Dipartimento di Informatica, Bioingegneria,
Robotica ed Ingegneria dei Sistemi

Formal Verification of Ad Hoc Networks

by

Riccardo Traverso

Theses Series **DIBRIS-TH-2014-04**

DIBRIS, Università di Genova
Via Opera Pia, 13 16145 Genova, Italy http://www.dibris.unige.it/

Università degli Studi di Genova

Dipartimento di Informatica, Bioingegneria,
Robotica ed Ingegneria dei Sistemi

Dottorato di Ricerca in Informatica

Ph.D. Thesis in Computer Science

Formal Verification of Ad Hoc Networks

by

Riccardo Traverso

January, 2014

Dottorato di Ricerca in Informatica
Dipartimento di Informatica, Bioingegneria, Robotica ed Ingegneria dei Sistemi
Università degli Studi di Genova

DIBRIS, Università di Genova
Via Opera Pia, 13
I-16145 Genova, Italy
http://www.dibris.unige.it/

Ph.D. Thesis in Computer Science (S.S.D. INF/01)

Submitted by Riccardo Traverso
DIBRIS, Università di Genova
riccardo.traverso@unige.it

Date of submission: January 2014

Title: Formal Verification of Ad Hoc Networks

Advisor: Giorgio Delzanno
DIBRIS, Università di Genova, Italy
giorgio.delzanno@unige.it

External Reviewers:
Gianluigi Zavattaro
INRIA - FOCUS Research Team
Università di Bologna, Italy
gianluigi.zavattaro@unibo.it

Barbara König
Department of Computer Science and Applied Cognitive Science
Universität Duisburg-Essen
barbara_koenig@uni-due.de

Abstract

Thanks to recent advances in hardware and software architectures, computer systems are more and more dependent on concurrency and distribution: multi-core processors are becoming the default choice even for embedded systems, and wireless devices are spreading in everyday life. The increasing complexity of systems makes testing and verification an even more difficult and challenging problem.

This work fits in this scenario, starting from a simple observation: usually, individual processes cooperate in a concurrent or distributed system by exploiting or discovering knowledge about their interconnections. Examples can be found at several levels, ranging from cache coherence protocols [Ste90] at the physical layer up to publish/subscribe systems [BQV05] at the application layer.

The behaviour of such protocols is often influenced by the number of nodes and the shape of interconnections, making very hard to predict under which connectivity configurations a given protocol might expose errors.

This search for a (possibly arbitrary) configuration connectivity graph easily lifts verification topics from finite-state systems to infinite-state systems, but unfortunately the majority of studies are focused on finite representations. In order to address the aforementioned open problems, the main goal of this work is the study of parameterized verification for concurrent and distributed systems where individual nodes communicate over a network topology, i.e. via (selective) broadcast messages. Ad hoc network protocols are the main source of inspiration for the features considered for the models.

Table of Contents

Chapter 1

Introduction

Thanks to recent advances in hardware and software architectures, computer systems are more and more dependent on concurrency and distribution: multi-core processors are becoming the default choice even for embedded systems, and wireless devices are spreading in everyday life. The increasing complexity of systems makes testing and verification an even more difficult and challenging problem.

This work fits in this scenario, starting from a simple observation: usually, individual processes cooperate in a concurrent or distributed system by exploiting or discovering knowledge about their interconnections. Examples can be found at several levels, ranging from cache coherence protocols [Ste90] at the physical layer up to publish/subscribe systems [BQV05] at the application layer.

Examples of widely used approaches to specify and verify concurrent and distributed systems include: state machines and automata [BLL+95, Alu91, HNSY94, Kat99], petri nets [Mur89], process calculi, [Hoa78, Pra95, MS10, MS92], and (graph) rewriting systems [BVEG+87, MOM02]. They have been applied and studied in conjunction with extensions like, e.g., broadcast communication, mobility, loss of messages, message corruption and time-dependent constraints, but very often the proposed models either do not directly support/exploit the specification of the connectivity graph as a separated input with respect to the protocol - forcing users to hardcode the links in it [BDL04, Hoa78, FvHM07] - or do not support the encoding of arbitrary network topologies at all [ADM04]. When it comes to analyze a protocol whose behaviour is influenced by the number of nodes and the shape of interconnections, questions about safety properties are related in a natural way to the search for a network topology that enables the protocol to violate the given property.

This search for a (possibly arbitrary) connectivity graph easily lifts verification topics from finite-state systems to infinite-state systems, but unfortunately the majority of studies are focused on finite representations. In order to address the aforementioned open problems, the main goal of

this work is the study of parameterized verification for concurrent and distributed systems where individual nodes communicate over a network topology, i.e. via (selective) broadcast messages. Ad hoc network protocols are the main source of inspiration for the features considered for the models. This work builds on the previous and first results on the parameterized verification of a model for ad hoc networks [DSZ10] where the communication topology is represented by a graph and the behaviour of each node is represented by a finite-state automaton that can either send a message (emitter role), wait for a message (receiver role), or perform an update of its internal state. Already at this level of abstraction, verification of protocols with broadcast communication turns out to be a very difficult task. A formal account of this problem is given in [DSZ10, DSZ11], where parameterized control state reachability is proved to be undecidable in an automata-based protocol model in which configurations are arbitrary graphs. The parameterized control state reachability problem consists in verifying the existence of an initial network configuration (with unknown size and topology) that may evolve into a configuration in which at least one node is in a given control state. If such a control state represents a protocol error, then this problem naturally expresses (the complement of) a safety verification task in a setting in which nodes have no information a priori about the size and connection topology of the underlying network.

