, $\mathcal{S}$ covers m if there exists a reachable marking m' of $\mathcal{S}$ such that m' is greater or equal to m i.e.*

$$\exists m' \in \mathcal{RS}(\mathcal{S}) s.t.\ \forall p \in P, m'(p) \geq m(p) \tag{6}$$

We recall that reachability [9] and coverability [7] are decidable on classic Petri nets. In the context of parametric Petri nets, coverability leads to two main problems presented previously, that is to say: the *existence problem*, written ($\mathscr{E}$-cov) and the *universal problem*, written ($\mathscr{U}$-cov). For instance, ($\mathscr{U}$-cov) asks: *"Does each valuation of the parameters implies that the valuation of the parametric P/T net system covers m ?" i.e.*

$$m \text{ is } \mathscr{U}\text{-coverable in } \mathcal{SP} \Leftrightarrow \begin{cases} \forall \nu \in \mathbb{N}^{Par}, \ \exists m' \in \mathcal{RS}(\nu(\mathcal{SP})) \\ \qquad\qquad \text{s.t. } m' \geq m \end{cases} \tag{7}$$

We can similarly define $\mathscr{E}$-reach and $\mathscr{U}$-reach for parameterised reachability.

3 Undecidability Results for the General Case

In their paper suming-up results of decidability for reset-nets and corresponding subclasses, Dufourd, Finkel and Schnoebelen noticed that *"Reachability is known to become undecidable as soon as the power of Petri nets is increased"* [5], for instance, adding *reset arcs* [2] or *inhibitor arcs* [6] makes reachability undecidable. In this section, we focus on showing that adding parameters to PN leads to undecidability. More specifically, ($\mathscr{U}$-cov) and ($\mathscr{E}$-cov) are undecidable on PPNs.

As we will proceed by reduction to the halting problem (and counter boundedness problem) for counter machines to answer our problem, we first recall some definitions. A 2-Counters Machine has a pointer and a tape which contains finite number of instructions in three types: *increment*, *decrement* and *zero-test*. The pointer reads the tape to execute increment or decrement instructions sequentially. When the pointer reaches a *zero-test* instruction, then it will jump to a certain position on the tape and continue. Formally, it consists of two counters c_1, c_2, a set of states $P = \{p_0, ...p_m\}$, a terminal state labelled *halt* and a finite list of instructions $l_1, ..., l_s$ among the following list:

- increment: increase c_k by one and go to next state, where $k \in \{1, 2\}$
- decrement: decrease c_k by one and go to next state, where $k \in \{1, 2\}$
- zero-test: if $c_k = 0$ go to state p_j else go to state p_l, where $p_j, p_l \in P \cup \{halt\}$ and $k \in \{1, 2\}$

We can assume without restriction that the counters are non negative integers *i.e.* that the machine is well-formed in the sense that a *decrement instruction* is guarded by a *zero-test* and that the counters are initialised to zero. It is well known that the *halting problem* (whether state *halt* is reachable) and the *counters boundedness problem* (whether the counters values stay in a finite set) are both *undecidable* as proved by Minksy [10].

Theorem 1 (Undecidability of $\mathscr{E}$-cov on PPN) *The $\mathscr{E}$-coverability problem for PPN is undecidable* [1].

Proof. We proceed by *reduction from a 2-counters machine*. Given a Minksy 2-counters machine $\mathcal{M}$, we construct a PPN that simulates it, $\mathcal{SP}_{\mathcal{M}}$, as follows.

- Each counter c_i is modeled by two places C_i and $\neg C_i$. The value of the counter is encoded by the number of tokens in C_i.
- For each state p of $P \cup \{halt\}$ a 1-bounded place p is created in the net.
- The instructions of the previous definition are modeled by the transitions and arcs depicted in Figure 5.
- A unique additional place π with an additional transition θ serves to initialise the net. The initial marking is composed of one token in π and one token in the place p corresponding to the initial state p of $\mathcal{M}$.

Initially, only θ can be fired, which leads to the initial configuration of the machine (state p_0 and counters values null), with one token in p_0, no tokens in C_1 and C_2 and a parameterised number of tokens in $\neg C_i$. The value of this parameter will therefore represent the upper bound of the counter over the instructions sequence. We have to verify that each time $m(C_i) + m(\neg C_i) = \lambda$. First we show that $\mathcal{SP}_{\mathcal{M}}$ simulates $\mathcal{M}$ by verifying the behaviour of each instruction:

[1] We can be more accurate by specifying that we need at least 1 parameter used on 6 distinct arcs. The question remains opened for fewer parameterised arcs

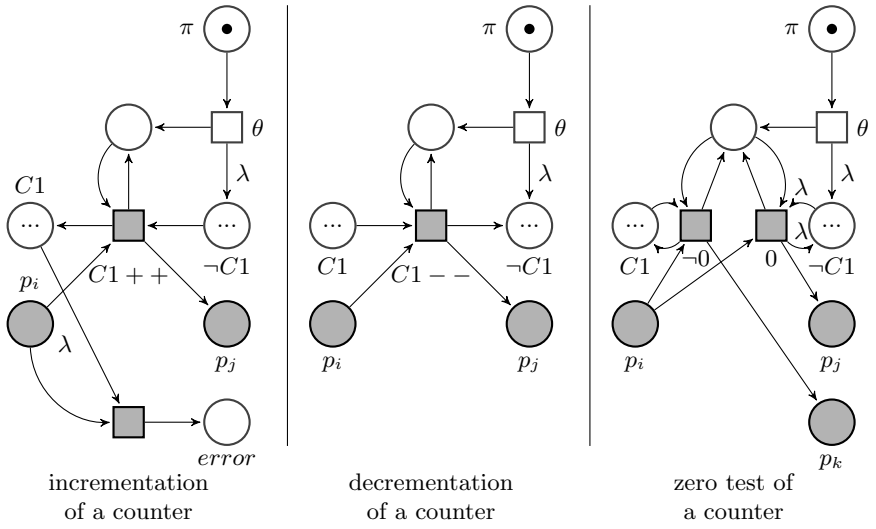


Fig. 5. Modeling a counter with PPN

- *Increment instruction:* As C_i models the counter, the transition $C_i + +$ adds one token in C_i, removes one token from $\neg C_i$ and changes the *current state* by removing the token from p_i and adding a token in p_j. The *error* states is marked iff the incrementation instruction is performed whereas we have already reached the upper bound over the execution. This state will be useful for the second proof.
- *Decrement instruction:* As C_i models the counter, the transition C_i - - removes one token from C_i, adds one token in $\neg C_i$ and changes the *current state* by removing the token from p_i and adding a token in p_j. We recall the machine is well-formed.
- *Zero Test:* As C_i models the counter, and as we know the sum of tokens available in C_i and $\neg C_i$, there is no token in C_i iff there are λ tokens in $\neg C_i$. According to this test the *current state* is updated by removing the token from p_i and adding a token in p_j or p_k. The value of the counter is left unchanged.

℘ Coverability is undecidable :
We will show that given a 2-counters machine $\mathcal{M}$, *(a)* $\mathcal{M}$ halts (it reaches the *halt* state) iff *(b)* there exists a valuation ν such that $\nu(\mathcal{SP}_{\mathcal{M}})$ covers the corresponding p_{halt} place.

- $(a) \Rightarrow (b)$ First, let us assume that $\mathcal{M}$ halts. As $\mathcal{M}$ halts, the execution of the machine is finite. On this execution the two counters are bounded by c_{lim1} and c_{lim2}. Let c_{lim} be the maximum of those two values. Let ν be the valuation such that $\nu(\lambda) = c_{lim}$. By the previous explanation, $\mathcal{SP}_{\mathcal{M}}$ simulates $\mathcal{M}$. Moreover, the valuation ν ensures that $\mathcal{SP}_{\mathcal{M}}$ does not reach

a deadlock state where p_{error} is marked. Therefore, when $\mathcal{M}$ reaches *halt*, $\mathcal{SP}_{\mathcal{M}}$ will add 1 token in p_{halt}. So, a marking where there is one token in p_{halt} is coverable.

- $(b) \Rightarrow (a)$ We proceed by contrapositive. Let us assume that $\mathcal{M}$ does not halt. We want to show that there is no valuation ν such that $\nu(\mathcal{SP}_{\mathcal{M}})$ adds a token in p_{halt}. Let us consider the two following distinct alternatives:
  - If the counters are bounded along the execution, either the value of λ is less than the maximum value of the counters and error will be reached during some increment resulting in a deadlock, or the value of λ is big enough so that error is never marked, but, in this case, then, as the machine does not halt, it means that it does not reach *halt*. So there is no instruction that leads to *halt* in $\mathcal{M}$. Therefore, according to the previous explanation, there is no transition that adds a token in p_{halt}.
  - If at least one counter is not bounded, then for any given valuation ν, we will reach an instruction *inc*(c_i), where i is 1 or 2, and $c_i = \nu(\lambda)$. Therefore, a token will be added in p_{error} leading to a deadlock. So $\mathcal{SP}_{\mathcal{M}}$ will not cover a terminal state.