While the original model only allows selective (i.e. restricted to the vicinity) synchronous broadcast messages from a finite alphabet, we study extensions with clocks, data, mobility of nodes, and asynchronous communication. For all of these models, we investigate decidability of verification problems formulated in terms of reachability, starting from initial configurations of arbitrary size, of a network configuration satisfying some property. Often, this translates into some parameterized version of the control state reachability problem.

1.1 Technical Contributions

Chapter 3 – Timed Ad Hoc Networks

We modify the Ad Hoc Networks model [DSZ10] described in Section 2.5 in order to use timed automata to describe the behaviour of the nodes [ADR+11], taking inspiration from the LMAC protocol for medium access control (see Section 1.2.2). Transitions are guarded by Boolean combinations of (in)equalities, each of them comparing a local clock with a constant. As a result of the firing of a transition, a subset of the clocks of the process may be reset. We study both dense and discrete time, and we explore how restricting the communication topology to classes of graphs affects decidability. It turns out that, with dense time, the parameterized local state reachability problem is undecidable already for fully connected topologies and two clocks per node, and star topologies with rays composed by two nodes and one clock. Undecidability still holds for a more general class where graphs are bound not to have simple paths with more than

N nodes, for any $N \geq 5$. To obtain decidability, in fully connected topologies clocks have to be restricted to one per node, while stars must only have one node per ray and one clock per node. With discrete time, the problem is decidable for any number of clocks in the class of graphs with bounded simple paths.

Chapter 4 – Reconfigurable Broadcast Networks

By allowing the underlying communication graph to arbitrarily rearrange its edges at runtime, we show that the considered parameterized reachability problems have a lower complexity [DSTZ12b] with respect to static communication graphs [DSZ10, DSZ11]. We first prove PTIME-completeness of a more general version of the local state reachability problem, where we can combine in formulae with conjunctions and disjunctions individual constraints on the presence of nodes in a given local control state. We then move to more complex formulae where we also allow negation, i.e. atoms can also be used to check for the absence of processes in a given local state. With these enriched reachability queries, the problem becomes NP-complete. Furthermore, when atoms are allowed not only to check the existence of a node in a given local state but they can also count nodes by providing lower and upper bounds, we prove the problem to be PSPACE-complete.

Chapter 5 – Asynchronous Broadcast Networks

For what concerns asynchronous communication we equip each node with a local buffer for incoming messages, thus separating the act of sending from the act of receiving [DT12, DT13a]. We explore decidability and complexity for three different policies to handle mailboxes (FIFO, multiset, lossy FIFO), fully connected and arbitrary topologies. We also prove that, with mailboxes treated as multisets of messages, the ability to test whether the local mailbox is empty makes the difference between decidability and undecidability. All of the problems proved to be decidable are shown to be PTIME-complete. It is interesting to see how the upper bounds to complexity are given via reductions to the Reconfigurable Broadcast Networks model of Chapter 4 without altering the protocol in any way. This shows that, with respect to the parameterized local state reachability problem, the two models are fundamentally equivalent: the same protocol, when run with either one or another of the two semantics, is able to expose the same local control states (even though with different executions).

Chapter 6 – Broadcast Networks of Register Automata

With networks of register automata [DST13b] nodes have a finite set of local registers which may be used to store natural numbers. Broadcast messages can carry data, that may be assigned to or tested against local registers. Furthermore, we model dynamic reconfigurations of the network

like in [DSTZ12b]. This model, with respect to the previous ones without data, is a step closer to model the core parts of more complex protocols, like consensus and routing (examples in Section 1.2.1 and Section 1.2.3). We focus on decidability and complexity of parameterized reachability, from arbitrarily large initial configurations, of a configuration exposing some given local states and patterns (expressed as relations between registers). We study how the amounts $r, f \in \mathbb{N}$ of data respectively contained in local registers and messages influences the problem. Without reconfigurations and by restricting configurations to fully connected graphs only, the problem is non-elementary already with $r = 0$ local registers and $f = 0$ message fields, and also with $r = 1$ registers and $f = 1$ fields. Undecidability occurs with $r \geq 2$ and $f \geq 1$. With dynamic reconfigurations we obtain better results, as the problem is PSPACE-complete with $r \geq 1$ and $f = 1$, and undecidable only for $r \geq 2$ and $f \geq 2$.

Chapter 7 – Model Checking with Groove

To conclude our study, we move back to models with finitely many processes and consider an orthogonal problem to those studied in the parameterized case, i.e., the state explosion problem that arises during the exploration of the corresponding reachability graph. To attack this problem, contrary to standard approaches based on explicit or symbolic model checking, we present preliminary experiments with a graph-based search engine called Groove [GdMR+12, DT13b] that provides effective symmetry reductions heuristics based on graph isomorphism. We consider as case studies two distributed protocols, we model them in the Groove input language as sets of graph transformation rules, and finally we test finite instances against appropriate safety properties. The two protocols are the Gafni-Bertsekas Link Reversal Routing algorithm (see Section 1.2.1), and the Paxos distributed consensus algorithm (see Section 1.2.3). As a comparison with more traditional, vector-based models, we present a Promela model of the latter and show the gain in state space reduction with respect to the results obtained with the Spin model checker.

1.2 Example Protocols

In our search for interesting features which should ideally be available to have in models close enough to real protocols, we chose a few examples to get inspiration from. We briefly list the in the next few subsections. The key features underlying the examples are:

- spontaneous movement of nodes,
- local data (e.g. for routing tables, medium access information, ...),
- exchange of structured messages,
- timeout-triggered transitions.