The undecidability of the halting problem on the 2-counters machine gives the undecidability of the $\mathscr{E}$-coverability problem.

Theorem 2 (Undecidability of $\mathscr{U}$-cov on PPN) *The $\mathscr{U}$-coverability problem for PPN is undecidable.*

Proof. $\mathscr{U}$ Coverability is undecidable:

We proceed by *reduction from a 2-counters machine*. We use the same construction as in the previous proof. We denote m_{error} the marking were $m_{error}(p) = 0$ for each $p \in P$ except $m_{error}(p_{error}) = 1$. We will show that given a 2-counter machine $\mathcal{M}$, *(a)* the counters are unbounded along the instructions sequence of $\mathcal{M}$ (counters boundedness problem) iff *(b)* for each valuation ν, $\nu(\mathcal{SP}_{\mathcal{M}})$ covers the m_{error}.

- $(a) \Rightarrow (b)$ First, let us assume that on a given instruction sequence, one counter of $\mathcal{M}$ is unbounded. By the second alternative considered in the proof for $\mathscr{E}$-cov we proved that for any valuation, a token will be added in p_{error}.
- $(b) \Rightarrow (a)$ Reciprocally, by contrapositive, we want to show that if the counters are bounded, there exists a valuation ν such that $\nu(\mathcal{SP}_{\mathcal{M}})$ does not cover m_{error}. This comes directly from the previous proof. As the counters are bounded along the instructions sequence, we consider a valuation ν such that $\nu(\lambda) = c_{lim}$ where c_{lim} is an upper bound of the values of the counters. By construction, there is no possibilities to add a token in p_{error}, otherwise, it means that $\mathcal{SP}_{\mathcal{M}}$ took an incrementation transition meaning that c_{lim} is not an upper bound.

The undecidability of the counters boundedness problem on the 2-counters machine gives the undecidability of the $\mathscr{U}$-coverability problem.

4 Subclasses of Parametric Petri Nets

4.1 Introducing Subclasses

On the one hand, our parametric model increases the modeling power of Petri nets but on the other hand, using parameters leads to complex models where properties become undecidable. In order to obtain parameterised models that are easier to analyse and therefore can be used in practice, we should reduce the power of modeling. We will therefore introduce some subclasses of the PPN in which we restrict the use of parameters to only markings, which could be used to model arbitrary number of identical processes, to only output arcs, which, we will see, is a bit more general or to only input arcs, which could model synchronizations among arbitrary numbers of identical process, and finally some combinations of those.

The following subclasses have therefore a dual interest. From a modeling point of view, restrict the use of parameters to tokens, output or input can be used to model concrete examples such as respectively processes or synchronisation of a given number of processes. From a theoretical point of view, it is interesting to introduce those subclasses of PPN in a concern of completeness of the study.

Definition 15 (P-parametric PN) *A P-parametric marked Petri Net (P-PPN),* $\mathcal{SP} = (\mathcal{NP}, \mu_0)$ *where* $\mathcal{NP}$ *is a Parametric Petri Net such that* Pre *and* $Post \in \mathbb{N}^{|P|\times|T|}$ *and* μ_0 *is a parametric marking of* $\mathcal{NP}$*.*

A P-PPN is a classic Petri net with a parametric initial marking.

Definition 16 (T-parametric PN) *A T-parametric Petri Net (T-PPN),* $\mathcal{SP} = (\mathcal{NP}, m_0)$ *where* $\mathcal{NP}$ *is a Parametric Petri Net and* m_0 *is a marking of* $\mathcal{NP}$

Intuitively, using parameters on outputs means we will create parametric markings. To complete this study, we can extract a subclass in which parameters involved in the Pre matrix and parameters involved in the Post matrix correspond to disjoints subsets of parameters. *i.e.* $par(Pre) \cap par(Post) = \emptyset$ where *par* is the application that maps to the set of parameters involved in a matrix (or a vector). We call this subclass *distinctT-PPN*[2]. We can even refine the subclass of distinctT-PPN by considering the two distinct classes of *Pre-T-parametric PPN* (preT-PPN), where $Post \in \mathbb{N}^{|P|}$ and *Post-T-parametric PPN* (postT-PPN), where $Pre \in \mathbb{N}^{|P|}$.

As we introduced several subclasses, it is interesting to study whether one of this subclass is more expressive than the other. We will show that P-PPN and postT-PPN are related. Therefore, we introduce here some useful definitions. Our translations add *silent actions* that detail the beahviour of the Petri nets. Therefore, we introduce a labelling function Λ from the set of transitions T to

[2]Studying the undecidability proof, it is relevant to think that using different parameters for the input and the output would reduce the modeling power.

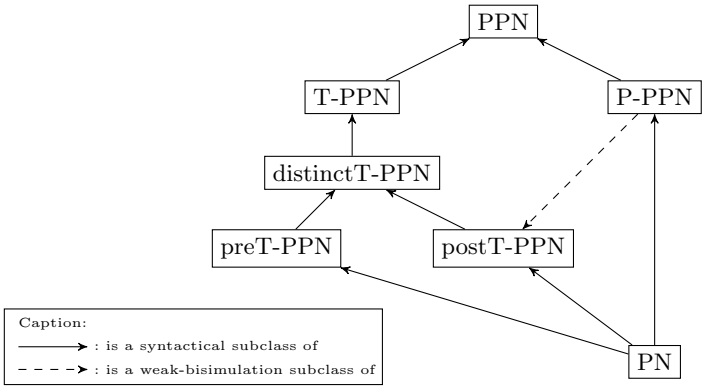


Fig. 6. Subclasses of PPN

Σ_ϵ, $\Lambda : T \to \Sigma_\epsilon$, such that $\Sigma_\epsilon \subseteq T \cup \{\epsilon\}$ and $\Lambda(t_i)$ equals either t_i or ϵ. We extend the previous definitions by using $m \xrightarrow{t} m'$ or $m \xrightarrow{\Lambda(t)} m'$ depending on the context[3]. For instance, $m \xrightarrow{\epsilon^*} m'$ means that m leads to m' by using zero or more internal ϵ-transitions. Given two markings m and m' we write:

$$m \xrightarrow{\alpha}_\epsilon m' \Leftrightarrow m \xrightarrow{\epsilon^*}\xrightarrow{\alpha}\xrightarrow{\epsilon^*} m' \text{ with } \alpha \neq \epsilon \tag{8}$$

Definition 17 (Weak-Simulation) *Given two labelled marked Petri nets,* $\mathcal{S}_1 = (P_1, T_1, F_1, \Lambda_1, \Sigma_\epsilon, m_1^0)$ *and* $\mathcal{S}_2 = (P_2, T_2, F_2, \Lambda_2, \Sigma_\epsilon, m_2^0)$*, a binary relation* $\mathcal{R} \subseteq \mathbb{N}^{|P_1|} \times \mathbb{N}^{|P_2|}$ *is a simulation if*

$$\forall (m_1, m_2) \in \mathcal{R} \Leftrightarrow \begin{cases} \forall \alpha \in \Sigma \text{ and } m_1' \text{ s.t. } m_1 \xrightarrow{\alpha}_\epsilon m_1', \\ \exists m_2' s.t.\ m_2 \xrightarrow{\alpha}_\epsilon m_2' \text{ and } (m_1', m_2') \in \mathcal{R} \end{cases} \tag{9}$$

If we can find a weak-simulation $\mathcal{R} \subseteq \mathbb{N}^{|P_1|} \times \mathbb{N}^{|P_2|}$ such that $(m_1^0, m_2^0) \in \mathcal{R}$ we say that $\mathcal{S}_2$ weakly simulates $\mathcal{S}_1$, which means intuitively that $\mathcal{S}_2$ can match all the moves of $\mathcal{S}_1$. Moreover if we can find another weak-simulation $\mathcal{R}' \subseteq \mathbb{N}^{|P_1|} \times \mathbb{N}^{|P_2|}$ such that $\mathcal{S}_1$ weakly simulates $\mathcal{S}_2$, we say that $\mathcal{S}_1$ and $\mathcal{S}_2$ are *weakly co-similar.*

Definition 18 (Weak-Bisimulation) *Given two labelled PN,* $\mathcal{S}_1 = (P_1, T_1, F_1, \Lambda_1, \Sigma_\epsilon, m_1^0)$ *and* $\mathcal{S}_2 = (P_2, T_2, F_2, \Lambda_2, \Sigma_\epsilon, m_2^0)$*, a binary relation* $\mathcal{R} \subseteq \mathbb{N}^{|P_1|} \times \mathbb{N}^{|P_2|}$

[3]Indeed, if we consider the alphabet A equals to the set of the transition T of the Petri Net, and L as the identity function, the two definitions are equivalent. Using labelling is more general and allows to introduce non deterministic behaviours.

is a weak-bisimulation [4] *if*

$$\forall (m_1, m_2) \in \mathcal{R} \Leftrightarrow \begin{cases} - \ \forall \alpha \in \Sigma \ and \ m_1' \ s.t. \ m_1 \xrightarrow{\alpha}_{\epsilon} m_1' \\ \ there \ is \ m_2' s.t. \ m_2 \xrightarrow{\alpha}_{\epsilon} m_2' \ and \ (m_1', m_2') \in \mathcal{R} \\ - \ \forall \alpha \in \Sigma \ and \ m_2' \ s.t. \ m_2 \xrightarrow{\alpha}_{\epsilon} m_2' \\ \ there \ is \ m_1' s.t. \ m_1 \xrightarrow{\alpha}_{\epsilon} m_1' \ and \ (m_1', m_2') \in \mathcal{R} \end{cases} \tag{10}$$

Two labelled Petri Nets $\mathcal{S}_1$ and $\mathcal{S}_2$ are *weakly bisimilar* if there is a weak bisimulation relating their initial markings. In the sequel, every transition called θ is mapped to ϵ by Λ whereas for a transition called t, $\Lambda(t) = t$. If the original PPN, $\mathcal{SP} = (P, T, Pre, Post, Par, \Lambda, \mu_0)$, has a set of transition T and T' denotes the set of transition of the constructed PPN, $\mathcal{SP}' = (P', T', Pre', Post', Par, \Lambda', \mu_0')$ then $T' = T \cup \Theta$ with $T \cap \Theta = \emptyset$. For each $t \in T$, $\Lambda(t) = t$ and for each θ in Θ, $\Lambda(\theta) = \epsilon$.

4.2 Translating P-PPN to postT-PPN

In order to simulate the behaviour of parameterised places, we translate those places in a parameterised initialisation process that needs to be fired before firing any other transitions in the net. The idea relies on using a new place π and a new transition θ enabled by this place, such that $\theta^\bullet$ initializes a P-PPN, as showed in Figure 7. We define the initial marking $m_0 = (0, ..., 0, 1)$ *i.e.* $\forall p \in P$, $m_0(p) = 0$ and $m_0(\pi) = 1$. We will show that $\mathcal{SP}'$ and $\mathcal{SP}$ are weakly-bisimilar by showing that each behaviour of $\mathcal{SP}$ can be done in $\mathcal{SP}'$ if we begin by firing θ and reciprocally.

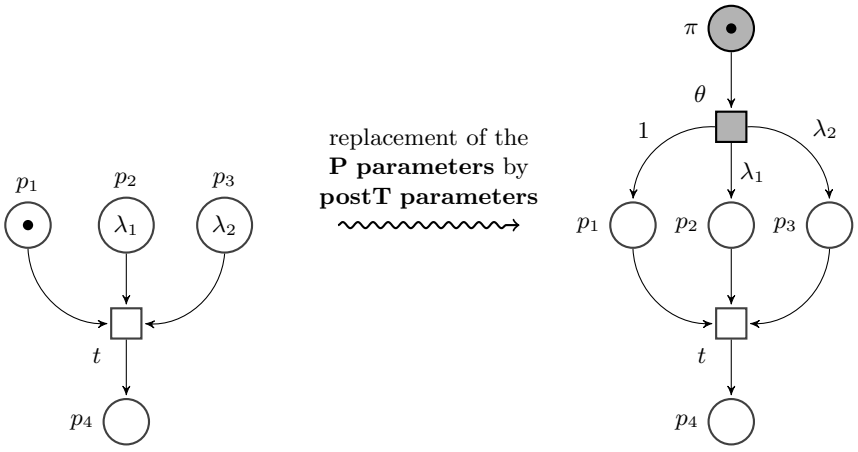


Fig. 7. From P-PPN to postT-PPN

[4] There exists several definitions of bisimulation, for instance preserving deadlocks or epsilon-branching, but the one we use is sufficient for our purpose.

Lemma 3 *$\forall \nu \in \mathbb{N}^{Par}$, $\nu(\mathcal{SP})$ and $\nu(\mathcal{SP}')$ are weakly bisimilar.*

Note that each path in $\mathcal{SP} = (P, T, Pre, Post, Par, \Lambda, \mu_0)$ can be done in $\mathcal{SP}' = (P', T', Pre', Post', Par, \Lambda', m'_0)$ by adding θ at the beginning. And reciprocally, each path in $\mathcal{SP}'$ begins by θ so is written $\theta.w$ where w is a path in $\mathcal{SP}$.

Proof. Let $\nu \in \mathbb{N}^{Par}$ a valuation of the parameters. We want to show that $\nu(\mathcal{SP})$ and $\nu(\mathcal{SP}')$ are weakly bisimilar. Let $\nu(\mu_0)$ be the parametric initial marking of $\nu(\mathcal{SP})$ and $\nu(m'_0) = m'_0$ the initial marking of $\nu(\mathcal{SP}')$. The only transition firable from m'_0 is θ and $m'_0 \xrightarrow{\theta} \nu(\mu_0)$ as shown in Figure 7. From $\nu(\mu_0)$, $\mathcal{SP}$ and $\mathcal{SP}'$ are isomorphic. So $\nu(\mathcal{SP}_1)$ and $\nu(\mathcal{SP}_2)$ are weakly-bisimilar.

Those results underline that using parameters on outputs is more powerfull than using parameters on markings. We can conclude that T-PPN are more expressive than PPN.

4.3 Translating postT-PPN to P-PPN

We will show that from a postT-PPN, $\mathcal{SP}_1 = (P_1, T_1, Pre_1, Post_1, Par_1, \Lambda_1, m_1^0)$ we can construct a P-PPN, $\mathcal{SP}_2 = (P_2, T_2, Pre_2, Post_2, Par_2, \Lambda_2, \mu_2^0)$ that weakly-simulates the behaviours of the postT-PPN. Reciprocally, the postT-PPN also weakly-simulates the behaviours of the P-PPN built.

For each transition t and place p such that the arc (t, p) is weighted by a parameter, we construct the net depicted in Figure 8 which replace this arc [5]. Therefore, $T_1 \subseteq T_2$. As previously, we introduce two labelling functions Λ_1 and Λ_2 from T_1 (resp. T_2) to T_ϵ such that, for each $t \in T_1$, $\Lambda_1(t) = \Lambda_2(t) = t$ and $\Lambda_2(t) = \epsilon$ otherwise (*i.e.* for each $t \in T_2 \backslash T_1$).

Lemma 4 *$\forall \nu \in \mathbb{N}^{Par}$, $\nu(\mathcal{SP}_1)$ and $\nu(\mathcal{SP}_2)$ are weakly cosimilar.*

Proof. We will prove the 2 weak-simulations.

- $\forall \nu \in \mathbb{N}^{Par}$, $\nu(\mathcal{SP}_2)$ simulates $\nu(\mathcal{SP}_1)$. Let us consider $\nu \in \mathbb{N}^{Par}$, $\nu(\mathcal{SP}_1)$ has the following behaviour: each time t is fired, $\nu(\lambda)$ tokens are created in p. In $\mathcal{SP}_2$, it is possible to generates $\nu(\lambda)$ tokens in p after firing the sequence $t\ \theta_{t,p,1}^{\nu(\lambda)}\ \theta_t\ \theta_{t,p,2}^{\nu(\lambda)}$, labeled $t\epsilon^*$. Moreover, this sequence resets the sub-net constructed for the weak-simulation. As the other transitions of the network are not affected, monotony gives directly the weak-simulation.

[5] Notice that if several labeled arcs come from the same transition, some places and transitions of the Figure 8 should be duplicated according to indices

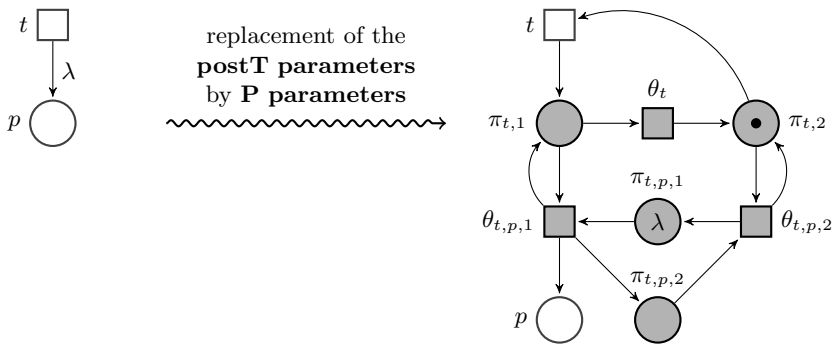


Fig. 8. From postT-PPN to P-PPN

– $\forall \nu \in \mathbb{N}^{Par}$, $\nu(\mathcal{SP}_1)$ simulates $\nu(\mathcal{SP}_2)$. Reciprocally, a marking with $\nu(\lambda)$ tokens in p allows to simulate the behaviours of every marking such that $m(p) \leq \nu(\lambda)$ according to monotony therefore, the reachable markings induced by creating less than $\nu(\lambda)$ tokens in $\mathcal{SP}_2$ are simulated by the one with $\nu(\lambda)$ tokens, and therefore by $\mathcal{SP}_1$. As the other transitions of the network are not affected, monotony gives directly the weak-simulation.