1.2.1 Link Reversal Routing

Routing protocols for ad hoc networks represent an important class of distributed protocols that would be interesting to validate by means of parameterized verification. Among ad-hoc routing protocols, Link Reversal Routing (LRR) [WW11, DT13b] is a particularly fitting example, since individual nodes rely only on local information about their neighbourhood. LRR algorithms are designed for large, dynamically changing networks, in which topology changes are too frequent to make flooding of routing informations a viable solution. The main goal is to quickly repair a corrupted route with a new valid, but not necessarily optimal, one. The adaptivity and scalability of LRR algorithms make them suitable for ad hoc networks. LRR works with an overlay network used to identify routes to a specific destination node. The overlay network is defined via a Directed Acyclic Graph (DAG) with exactly one destination node (a node with only incoming links). Other nodes have either incoming and outgoing links or just outgoing links. When the last outgoing link of a node breaks, the node becomes a sink for the network and it starts route maintenance. One possible strategy, called full reversal, is to reverse all incoming edges. After link reversal, the maintenance procedure is recursively applied to the surrounding nodes. The algorithm stabilizes after finitely many steps if the graph is not partitioned.

1.2.2 Lightweight Medium Access Protocol

The Lightweight Medium Access Protocol (LMAC) [vHH04], belonging to the family of time division multiple access protocols, is a distributed algorithm specifically designed with wireless sensor networks in mind in order to be energy efficient. The protocol assigns a unique time slot to each node in the wireless network. During its time slot, a node may broadcast any message without the risk colliding with other transmissions. Time slots may be reused as long as they do not cause collisions (i.e. when the intersection of the vicinities of any two nodes assigned with the same time slot is empty).

The rationale of the algorithm is the following. Any node in the network, during its time slot, transmits also a control message with information about the status of assigned and free time slots in his vicinity. All of its neighbours update status accordingly, making sure that both the current time slot and all of the ones specified by the control message are listed as already assigned. If the time slot chosen by the receiving node is already used by some other one (distant up to two hops), the node picks another random time slot. When a node detects a collision between the transmissions of two or more of its neighbours, it marks the time slot as in conflict and warns his neighbours with his next control message.

This protocol was the target of several model checking studies [FvHM07, SRS09] which managed to find configurations exposing unresolved collisions in networks up to 6 nodes.

1.2.3 Paxos

Paxos [Lam98, Lam01] is a distributed algorithm for solving consensus in a network of (possibly unreliable) processes. The consensus problem requires a number of processes to choose a single value shared between the participants. Since individual processes may be faulty (i.e. they may crash), the protocol is designed to take this into account. It distinguishes between three roles for processes: proposer, acceptor, and learner. Each proposer is assigned with a subset of natural numbers to be used as fresh round identifiers. When starting a new round r, the proposer broadcasts a message to collect votes from acceptors. Each acceptor which did not already answered to a round $r' > r$ answers the proposer by sending the last round identifier and proposed value seen. If a quorum is reached, the proposer broadcasts another message with the newly proposed value v, which is the one associated to the biggest round identifier sent by acceptors in the previous phase, if any, or a new one otherwise. Acceptors react by updating their local variables with the new values and answer with a confirmation message directed towards learner processes containing the selected round r and value v. Learners count such messages, and when a pair $\langle r, v \rangle$ is voted by the majority of acceptors they finally confirm v as the elected value. At this point, the protocol is guaranteed to stabilize: each subsequent election round will continue to choose v.

Chapter 2

Preliminaries

2.1 Two Counter Machines

Definition 1. *A two-counter machine $\mathcal{M}$ is a triple $\langle Loc, Inst, \ell_0 \rangle$ where:*

- *Loc is a finite set of control locations;*
- *$Inst \subseteq Loc \times Op \times Loc$ is a finite set of instructions such that $Op = \{c_i{+}{+}, c_i{-}{-}, c_i == 0 \mid i \in \{1, 2\}\}$ is a set of operators over the counters c_1 and c_2;*
- *$\ell_0 \in Loc$ is the initial location.*

Configurations are tuples $\langle \ell, v_1, v_2 \rangle$ such that $\ell \in Loc$ is the current location and v_1, v_2 are natural numbers denoting the current value of x_1 and x_2, respectively.

The operational semantics is defined in a standard way: the execution of increment and decrement updates the control location and the current value of the corresponding counter, while a zero-test updates the location whenever the test is satisfied in the current state of the counter. Formally:

- $\langle \ell, v_1, v_2 \rangle$ evolves in one step into $\langle \ell', v_1 + 1, v_2 \rangle$ [resp. $\langle \ell', v_1, v_2 + 1 \rangle$] by executing the instruction $\langle \ell, x_1 + +, \ell' \rangle$ [resp. $\langle \ell, x_2 + +, \ell' \rangle$];
- $\langle \ell, v_1, v_2 \rangle$ evolves in one step into $\langle \ell', v_1 - 1, v_2 \rangle$ [resp. $\langle \ell', v_1, v_2 - 1 \rangle$] by executing the instruction $\langle \ell, x_1 - -, \ell' \rangle$ [resp. $\langle \ell, x_2 - -, \ell' \rangle$];
- $\langle \ell, v_1, v_2 \rangle$ evolves in one step into $\langle \ell', v_1, v_2 \rangle$ by executing the instruction $\langle \ell, x_1 == 0, \ell' \rangle$ whenever $v_1 = 0$ [resp. $\langle \ell, x_2 == 0, \ell' \rangle$ whenever $v_2 = 0$].

The halting problem for two counter machines is a well known undecidable problem. We will often use it with reductions in order to prove undecidability results.

2.2 Petri Nets

Definition 2. *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);*
- *$\vec{m_0}$ 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 said to be *1-safe* if, in every reachable marking, every place has at most one token. Reachability of a specific marking $\vec{m_1}$ from the initial marking $\vec{m_0}$ is EXPSPACE-complete problem [Lip76, Rac78] for Petri nets, and PSPACE-complete for 1-safe nets [CEP95].

2.3 Well-Structured Transition Systems

The theory of Well Structured Transition Systems (WSTSs) [ACJT96, AJ01] identifies a general class of infinite-state systems for which several verification problems are decidable, like reachability.