Therefore, $\mathcal{SP}_1$ and $\mathcal{SP}_2$ are *weakly co-similar*.

Remark 1. This is not a weak bisimulation. Indeed, if $\mathcal{SP}_2$ adds 3 tokens in p (leading to a marking m_2) whereas $\mathcal{SP}_1$ adds $\nu(\lambda) = 4$ tokens in p (leading to a marking m_1). Then any transitions needing more than 3 tokens could only be fired from m_1 in $\mathcal{SP}_1$ only. Here the two simulations relations are not reciprocal: m_1 would simulates m_2 but m_2 would not.

5 Decidability Results

We will now consider the parameterised properties defined in Section 2 and the different subclasses of parameterised models of Section 4. Table 1 sums up the results that we present in this section.

	$\mathscr{U}$-problem		$\mathscr{E}$-problem	
	Reachability	Coverability	Reachability	Coverability
preT-PPN	?	?	?	D
postT-PPN	?	D	?	D
PPN	U	U	U	U
distinctT-PPN	?	?	?	D
P-PPN	?	D	D	D

Table 1. Decidability results for parametric coverability and reachability

5.1 Study of Parameterised Coverability

The easiest proofs rely on monotony. Indeed, some instances simulate other instances. We recall that the *zero valuation* (written 0) is the valuation that maps every parameter to zero.

Lemma 5 *Decidability of $\mathscr{U}$-coverability on postT-PPN (resp. P-PPN) can be reduced to a test with the zero valuation.*

Proof. For postT-PPN and P-PPN, the zero valuation is the one allowing the lowest amount of behaviours for coverability *i.e.* it is the most restrictive valuation for coverability. Indeed, considering a marking m that we try to cover, m is $\mathscr{U}$-coverable if and only if there is a firing sequence w such that $m_0 \xrightarrow{w} m1 \geq m$ in the 0-instanced postT-PPN (or P-PPN). Formally, given a postT-PPN or a P-PPN $\mathcal{SP}$ and a marking m, we have:

$$\exists \nu \text{ s.t. } m \text{ is not coverable in } \nu(\mathcal{SP}) \text{ iff } m \text{ is not coverable in } 0(\mathcal{SP})$$

Indeed, for any valuation ν we can fire w in the ν-instanced PPN, leading to a marking $m2 \geq m1$ by monotony. Moreover, on the instance of a PPN (*i.e.* on a PN), the coverability is known decidable, so we can answer to the problem on the zero instanced postT-PPN (or P-PPN). If the answer is *no*, then we have found a counter example. Else, monotony directly implies that using a greater valuation ν will provide at least behaviours covering the current ones. The *winning behaviour* that allowed to answer yes for the zero-instance will still works on this *ν-instance*. So every instance will satisfy the coverability.

Therefore we can claim that $\mathscr{U}$-cov is decidable on postT-PPN and P-PPN. Let us consider $\mathscr{E}$-cov for the same subclasses.

Theorem 6 *$\mathscr{E}$-cov is decidable on P-PPN.*

Proof. Decidability of $\mathscr{E}$-cov on P-PPN:
We consider a P-PPN, $\mathcal{SP}_1$. We will now build a PN $\mathcal{S}_2$ with *token-canons* that will supply the parameterised places of $\mathcal{SP}_1$ as depicted in Figure 9. Each *token-canon* consists in two places π_p, π'_p and two transitions θ_p, θ'_p. θ_p supplies p of $\mathcal{S}_2$. Moreover, each transition of $\mathcal{SP}_1$ is added as an input and an output of π'_p, meaning that the net is blocked as long as every θ'_p has not been fired. This is repeated for each place initially marked by a parameter. We initialize $\mathcal{S}_2$ with 1 token in each π_p. So for each valuation ν of $\mathcal{SP}_1$, firing the sequence $\theta_p^{\nu(\lambda_p)}\ \theta'_p$ for each parameterised place p leads to a marking m_2 equals to the valuation of the initial marking of $\mathcal{SP}_1$. Moreover, the θ-transitions added have been fired, so every π'_p is marked. The two nets have now the same behaviour. This shows that $\mathcal{S}_2$ simulates any valuation of $\mathcal{SP}_1$. Therefore, the existence of a valuation such that a given marking is covered can be reduced to the coverability of the same marking (completed with 0 for each π_p and 1 for each π'_p added) which is known decidable as a classic coverability problem on an unbounded Petri net.

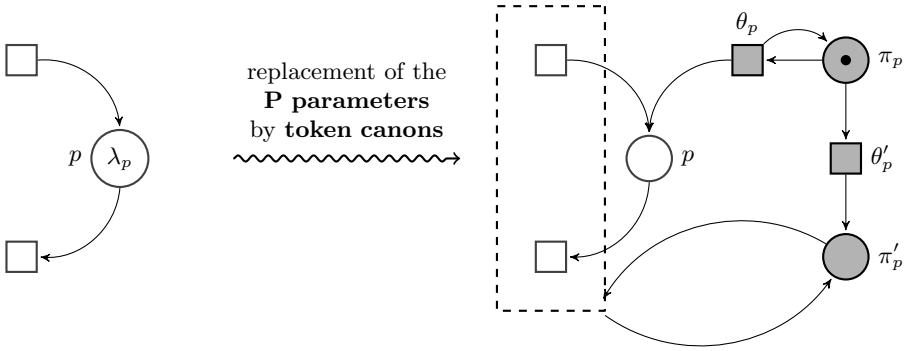


Fig. 9. From PPN to PN

Corollary 7 *$\mathscr{E}$-cov is decidable on postT-PPN.*

Proof. Decidability of $\mathscr{E}$-cov on postT-PPN:
We proved in previous section that postT-PPN and P-PPN are weakly-cosimilar. Therefore, given a postT-PPN we can built a P-PPN which is weakly-co-similar. Moreover, as coverability can be reduced to firing transition (by adding an observer transition), weak-simulation holds coverability. Theorem 6 gives us the decidability.

Theorem 8 *$\mathscr{E}$-cov is decidable for preT-PPN.*

Proof. $\mathscr{E}$-cov for the preT-PPN is decidable:
Let us consider a preT-PPN and a marking m that we try to cover. For an input transition with a weight of zero, we do not require the input place to be marked. Therefore, in terms of *input parameters*, by monotony, the zero valuation is the most permissive one for firing. Thus, there is at some valuation a firing sequence w such that $m_0 \xrightarrow{w} m_1 \geq m$ if and only if we can fire w in the *0-instanced* one, leading to a marking $m_2 \geq m_1$. Formally, given a preT-PPN $\mathcal{SP}$, we have:

$$m_0 \xrightarrow{w} m_1 \geq m \text{ in } \nu(\mathcal{SP}) \text{ iff } m_0 \xrightarrow{w} m_2 \geq m \text{ in } 0(\mathcal{SP}) \text{ with } m_2 \geq m_1$$

Informally, it means that the zero instance of the preT-PPN has the greatest amount of behaviours (in terms of coverability). Therefore it is the one which is necessary and sufficient to satisfy the $\mathscr{E}$-cov of m, meaning that if it does not satisfy the property, monotony implies that any instance of the preT-PPN will not satisfy either. If the *0-instanced* net covers m, we have a witness for the $\mathscr{E}$-cov.

Corollary 9 *$\mathscr{E}$-cov is decidable for distinctT-PPN.*

Proof. $\mathscr{E}$-cov for the distinctT-PPN is decidable:
As we can create a partition over Par between Par_{Pre} and Par_{Post}, respectively

sets of parameters involved on inputs and outputs which are disjoint. We can consider the partial valuation $0_{|Par_{Pre}}$, which maps every parameter of Par_{Pre} to 0. We therefore get a postT-PPN on which the problem is decidable. Moreover, the post-PPN built is the one with the greatest amount of behaviours for coverability as explained previously. Considering that, if we cannot find any instance of this postT-PPN satisfying the property, we cannot find any instance of this distinctT-PPN satisfying it either.

5.2 Study of Parameterised Reachability

In classic Petri nets *decidability of reachability* certainly implies *decidability of coverability*. Indeed, given a marked Petri Net and a coverability problem, we can construct another marked Petri Net over which the previous coverability problem is equivalent to a reachability problem. Actually, with notations of Figure 10 covering a marking m is equivalent to reach the marking with only one token in place p_g in the net augmented with this new place p_g, a new transition t_{goal} such that ${}^\bullet t_{goal} = m$, with $Post(p_g, t_{goal}) = 1$ and, for each place p, a transition $t_{empty,p}$ such that: if p is not equal to p_g, $Pre(p, t_{empty,p}) = 1$, $Pre(p_g, t_{empty,p}) = 1$ and $Post(p_g, t_{empty,p}) = 1$. It is clear that the same can be done for PPN, which implies that *decidability of $\mathscr{E}$-reachability* implies *decidability of $\mathscr{E}$-coverability* and *decidability of $\mathscr{U}$-reachability* implies *decidability of $\mathscr{U}$-coverability*. Section 3 provides therefore the undecidability of ($\mathscr{E}$-reach) and ($\mathscr{U}$-reach) in the general case of PPN.

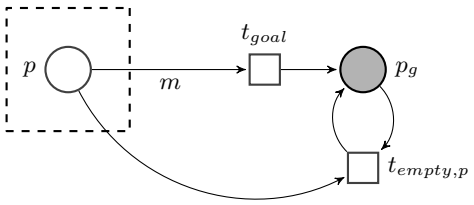


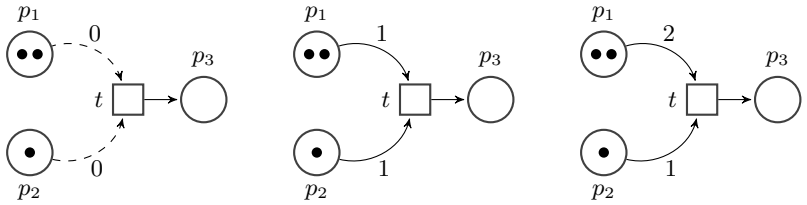
Fig. 10. Reducing Coverability to Reachability

Theorem 10 *$\mathscr{E}$-reach is decidable on P-PPN.*

Proof. We can trivially adapt the conclusion of the proof of Theorem 6. We keep the same construction: the existence of a valuation such that a given marking is reached can be reduced to the reachability of the same marking (completed with 0 for each π_p and 1 for each π'_p added) which is known decidable as a classic reachability problem on an unbounded Petri net.

Nevertheless, for the other subclasses, the decidability of reachability is more complex. Intuitively, increasing the valuation used to instanciate a preT-PPN

(resp. a postT-PPN) leads to disable (resp. enable) transitions, *i.e.* the coverability of a marking, but this is not sufifcient to deduce the exact number of tokens involved, *i.e.* reachability.



(a) 0-instance (b) 1-instance (c) ν-instance

Fig. 11. Several instances of a preT-PPN

Figure 11 presents a preT-PPN. It is obvious that using the 0-valuation leads to enable the firing of t in any case, so it allows to cover any amount of tokens in p_2. In Figure 11(a), the coverability set is $CS_0 = \{m | m \leq (2, 1, \omega)\}$. On the other hand, increasing the valuation leads to potentially disable t. We will therefore reduce the coverability set as we strengthen the pre-condition to fire t: in Figure 11(b), the coverability set is $CS_1 = \{m | m \leq (2, 1, 0) \vee m \leq (1, 0, 1)\} \subseteq CS_0$, and in Figure 11(c), we have $CS_2 = \{m | m \leq (2, 1, 0) \vee m \leq (0, 0, 1)\} \subseteq CS_1$. Nevertheless, this strengthening of the pre-condition, does not imply general consequences in terms of reachability sets. Indeed, in Figure 11(a), the reachability set is $\{(2, 1, n) | n \in \mathbb{N}\}$, whereas in Figure 11(b) we can reach $\{(2, 1, 0), (1, 0, 1)\}$ and in Figure 11(c), we can reach $\{(2, 1, 0), (0, 0, 1)\}$.

(a) 0-instance (b) 1-instance

Fig. 12. Several instances of a postT-PPN

Equivalent observations rise from the study of Figure 12. When increasing the valuation, we may fire at least the same transitions, therefore, the coverability set is increasing: Figure 12(a) can cover any markings lower or equal to $(2, 0)$ and can reach the set $\{(2, 0), (1, 0), (0, 0)\}$ whereas Figure 12(b) can cover markings lower or equal to $(2, 0)$,$(1, 1)$ or $(0, 2)$ but can reach the set $\{(2, 0)(1, 1)(0, 2)\}$.

6 Conclusion

6.1 Main results

In this paper, we have introduced the use of discrete parameters and suggested parametric versions of the well known reachability and coverability problems. The study of the decidability of those problems leads to the results summed up in Figure 13 for coverability (Classes inside a dashed outline are decidable for the two corresponding parametric coverability problems). We recall that the

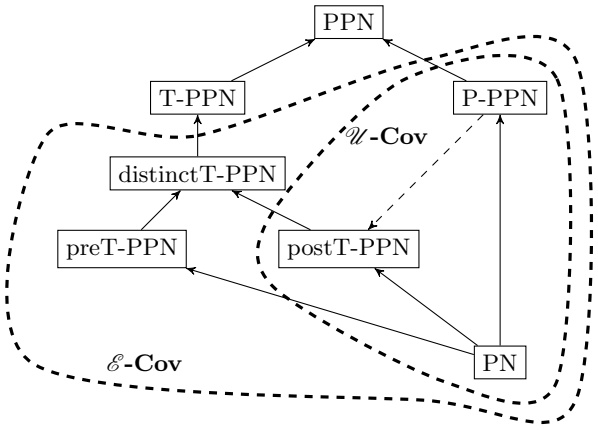


Fig. 13. What is decidable among the subclasses ? *(for coverability)*

other results are presented in Tab 1.

6.2 Future work

If we have strong intuitions for several empty cases such as decidability of $\mathscr{U}$-Coverability on preT-PPN and distinctT-PPN which would join the intuition that using the same parameters on inputs and outputs considerably increases the power of modeling of classic Petri nets, a deeper study should be carried to answer the decidability of Parametric-Reachability for instance. Being able to treat these parameterised models constitutes a scientific breakthrough in two ways:

- It significantly increases the level of abstraction in models. We will therefore be able to handle a much larger and therefore more *realistic class of models.*
- The existence of parameters can also address more relevant and *realistic verification issues.* Instead of just providing a binary response on the satisfaction or not of an expected property, we can aim to*synthetise* constraints

on the parameters ensuring that if these constraints are satisfied, the property is satisfied. Such conditions for the proper functioning of the system are essential information for the designer.

Acknowledgement

We wish to thank the anonymous reviewers, who helped us to improve the paper by their suggestions.

References

1. Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 592–601, New York, NY, USA, 1993. ACM.
2. Toshiro Araki and Tadao Kasami. Some decision problems related to the reachability problem for Petri nets. *Theoretical Computer Science*, 3(1):85–104, October 1976.
3. Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson, and Tayssir Touili. Regular model checking. In CAV, 2000.
4. G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. A symbolic reachability graph for coloured petri nets. *Theor. Comput. Sci.*, 176(1-2):39–65, April 1997.
5. C. Dufourd, A. Finkel, and Ph. Schnoebelen. Reset nets between decidability and undecidability. In Kim G. Larsen, Sven Skyum, and Glynn Winskel, editors, *Automata, Languages and Programming*, volume 1443 of *Lecture Notes in Computer Science*, pages 103–115. Springer Berlin Heidelberg, 1998.
6. N. D. Jones, L. H. Landweber, and Y. E. Lien. Complexity of some problems in Petri nets. *Theoretical Computer Science 4*, pages 277–299, 1977.
7. Richard M. Karp and Raymond E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2):147 – 195, 1969.
8. Markus Lindqvist. Parameterized reachability trees for predicate/transition nets. In Grzegorz Rozenberg, editor, *Advances in Petri Nets 1993*, volume 674 of *Lecture Notes in Computer Science*, pages 301–324. Springer Berlin Heidelberg, 1993.
9. Ernst W. Mayr. An algorithm for the general petri net reachability problem. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, STOC '81, pages 238–246, New York, NY, USA, 1981. ACM.
10. Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.

Fundamenta Informaticae XX (2016) 1–29
IOS Press

Adding data registers to parameterized networks with broadcast

Giorgio Delzanno
DIBRIS, University of Genova
Italy

Arnaud Sangnier
LIAFA, Univ Paris Diderot, Paris Cité Sorbonne, CNRS
France

Riccardo Traverso
DIBRIS, University of Genova
Italy

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

1. Introduction

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

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

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

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

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

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

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

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

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

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

2. Broadcast Networks of Register Automata

2.1. Syntax and semantics

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

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

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

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

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

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

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

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

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

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

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

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

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

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

(1) For all $i \in [1..f]$, if there exists $j \in [1..r]$ s.t. $\alpha_i = ?j$ [resp. $\alpha_i = ?\overline{j}$], then $L_M(u)[j] = L_M(v)[p_i]$ [resp. $L_M(u)[j] \neq L_M(v)[p_i]$];

(2) For all $j \in [1..r]$, if there exists $i \in [1..f]$ such that $\alpha_i = \downarrow j$ then $M[j] = L_M(v)[p_i]$ otherwise $M[j] = L_M(u)[j]$.