The underlying idea behind WSTSs is to somehow derive a finite symbolic representation of the infinite-state system taken into consideration. It is related to symbolic model checking, but instead of being a mere tool to contain state-space explosion, here, constraints are the key to achieve decidability of verification problems.

The fundamental step is the definition of an ordering $\sqsubseteq$ over the states of the system; constraints are finitely encoded by upper sets with respect to $\sqsubseteq$, meaning that a single state can encode infinitely many others, and transitions between states are lifted to transitions between constraints. Whenever a system is a WSTS the reachability algorithm eventually terminates since, because of the ordering, at a certain point newly computed constraints will be always subsumed by older ones. The algorithm is a backward analysis that starts from the target set of constraints and

iteratively computes one-step predecessors until no new ones are found. At each step, if the intersection with the initial states of the system is non empty, the algorithm answers that the target set is reachable. When the intersection is empty even when every possible predecessor of the target set has been computed, the algorithm answers that the target set is not reachable.

Please note that, because of the abstraction, it is not possible to compute reachability of individual states. However, in infinite-state systems, it is more common to have to check the reachability of sets of good/bad states instead. These sets are best described with patterns of constraints, so this is not really an issue.

The class of WSTSs is very wide, including many examples of previously known infinite state systems [FS01], including:

- finite-state systems;
- Petri nets and extensions:
  - transfer arcs,
  - reset arcs;
- certain types of string rewrite systems;
- lossy channel systems;
- timed automata.

The following paragraphs shortly present the basic definitions.

Upper sets 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 $(A, \preceq)$, the *minimal elements* of a set $B \subseteq A$ is the set $M \subseteq B$ satisfying the two following conditions: for every $b \in B$, there exists $m \in M$ such that $m \preceq b$ and for every element $m \in M$, there does not exist $m' \in M$ such that $m' \preceq m$.

Well-quasi-order A preorder $(D, \sqsubseteq)$ is also a *well-quasi-order (wqo)* if, for every infinite sequence $d_0, d_1, d_2, \ldots$ of elements in D, there exist i and j such that $i < j$ and $d_i \sqsubseteq d_j$. Some examples of well-quasi-orders are:

- the set of natural numbers and the less or equal relation;
- any finite set with the identity relation;

- by Dickson's lemma, if $(D, \sqsubseteq)$ is a wqo, then for any k also $(D^k, \sqsubseteq^k)$ is a wqo.

It is well-known that 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.

Transition system A *transition system* $\mathcal{T}$ is a tuple $\langle \Sigma, \rightarrow \rangle$ where Σ is a set of *states* (or *configurations*) of the system, and $\rightarrow \subseteq \Sigma \times \Sigma$ is the *transition relation*. It is common to write $\rightarrow^*$ to mean the reflexive and transitive closure of $\rightarrow$. Given a set of states $S \subseteq \Sigma$, the *predecessors set* $\mathrm{Pre}_{\mathcal{T}}(S)$ is defined as $\{\sigma \in \Sigma \mid \exists \sigma' \in S . \sigma \rightarrow \sigma'\}$.

Set-reachability problem Given a transition system $\mathcal{T} = \langle \Sigma, \rightarrow \rangle$ and two sets of states I and F, the set-reachability problem consists in checking whether there exist $i \in I$ and $r \in R$ such that $i \rightarrow^* r$. As a side note, the same condition could equivalently expressed in a more compact way as $I \rightarrow^* R$.

Constraint system Let $\mathcal{T} = \langle C, \rightarrow \rangle$ be a transition system. A *constraint system* for $\mathcal{T}$ is a set Ψ of objects called constraints. An interpretation function $[\![\cdot]\!] : \Psi \longrightarrow 2^C$ associates a semantics to each constraint. The set Ψ is equipped with a quasi order $\sqsubseteq$ such that $\phi \sqsubseteq \phi'$ if $[\![\phi]\!] \supseteq [\![\phi']\!]$. Both the interpretation function and the quasi order are extended to sets of constraints Φ and Φ', where $[\![\Phi]\!] = \bigcup_{\phi \in \Phi} [\![\phi]\!]$ and $\Phi \sqsubseteq \Phi'$ if $[\![\Phi]\!] \supseteq [\![\Phi']\!]$.

A constraint system Ψ is *effectively closed* with respect to its transition system $\mathcal{T} = \langle C, \rightarrow \rangle$ if, for every $\phi \in \Psi$, there exist a finite set of constraints Φ such that $\mathrm{Pre}_{\mathcal{T}}([\![\phi]\!]) = [\![\Phi]\!]$. Moreover, if $\sqsubseteq$ is a well-quasi-order, then Ψ is said to be *well-quasi-ordered*.

Decidability of set-reachability As proved in [AJ01], the set-reachability problem is decidable for a transition system $\mathcal{T}$ and an associated constraint system Ψ if Ψ is well-quasi-ordered and effectively closed. The algorithm is reported in listing 2.1.

2.4 Networks of Timed Automata

Developed by the Universities of Aalborg and Uppsala, UPPAAL [BDL04, BLL+95, upp] is a well known model checking tool for systems described by networks of communicating timed automata (NTA) with data types. It has a WSTS at its core, and a rich assertional language to express properties to be verified.

```
Procedure SymbolicReachability
Input
    T = ⟨C, →⟩ : transition system
    I = [[Φ_I]] : set of states
    Φ_F : finite set of constraints
Var V, W : sets of constraints
Begin
    W := Φ_F;
    V := ∅;
    while W ≠ ∅ do
        choose φ ∈ W;
        W := W − {φ};
        if [[φ]] ∩ I ≠ ∅ then
            return reachable;
        elseif ∄φ' ∈ V : φ' ⊑ φ then
            V := V ∪ {φ};
            W := W ∪ Φ where [[Φ]] = Pre_T([[φ]]);
        fi
    od
    return unreachable;
End
```

Listing 2.1: Algorithm for set-reachability.