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

(Broadcast) $E = E'$ and there exist $v \in V$ and $\langle q, \mathbf{b}(m, p_1, \ldots, p_f), q' \rangle \in R$ such that $L_Q(v) = q$, $L'_Q(v) = q'$ and for all $u \in V \setminus \{v\}$:

- if $u \sim v$ then $L'(u) \in \mathcal{R}(v, u, \mathbf{b}(m, p_1, \ldots, p_f))$, or, $\mathcal{R}(v, u, \mathbf{b}(m, p_1, \ldots, p_f)) = \emptyset$ and $L(u) = L'(u)$;

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

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

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

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

2.2. Coverability Problem

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

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

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

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

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

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

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

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

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

3. An Example: Route Discovery Protocol

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

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

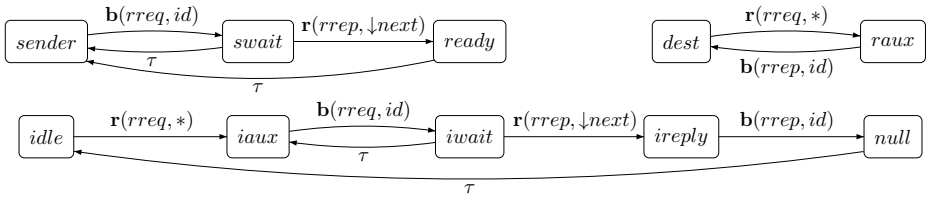


Figure 1. Route discovery example

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

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

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

4. Reconfiguration in Arbitrary Graphs

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

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

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

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

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

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

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

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

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

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

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

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

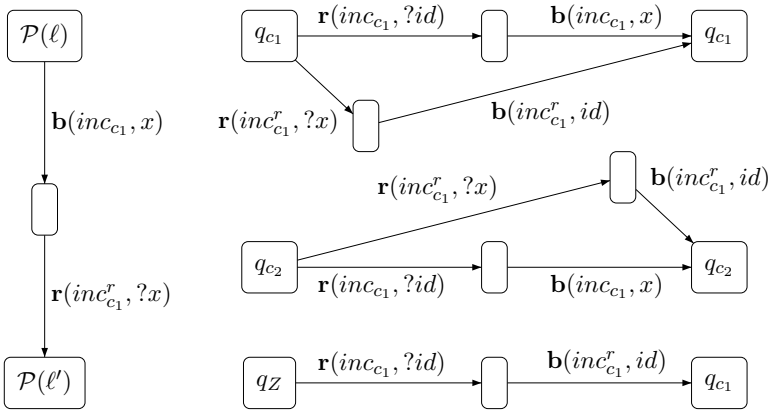


Figure 2. Increment of counter c_1

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

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

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

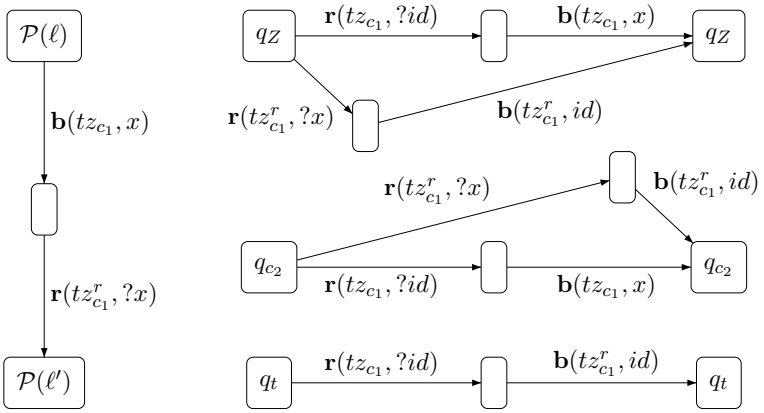


Figure 3. Testing for zero counter c_1

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

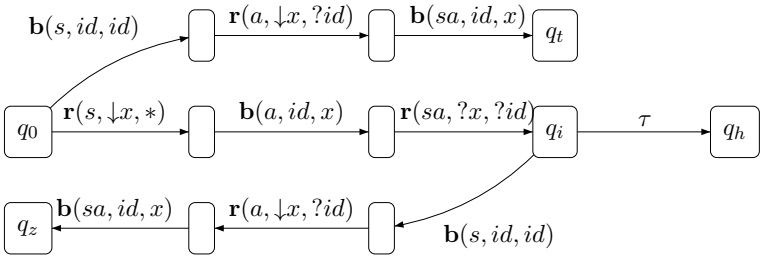


Figure 4. $\mathcal{P}_{lb}$: backward list-builder for q_h, q_z, q_t, and Γ_0 on x

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

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

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

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

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

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

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

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

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

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

Proof:
A Petri net N is a tuple $N = \langle P, T, \vec{m_0} \rangle$, where: P is a finite set of places, T is a finite set of transitions t such that ${}^\bullet t$ and $t^\bullet$ are multisets of places (pre- and post-conditions of t), and $\vec{m_0}$ is a multiset of places that indicates how many tokens are located in each place in the initial net marking. Given a marking $\vec{m}$, the firing of a transition t such that ${}^\bullet t \subseteq \vec{m}$ leads to a new marking $\vec{m'}$ obtained as $\vec{m'} = \vec{m} \setminus {}^\bullet t \cup t^\bullet$. A Petri net P is 1-safe if in every reachable marking every place has at most one token. Reachability of a marking $\vec{m_1}$ from the initial marking $\vec{m_0}$ is decidable for Petri nets, and PSPACE-complete for 1-safe nets [8].

Given a 1-safe net $N = \langle P, T, \vec{m_0} \rangle$ and a marking $\vec{m_1}$, we encode the reachability problem into $Cov(|N|, 1)$. We will assume that $P = \{p_1, \ldots, p_r\}$ and that p_1 is the unique place such that $\vec{m_0}(p_1) = 1$ and p_r the only place such that $\vec{m_1}(p_r) = 1$ (without lost of generality we can in fact reduce the reachability problem of 1-safe net into such a simple case). We now explain how to simulate the behavior of N with a $(r, 1)$-protocol $\mathcal{P} = \langle Q, R, q_0 \rangle$. The protocol $\mathcal{P}$ contains two control states *full* and *empty* from which the only possible action is the broadcast of a message containing the value stored in the first register. This is depicted in Figure 5. We see that nodes in state *empty* will always broadcast messages

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

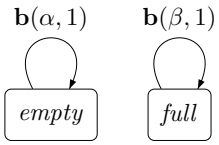


Figure 5. Encoding the nodes of type *full* and *empty*

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

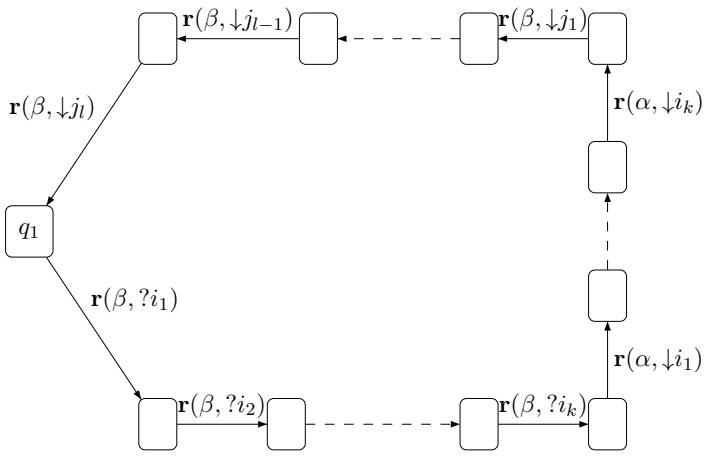


Figure 6. Encoding transition t with ${}^\bullet t = \{p_{i_1}, \ldots, p_{i_k}\}$ and $t^\bullet = \{p_{j_1}, \ldots, p_{j_l}\}$

full or *empty* by broadcasting a message that no one will receive and then a node can go to state q_1 by receiving a message sent by a node *full* and it will store the identifier in the first register, this to simulate that the initial marking of N is the one with one token in p_1. Finally, a node will go in state *end* if there is an identifier of a node of type *full* in the f-th register. One can then easily prove that the protocol $\mathcal{P}$ verifies the property that $\vec{m_1}$ is reachable from $\vec{m_0}$ in N if and only if there exists $\gamma \in Reach(\mathcal{P})$ such that $\gamma \models end$. $\square$

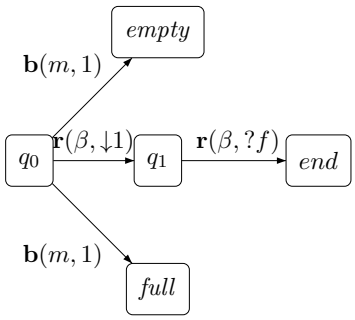


Figure 7. Initialization of the simulation and ending of the simulation of the 1-safe net

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

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

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

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

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

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

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

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

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

3. For each $v \in V$, if there exist $j \neq j' \in [1..r]$ s.t. $L_M(v)[j] = L_M(v)[j']$, i.e. a node with the same value in two distinct registers, then there exists a self loop $(w, j, j', w) \in \delta$ with $\lambda_Q(w) = L_Q(v)$.