Timed automata have a finite set of control states and an arbitrary number of clocks ranging over real numbers. Conditions over local clocks may be used as guards for transitions and invariants on states, which should be respectively satisfied in order to take a transition, and in order to move to and stay in a particular control state. While taking a transition a subset of the clocks may be reset to 0, according to the applied transition rule. Time flows at the same rate for every clock of every automaton in the network. Synchronization between two automata takes place through unicast communication channels. The channels are fixed a priori by the protocol. For a set C of clocks, $B(C)$ denotes the set of conjunctions over simple conditions of the form $x \lhd c$ or $x - y \lhd c$ where $x, y \in C$, $c \in \mathbb{N}$, and $\lhd \in \{<, \leq, =, \geq, >\}$.

Timed Automaton A Timed Automaton (TA) is a tuple $\langle L, l_0, C, A, E, I \rangle$ where: L is a set of control locations; $l_0 \in L$ is the initial state; C is a set of clocks; A is a set of actions, co-actions and internal actions; $E \subseteq L \times A \times B(C) \times 2^C \times L$ is a set of transition rules labeled by an action, a guard over the clocks, and a reset set; $I : L \to B(C)$ is a function that assigns invariants to locations.

Before introducing the semantics, some notations. Clock valuations are functions $u : C \to \mathbb{R}_{\geq 0}$. The initial clock valuation u_0 maps every $c \in uppClocks$ to 0. The set of all clock valuations is written $\mathbb{R}^C$. Let u be a valuation. $u \in I(l)$ means that u satisfies the invariant on l. Given $d \in \mathbb{R}_{\geq 0}$, $u + d$ is the valuation such that $\forall c \in C.(u + d)(c) = u(c) + d$. For $r \subseteq C$, $[r \mapsto 0]u$ is a valuation u' such that for every clock c in C:

$$u'(c) = \begin{cases} 0 & c \in r \\ u(c) & c \notin r \end{cases}$$

Semantics of a TA The semantics of a TA $\langle L, l_0, C, A, E, I\rangle$ is given by the labelled transition system $\langle S, s_0, \to\rangle$ where $S \subseteq L \times \mathbb{R}^C$ is the set of states, $s_0 = \langle l_0, u_0\rangle$ is the initial state, and $\to \subseteq S \times (\mathbb{R}_{\geq 0} \cup A) \times S$ is the transition relation such that:

- $\langle l, u\rangle \xrightarrow{d} \langle l, u + d\rangle$ if $d \in \mathbb{R}_{\geq 0}$ and $\forall d' \in [0, d].u + d' \in I(l)$;
- $\langle l, u\rangle \xrightarrow{a} \langle l', u'\rangle$ if $\exists\langle l, a, g, r, l'\rangle \in E$ such that $u \in g \wedge u' = [r \mapsto 0]u \wedge u' \in I(l')$.

Network of TA A Network of n Timed Automata (NTA) is a family of TA $\mathcal{A}_i = \langle L_i, l_i^0, C, A, E_i, I_i\rangle$ where $1 \leq i \leq n$. A location vector $\bar{l}$ is a vector $\langle l_1, \ldots, l_n\rangle$. The initial location vector is $\bar{l_0} = \langle l_1^0, \ldots, l_n^0\rangle$. Invariants from every automaton are put together with $I(\bar{l}) = \bigwedge_i I_i(l_i)$. The substitution of l_i with l_i' in a location vector $\bar{l}$ is written $\{l_i'/l_i\}\bar{l}$

Semantics of a NTA The semantics of a NTA $\langle L_i, l_i^0, C, A, E_i, I_i\rangle$ with n automata is given by the labelled transition system $\langle S, s_0, \to\rangle$ where $S = (L_1 \times \ldots \times L_n) \times \mathbb{R}^C$ is the set of states, $s_0 = \langle \bar{l_0}, u_0\rangle$ is the initial state, and $\to \subseteq S \times S$ is the transition relation such that:

- $\langle \bar{l}, u\rangle \xrightarrow{d} \langle \bar{l}, u + d\rangle$ if $d \in \mathbb{R}_{\geq 0}$ and $\forall d' \in [0, d].u + d' \in I(\bar{l})$;
- $\langle \bar{l}, u\rangle \xrightarrow{a} \langle \{l_i'/l_i\}\bar{l}, u'\rangle$ if $\exists\langle l_i, \tau, g, r, \bar{l_i'}\rangle \in E_i$ such that $u \in g \wedge u' = [r \mapsto 0]u \wedge u' \in I(\{l_i'/l_i\}\bar{l})$.
- $\langle \bar{l}, u\rangle \xrightarrow{a} \langle \{l_j', l_i'/l_j, l_i\}\bar{l}, u'\rangle$ if there exist $\langle l_i, c?, g_i, r_i, \bar{l_i'}\rangle$ and $\langle l_j, c!, g_j, r_j, \bar{l_j'}\rangle$ such that $u \in (g_i \wedge g_j) \wedge u' = [r_i \cup r_j \mapsto 0]u \wedge u' \in I(\{l_j', l_i'/l_j, l_i\}\bar{l})$.

The basic NTA model is further extended by UPPAAL with several useful features that make relatively easy to describe richer models:

- bounded integers;

- broadcast channels that enables one sender $c!$ to synchronize with an arbitrary number of receivers $c?$;
- arrays of clocks, channels and integers;
- user-defined data types;
- urgent channels, special channels that when involved in enabled synchronization transitions they do not allow time to pass;
- urgent locations, where time is not allowed to pass;
- committed locations, that are urgent locations whose outgoing transitions must take place before any other.

The tool performs model checking by taking advantage of more compact, symbolic representations of the state space [HNSY94, YPD94, BLL+95]. Symbolic network configurations are pairs $\langle \bar{l}, D \rangle$ where $\bar{l}$ is a location vector and D is a constraint over clocks and data. The constraint characterizes all actual valuations and states that the system is allowed to assume in $\langle \bar{l}, D \rangle$.

Despite the scalability issues, UPPAAL methods has been successfully applied to a variety of verification contexts, both by industry and academics. This approach has a drawback in describing the dynamics of a constantly changing network. Once the model is done, the interconnections are fixed and it is not possible to answer to questions about the existence of a network configuration that exposes errors. This problem shows up clearly when trying to verify, in example, protocols for wireless networks, where we would like safety properties to hold for every possible network configuration. A good example of this problem is in the work of Fehnker, van Hoesel and Mader [FvHM07]: they defined a simplified model of LMAC medium access control protocol – in order to limit state-space explosion – and then they searched for undetected collisions in all networks consisting of four and five nodes. They had to manually change the model for every possible topology, and ended up running more than eight thousand queries to complete their task.

2.5 Ad Hoc Networks

In this section we will recall the model called Ad Hoc Networks [DSZ10] that is the basis for our work.

An Ad Hoc Network (AHN) is a network of communicating automata distributed over an undirected graph used to encode connectivity information. Processes synchronize themselves via synchronous selective broadcast messages over a finite message alphabet. Every node in the network may communicate to its neighbourhood only. When a node n broadcasts message s, every

neighbor m of n enabled to the reception of a (via some transition rule) is forced to take the transition together with N.

Several verification problems have been studied for this model, each of those parameterized with respect to the number of processes and the connectivity graph:

- COVER: reachability of a configuration with at least one process in a given state, starting from some arbitrarily big initial configuration;
- TARGET: reachability of a configuration with all processes in a given state, starting from some arbitrarily big initial configuration;
- REPEAT-COVER: existence of a computation traversing infinitely often configurations with at least one process in a given state, starting from some arbitrarily big initial configuration.

As proved in [DSZ10, DSZ11], those problems are all undecidable for arbitrary graphs and one has to restrict initial configurations of the system to specific classes of graphs in order to gain decidability at least for COVER (via WSTSs).

For AHNs where the connectivity graph is enabled to change between a transition and another, COVER, TARGET and REPEAT-COVER become decidable. This enforces the thought that interactions between the features of a model should be researched throughly, in order to find a better compromise between the expressive power of the model and the decidable verification problems.

Example 1. *In order to clarify the semantics, in Figure 2.1 we provide an example execution of a protocol which counts at least the two transition rules shown. Local control states of individual processes (i.e. nodes) are represented by natural numbers. In the first configuration considered, the only node labelled by* 1 *is enabled to fire a broadcast transition and send message* a *to all of its neighbours. Of them, only those labelled by* 3 *have the reception rule for message* a *enabled and therefore they synchronize with the sender in such a way that, in the following state of the system, both the sender and each enabled receiver in his neighbourhood change states according to the transition rules. The process goes on as long as new processes reach state* 1.

2.6 Lossy Channel Systems

Lossy Channel Systems [AJ93] are a class of infinite-state systems where finite-state processes interact via a finite set of unbounded lossy FIFO channels, i.e. channels with unbounded capacity but that can non-deterministically lose messages at runtime. The behaviour individual processes is specified via a labelled transition system. A transition can be either a simple internal action, an

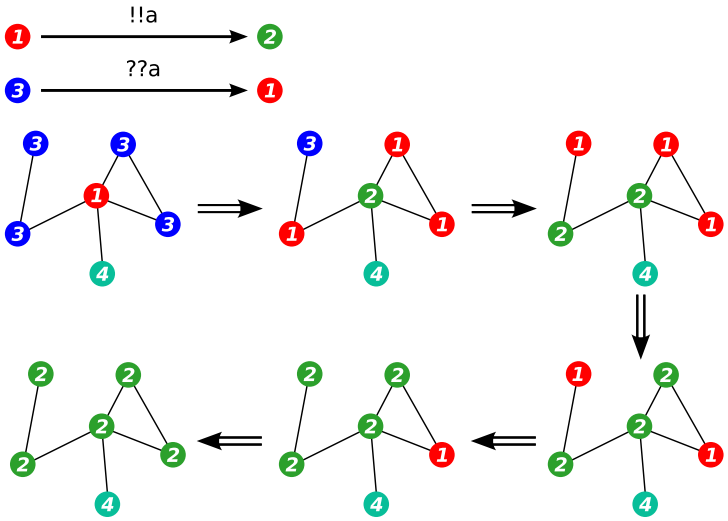


Figure 2.1: Example execution of an AHN

observable action, a send of a message from a finite alphabet to one of the channels, or a reception of the message at the head of a channel. The ordered sequence (string) of all observable actions of an execution is called a trace.

Such systems have been thoroughly investigated during the years, because of their good properties w.r.t. verification and their expressiveness, e.g. for modelling the Alternating Bit Protocol. Decidable problems for Lossy Channel Systems include checking reachability of either global or local states and checking inclusion of the set of all traces of the system in a given regular set of strings.

In Chapter 5, which is dedicated to Asynchronous Broadcast Networks, we investigate, among other ones, a lossy semantics for messages similar to that of Lossy Channel Systems.

Chapter 3

Timed Ad Hoc Networks

3.1 Introduction

In this chapter, an extended version of the FORMATS'11 paper [ADR+11], we introduce a model of distributed systems obtained by enriching the AHN model of [DSZ10] (Section 2.5) with time-sensitive specification of individual nodes. In the resulting model, called Timed Ad Hoc Newtorks (TAHN), the connection topology is still modeled as a graph in which nodes communicate via broadcast messages. The behaviour of a node however is defined as a timed automaton. Each node has a finite set of clocks, and all clocks in the network advance at the same rate. Transitions in each node are guarded by conditions on clocks, and may reset some of them depending on the post-condition.