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

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

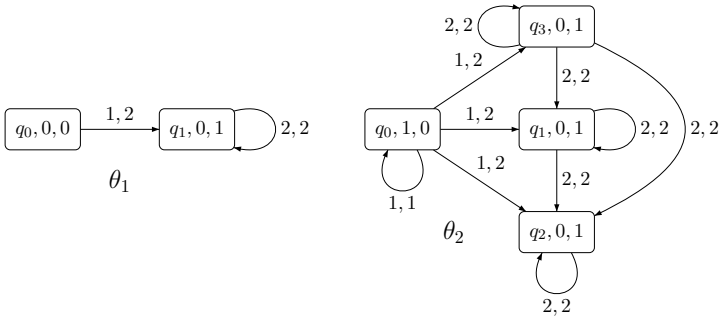


Figure 8. Example of computations of symbolic post

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

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

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

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

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

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

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

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

Lemma 4.2. Let θ, θ' be two symbolic configurations. We have $[\![\theta]\!] \cup [\![\theta']\!] \subseteq [\![\theta \sqcup \theta']\!]$.

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

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

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

□

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

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

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

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

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

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

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

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

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

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

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

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

$W' := W$, $\delta' := \delta$, $\lambda' := \lambda$
if there exists $w \in W$ such that $\lambda_Q(w) = q$ **then**
  if there does not exist $w' \in W'$ such that $\lambda'_Q(w') = q'$ **then**
    Create a node $w' \in W'$ with $\lambda'_Q(w') = q'$ and $\lambda'_M(w') := \vec{0}$
  end if
  Let $w' \in W'$ such that $\lambda'_Q(w') = q'$
  for all $j \in [1..r]$, if $\lambda_M(w)[j] = 1$ then $\lambda'_M(w')[j] := 1$
  for all $(w, a, b, w'') \in \delta'$, $\delta' := \delta' \cup \{(w', a, b, w'')\}$
  for all $(w, a, b, w) \in \delta'$, $\delta' := \delta' \cup \{(w', a, b, w')\}$
  for all $\langle q'', \mathbf{r}(m, \alpha), q''' \rangle \in R$ such that there exists $w'' \in W$ with $\lambda_Q(w'') = q''$ **do**
    if $\alpha = ?\overline{k}$ or $\alpha = *$ **then**
      if there does not exist $w''' \in W'$ such that $\lambda'_Q(w''') = q'''$ **then**
        Create a node $w''' \in W'$ with $\lambda'_Q(w''') = q'''$ and $\lambda'_M(w''') := \vec{0}$
      end if
      Let $w''' \in W'$ such that $\lambda'_Q(w''') = q'''$
      for all $j \in [1..r]$, if $\lambda_M(w'')[j] = 1$ then $\lambda'_M(w''')[j] := 1$
      for all $(w'', a, b, v) \in \delta'$, $\delta' := \delta' \cup \{(w''', a, b, v)\}$
      for all $(w'', a, b, w'') \in \delta'$, $\delta' := \delta' \cup \{(w''', a, b, w''')\}$
    end if
    if $\alpha = \downarrow k$ **then**
      if there does not exist $w''' \in W'$ such that $\lambda'_Q(w''') = q'''$ **then**
        Create a node $w''' \in W'$ with $\lambda'_Q(w''') = q'''$ and $\lambda'_M(w''') := \vec{0}$
      end if
      Let $w''' \in W'$ such that $\lambda'_Q(w''') = q'''$
      $\lambda_M(w''')[k] = 1$ and $\delta' := \delta' \cup \{(w', p, k, w'''), (w''', k, k, w''')\}$
      For all $j \in [1..r]$, if $\lambda_M(w'')[j] = 1$ then $\lambda'_M(w''')[j] := 1$
      if $\lambda_M(w)[p] = 1$, Then for all $(w, p, b, v) \in \delta$, $\delta' := \delta' \cup \{(w''', k, b, v)\}$
      for all $(w'', a, b, v) \in \delta'$ with $a \neq k$, $\delta' := \delta' \cup \{(w''', a, b, v)\}$
      For all $(w'', a, b, w'') \in \delta'$ with $a \neq k$ and $b \neq k$, $\delta' := \delta' \cup \{(w''', a, b, w''')\}$
    end if
    if $\alpha = ?k$ and $(w, p, k, w'') \in \delta$ or $(w'', k, p, w) \in \delta$ **then**
      if there does not exist $w''' \in W'$ such that $\lambda'_Q(w''') = q'''$ **then**
        Create a node $w''' \in W'$ with $\lambda'_Q(w''') = q'''$ and $\lambda'_M(w''') := \vec{0}$
      end if
      Let $w''' \in W'$ such that $\lambda'_Q(w''') = q'''$
      for all $j \in [1..r]$, if $\lambda_M(w'')[j] = 1$ then $\lambda'_M(w''')[j] := 1$
      for all $(w'', a, b, v) \in \delta'$, $\delta' := \delta' \cup \{(w''', a, b, v)\}$
      for all $(w'', a, b, w'') \in \delta'$, $\delta' := \delta' \cup \{(w''', a, b, w''')\}$
    end if
  end for
end if

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

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

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

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

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

Thanks to Lemma 4.2, we can deduce that

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

and hence we have:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Proof:
We will prove that for all $n \in \mathbb{N}$, for all $\gamma \in [\![\texttt{POST}^n_{\mathcal{P}}(\theta)]\!]$, there exists $\gamma' \in \texttt{post}^*_{\mathcal{P}}([\![\theta]\!])$ such that $\gamma \trianglelefteq \gamma'$. We reason by induction on n. First in the case $n = 0$, the property holds trivially (we recall that $\texttt{POST}^n_{\mathcal{P}}(\theta) = \theta$). Now assume the property holds for $n \in \mathbb{N}$ and we will show it is still true for $n + 1$. Let $\gamma \in [\![\texttt{POST}^{n+1}_{\mathcal{P}}(\theta)]\!]$. From Lemma 4.8, we deduce that there exists $\gamma' \in \texttt{post}^*_{\mathcal{P}}([\![\texttt{POST}^n_{\mathcal{P}}(\theta)]\!]) \cap [\![\texttt{POST}^{n+1}_{\mathcal{P}}(\theta)]\!]$ such that $\gamma \trianglelefteq \gamma'$. Hence there exists $\gamma_1 \in [\![\texttt{POST}^n_{\mathcal{P}}(\theta)]\!]$ such that $\gamma_1 \Rightarrow \ldots \Rightarrow \gamma'$. By induction hypothesis, there exist $\gamma_2 \in \texttt{post}^*_{\mathcal{P}}([\![\theta]\!])$ such that $\gamma_1 \trianglelefteq \gamma_2$. But then thanks the first item of Lemma 4.7, since $\gamma_1 \trianglelefteq \gamma_2$ and $\gamma_1 \Rightarrow \ldots \Rightarrow \gamma'$ we deduce that there exists γ_2' such that $\gamma_2 \Rightarrow \ldots \Rightarrow \gamma_2'$ and $\gamma' \trianglelefteq \gamma_2'$. Note that since $\gamma_2 \in \texttt{post}^*_{\mathcal{P}}([\![\theta]\!])$, then $\gamma_2' \in \texttt{post}^*_{\mathcal{P}}([\![\theta]\!])$ and since $\gamma \trianglelefteq \gamma'$ and $\gamma' \trianglelefteq \gamma_2'$, we also have $\gamma \trianglelefteq \gamma_2'$. □

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

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

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

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

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

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

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

5. Fully Connected Topologies and No Reconfiguration

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

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

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

Proof:
We now show that the protocol $\mathcal{P}^{fc}_{lb}$ in Figure 9 is a list builder for $\Rightarrow_b$ and Γ^{fc}_0 on x and states q_h, q_z, q_t (where the lists are built backwards from q_t). Let $\gamma_0 = \langle V, E, L\rangle \in \Gamma^{fc}_0$ be an initial configuration. As soon as a node $v \in V$ decides to start the construction of the list, it broadcasts its identifier to every other node with a message $\mathbf{b}(\mathit{tail}, \mathit{id})$. Since the network is fully connected, every process has to react to the

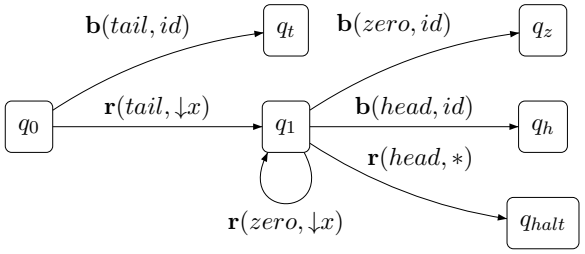


Figure 9. $\mathcal{P}_{lb}^{fc}$: list builder for $\langle q_h, q_z, q_t \rangle$ and fully connected configurations on x

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

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

□

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

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

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

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

The following properties then hold.