Following [DSZ10, DSZ11], we study decidability and undecidability properties of the reachability problems parametric on the initial configuration of a TAHN, i.e., the problem of 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 the Timed Networks of [AJ03]. A major difference between TAHN and Timed Networks is that in the latter model the connection topology is always a fully-connected graph, i.e., broadcast communication is not selective in the sense that 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, local control state 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 hold in the more general class of graphs with bounded simple path (for some bound $N \geq 5$ on the length – number of nodes – of paths). Note that this result contrasts with the untimed case where the state reachability is decidable for bounded path topologies (hence for stars) [DSZ10].
- 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 timed automaton in each node has a single clock (as in Timed Networks).
- Finally, when considering discrete time instead of continuous time, we show that the local control 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.

Acknowledgements The author would like to thank Prof. Parosh Aziz Abdulla, Prof. Giorgio Delzanno, Othmane Rezine, and Prof. Arnaud Sangnier as co-authors of the article [ADR+11] from which this chapter is taken.

Related Work

Model checking for timed automata has been applied to verify protocols for ad hoc networks with a fixed number of nodes in [FvHM07]. Models with a discrete global clock and a lazy exploration of configurations of fixed size has been considered in [SRS09]. Formal specification languages for timed models of ad hoc networks have been proposed, e.g., in [MS10]. In contrast to these works, we consider 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 transfers. In the untimed case the combination of rendez-vous and transfers is considered in Datanets, a model in which processes have data taken from an ordered domain [LNO+08].

3.2 Timed Ad Hoc Networks

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.

3.2.1 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 continuously all at the same rate, and, 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 through *selective broadcast*, where a process sends a broadcast message which can be received only by its neighbors in the network. The communication topology is represented through a graph. Finally, transitions may have guards with constraints on the current values of the local clocks and they may reset the values of some clocks.

We now move to the formal definition of the model. We assume that each process operates on a set X of clocks. 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 idea:

(i) τ represents a local move ;

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

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

We are now ready to formally define the model we will study.

Definition 3. *A Timed Ad Hoc Network (TAHN for short) is a pair $\mathcal{T} = \langle G, P \rangle$ where:*

- *$G = \langle V, E \rangle$ is graph composed of a finite set of vertices V and a set of edges $E \subseteq V \times V \setminus \{(v, v) \mid v \in V\}$;*
- *P, called the protocol, is a tuple $\langle Q, X, \Sigma, \mathcal{R}, q^{init} \rangle$ 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, the graph G defines the topology of $\mathcal{T}$ where the set V represents the nodes and E defines the connectivity. The vertices belonging to an edge are called the *endpoints* of the edge. For an edge $\langle u, v \rangle \in E$, we often use the notation $u \sim v$ and say that the vertices u and v are adjacent to each other. For what concerns P, it defines the protocol that runs inside each node, where Q is the set of *local states*, while $\mathcal{R}$ is a set of *rules* describing the behavior of the protocol. We will use the notation $\langle q, g \xrightarrow{e} R, q' \rangle$ to represent the rule (q, g, e, R, q'). For a protocol $P = \langle Q, X, \Sigma, \mathcal{R} \rangle$, we denote by $nbclocks(P)$ the size of X, i.e. the number of clocks it uses.

3.2.2 Operational Semantics

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

- $\mathcal{Q} : V \mapsto Q$ is a function that maps each node of the graph to 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.

Note 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 of $\mathcal{T}$. A configuration is said to be *initial* if it is of the form $\langle \mathcal{Q}^{init}, \mathcal{X}^{init} \rangle$ 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.

Before giving the semantics, we still need to introduce a notation to characterize the nodes in a configuration that are able to receive a message a sent by a node $v \in V$. Given a configuration $\gamma = \langle \mathcal{Q}, \mathcal{X} \rangle$ and a message $a \in \Sigma$, we denote by $En(\gamma, a)$ the following set of vertices able to receive a in γ:

$$En(\gamma, a) = \{v \in V \mid \text{ there is a rule } \langle q, g \xrightarrow{??a} R, q' \rangle \text{ s.t. } \mathcal{Q}(v) = q \text{ and } \mathcal{X}(v) \models g\}$$

The semantics associated to the TAHN $\mathcal{T}$ is then given by the transition system $(\mathcal{C}_\mathcal{T}, \Longrightarrow_\mathcal{T})$ where the transition relation $\Longrightarrow_\mathcal{T} \subseteq \mathcal{C}_\mathcal{T} \times \mathcal{C}_\mathcal{T}$ is such that given two configurations $\gamma = \langle \mathcal{Q}, \mathcal{X} \rangle$ and $\gamma' = \langle \mathcal{Q}', \mathcal{X}' \rangle$, we have $\gamma \Longrightarrow_\mathcal{T} \gamma'$ if and only if one of the following conditions is satisfied:

Local: There exists a rule $\langle q, g \xrightarrow{\tau} R, q' \rangle$ 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 $w \in V - \{v\}$, we have $\mathcal{Q}'(w) = \mathcal{Q}(w)$, $\mathcal{X}'(w) = \mathcal{X}(w)$.

Broadcast: There exists a rule $\langle q, g \xrightarrow{!!a} R, q' \rangle$ 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 $w \in V - \{v\}$, we have:

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

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 $\mathcal{X}'(v)(x) = \mathcal{X}(v)(x) + \delta$.