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

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

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

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

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

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

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

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

We focus our attention on $\mathtt{PRE}_b$ for a given broadcast b and a given symbolic configuration ξ. In the rest of the section we assume that

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

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

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

To illustrate the rationale behind our construction of $\mathtt{PRE}_b$, we first illustrate the key ideas with the help of an example.

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

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

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

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

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

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

$$Ext_{q'}(\xi) = \{\xi\} \cup \{\xi \oplus [q']\} \cup \{[m_1, \ldots, m_i \oplus [q'], \ldots, m_k] \mid i \in [1, k]\}$$

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

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

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

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

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

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

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

- For rules with test and ignore action $pre(u_i)$, $i \in [1, 2]$, is obtained by replacing each occurrence of a postcondition with the corresponding precondition of a (non deterministically selected) reception rule for msg (i.e $pre(u_i)$ and u_i have the same size).

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

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

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

$$m_j = u_j \oplus v_j$$

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

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

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

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

5.3. Decision Procedure

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

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

Proof:
We show how to apply the symbolic predecessor computation based on $\mathtt{PRE}_{\mathcal{P}}$. Let φ be a query with set of variables Z. The (in)equalities in φ induce a finite set $P_1, \ldots, P_k$ of partitions of Z. Each partition $P_i = \{X^i_1, \ldots, X^i_{u_i}\}$ is such that X^i_j contains variables that may take the same value (i.e. there are no $\neq$ constraints between them in φ). For a partition X, we define the multiset m_X of symbols in Q for which there exists a predicate $q(z)$ with $z \in X$. Thus $P_i = \{X^i_1, \ldots, X^i_{u_i}\}$ can be represented via the multiset of multisets $s_i = [m_{X^i_1}, \ldots, m_{X^i_{u_i}}]$. The set $I = \{s_1, \ldots, s_k\}$ corresponds to the minimal elements of the set of configurations that satisfy φ. To apply the algorithm we set $Bad = I^{\uparrow}$, i.e., $I = min(Bad)$. We compute then the least fixpoint of $\mathtt{PRE}_{\mathcal{P}}$, say $\mathtt{PRE}^*_{\mathcal{P}}(I)$. To check if the resulting set of symbolic configurations contains an initial state, we need to search for a finite basis $\langle V, L, E\rangle$ (where $E = V \times V \setminus \{(v,v) \mid v \in V\}$) in which all nodes have initial states as labels, and in which there cannot be two nodes with the same value in the register (initially all processes have distinct values in local registers). Using the multiset representation, we need to search for a multiset consisting of multisets of the form $[q_0]$ where q_0 is the initial state of the protocol, i.e. coverability holds if and only if $[[q_0], \ldots, [q_0]] \in \mathtt{PRE}^*_{\mathcal{P}}(I)$. □

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

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

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

6. Conclusions

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

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

Table 1. Decidability and complexity boundaries

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

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

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

References

[1] Abdulla, P. A., Cerans, K., Jonsson, B., Tsay, Y.-K.: General Decidability Theorems for Infinite-State Systems, *LICS'96*, IEEE Computer Society, 1996.

[2] Abdulla, P. A., Delzanno, G., Rezine, O., Sangnier, A., Traverso, R.: On the Verification of Timed Ad Hoc Networks, *FORMATS'11*, 6604, Springer, 2011.

[3] Abdulla, P. A., Jonsson, B.: Ensuring completeness of symbolic verification methods for infinite-state systems, *Theor. Comput. Sci.*, **256**(1-2), 2001, 145–167.

[4] Alur, R., Dill, D. L.: A Theory of Timed Automata, *Theor. Comput. Sci.*, **126**(2), 1994, 183–235.

[5] Aminof, B., Jacobs, S., Khalimov, A., Rubin, S.: Parameterized Model Checking of Token-Passing Systems, *VMCAI'14*, 8318, 2014.

[6] Bertrand, N., Fournier, P., Sangnier, A.: Playing with Probabilities in Reconfigurable Broadcast Networks, *FOSSACS'14*, 8412, Springer, 2014.

[7] Bollig, B., Gastin, P., Schubert, J.: Parameterized Verification of Communicating Automata under Context Bounds, *RP'14*, 8762, 2014.

[8] Cheng, A., Esparza, J., Palsberg, J.: Complexity Results for 1-Safe Nets, *TCS*, **147**(1&2), 1995, 117–136.

[9] Clarke, E. M., Talupur, M., Touili, T., Veith, H.: Verification by Network Decomposition, *CONCUR'04*, 3170, 2004.

[10] Delzanno, G.: Constraint-Based Verification of Parameterized Cache Coherence Protocols, *FMSD*, **23**(3), 2003, 257–301.

[11] Delzanno, G., Rosa-Velardo, F.: On the coverability and reachability languages of monotonic extensions of Petri nets, *Theor. Comput. Sci.*, **467**, 2013, 12–29.

[12] Delzanno, G., Sangnier, A., Traverso, R.: Parameterized Verification of Broadcast Networks of Register Automata, *RP'13*, 8169, Springer, 2013.

[13] Delzanno, G., Sangnier, A., Traverso, R.: *Parameterized Verification of Broadcast Networks of Register Automata (Technical Report)*, Technical report, TR-13-03, DIBRIS, University of Genova, 2013, Available at the URL `http://verify.disi.unige.it/publications/`.

[14] Delzanno, G., Sangnier, A., Traverso, R., Zavattaro, G.: On the Complexity of Parameterized Reachability in Reconfigurable Broadcast Networks, *FSTTCS'12*, 18, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[15] Delzanno, G., Sangnier, A., Zavattaro, G.: Parameterized Verification of Ad Hoc Networks, *CONCUR'10*, 6269, Springer, 2010.

[16] Delzanno, G., Sangnier, A., Zavattaro, G.: On the Power of Cliques in the Parameterized Verification of Ad Hoc Networks, *FOSSACS'11*, 6604, Springer, 2011.

[17] Emerson, E. A., Namjoshi, K. S.: On Model Checking for Non-Deterministic Infinite-State Systems, *LICS'98*, IEEE Computer Society, 1998.

[18] Esparza, J.: Keeping a Crowd Safe: On the Complexity of Parameterized Verification (Invited Talk), *STACS'14*, 25, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014.

[19] Esparza, J., Finkel, A., Mayr, R.: On the Verification of Broadcast Protocols, *LICS'99*, IEEE Computer Society, 1999.

[20] Esparza, J., Ganty, P., Majumdar, R.: Parameterized Verification of Asynchronous Shared-Memory Systems, *CAV'13*, 8044, Springer, 2013.

[21] Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere!, *Theor. Comput. Sci.*, **256**(1-2), 2001, 63–92.

[22] German, S. M., Sistla, A. P.: Reasoning about Systems with Many Processes, *J. ACM*, **39**(3), 1992, 675–735.

[23] Kaminski, M., Francez, N.: Finite-Memory Automata, *Theor. Comput. Sci.*, **134**(2), 1994, 329–363.

[24] Konnov, I., Veith, H., Widder, J.: Who is afraid of Model Checking Distributed Algorithms?, Unpublished contribution to: CAV Workshop $(EC)^2$, 2012.

[25] Lazic, R., Newcomb, T., Ouaknine, J., Roscoe, A. W., Worrell, J.: Nets with Tokens which Carry Data, *Fundam. Inform.*, **88**(3), 2008, 251–274.

[26] Minsky, M.: *Computation, Finite and Infinite Machines*, Prentice Hall, 1967.

[27] Schmitz, S., Schnoebelen, P.: The Power of Well-Structured Systems, *CONCUR'13*, 8052, Springer, 2013.

[28] Schnoebelen, P.: Revisiting Ackermann-Hardness for Lossy Counter Machines and Reset Petri Nets, *MFCS'10*, 6281, Springer, 2010.