3.2.3 Topologies

In the rest of the chapter we will often restrict the configurations of the considered TAHN to belong to specific families of graphs, which 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. We will 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 $\ell \geq 0$, $\texttt{STAR}(\ell)$ is the star topology of depth ℓ. It characterizes graphs $G = \langle V, E \rangle$ 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 exactly one $v_{i+1} \in V_{i+1}$ with $v_i \sim v_{i+1}$;

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

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

  In other words, in a star topology of dimension ℓ there is a central node v_0 and an arbitrary number of *rays*. A ray consists of a sequence of ℓ nodes, starting from v_0 followed by some $v_1 \in V_1$, and then by some $v_2 \in V_2$, etc. We call v_0 the *root*, call the nodes in $V_1, \ldots, V_{\ell-1}$ the *internal nodes*, and call the nodes in V_ℓ the *leaves* of G.

- For $\ell \geq 0$, $\texttt{BOUNDED}(\ell)$ is the bounded path topology with bound ℓ. It characterizes graphs for which the length of the maximal simple path is bounded by ℓ. Formally, if $G \in \texttt{BOUNDED}(\ell)$ with $G = \langle V, E \rangle$ then there does not exist a finite sequence of vertices $(v_i)_{1 \leq i \leq m}$ 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 $i \in \{1, \ldots, m-1\}$.
- $\texttt{CLIQUE}$ is the set of cliques which characterizes graphs $G = \langle V, E \rangle$ such that $v \sim w$ for all $v, w \in V$ with $v \neq w$.

3.2.4 State reachability problem

We now present the verification problem object of our study. This problem consists in determining whether there exists an initial configuration such that the connectivity graph belongs to a certain topology from which it is possible to reach another configuration exhibiting a specific state (for instance an error state). We do not restrict the number of nodes appearing in initial configurations, but we look for initial configurations belonging to specific topologies. Note that each topology we previously introduced has an infinite cardinality, hence we cannot simply enumerate all of the graphs belonging to the given topology. In fact, as we shall see, the main challenge is exactly the infinite number of initial configurations.

Let $\mathcal{T} = \langle G, P \rangle$ be a TAHN with $G = \langle V, E \rangle$ and $P = \langle Q, X, \Sigma, \mathcal{R}, q^{init} \rangle$. We say that a configuration γ_n is *reachable* in $\mathcal{T}$ if 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 the associated transition system starting from an initial configuration γ_0. Given a state $q \in Q$, we say that that q is reachable in $\mathcal{T}$ if there exists a reachable configuration $\gamma = \langle \mathcal{Q}, \mathcal{X} \rangle$ and a vertex $v \in V$ such that $\mathcal{Q}(v) = q$.

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

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

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

In [DSZ10], a model of Ad Hoc Networks without time has been studied; it is the same as the one we introduced here if we consider only protocols without clocks. The authors have shown that, if we consider all the finite graphs for the set of configurations, then the control state reachability is undecidable, but we can regain the decidability by restricting the configurations to bounded path graphs. The results are repeated using our notations by the following theorem.

Theorem 1. *[DSZ10]*

1. $\texttt{TAHN}-\texttt{Reach}\,(\texttt{GRAPH}, 0)$ *is undecidable.*

2. *For all* $K \geq 1$, $\texttt{TAHN}-\texttt{Reach}\,(\texttt{BOUNDED}(K), 0)$ *is decidable.*

3.3 Timed Networks with Transfers

3.3.1 Presentation

In [AJ03], the authors introduce a model called Timed Network (TN for short), which can be used to describe systems consisting of an arbitrary number of processes, each of which being a finite-state system operating on real-valued clocks. This model is similar to the model of TAHN we introduced, but they can be distinguished by the three fundamental differences stated below.

1. A TN contains a distinguished *controller* that is a finite-state automaton without any clocks (we point out the fact that adding clocks to the controller does not affect the decidability results).
2. Each process in a TN may communicate with all other processes, hence it is not meaningful to describe topologies in the case of TNs.
3. Communication takes place through rendez-vous between fixed sets of processes rather than broadcast messages.

In [AJ03] it is proved that for such a model, when considering processes equipped with a single clock, one can decide whether there is an initial configuration from which the controller is able to reach a given control state. This decidability result however does not hold any more as soon as the processes use two clocks. In this section, we will propose an extension of TNs by adding the so-called transfer to the model. The rationale behind transfer actions is the following: when a transfer action is fired, every process in a given local state is forced to change state. This is as with transfer arcs in Petri nets, which allow to move every token contained in a certain place to another one [FRSVB02]. These transfers will be useful to simulate in some particular cases the behaviour of TAHNs. Since in a TAHN a broadcasted message forces every process enabled to the reception to react, it is clear that a model based on rendez-vous between a finite number of processes is not sufficient to mimic such a behaviour.

3.3.2 Syntax

In this section we provide the definition of *Timed Networks with Transfer* (shortly *TNT*) which extends the TNs introduced in [AJ03]. Similarly to TAHNs, a TN contains an arbitrary number of identical *timed processes* that operate on a finite number of local, real-valued clocks which

advance together at the same rate. In addition, a TNT can change its configuration according to a finite number of *rules*. Each rule describes a set of transitions in which the controller and a fixed number of processes synchronize and simultaneously change their states, following either a rendez-vous or a transfer. A rule may have guards based on the local state of the controller, together with the local states and clock values of the involved processes. If the preconditions for a rule are satisfied, then a transition may be performed. In this case, the controller and each participating process simultaneously change their states according to the rule. Finally, during the firing of a transition, a process may reset some of its clocks to 0. Note that TNT use the same notion of guards and resets as the one previously introduced for TAHNs.

Definition 4. *A Timed Network with Transfer (TNT)* $\mathcal{N}$ *is a tuple* $\left(Q^{ctrl}, Q^{proc}, X, \mathcal{R}, q^{init}_{ctrl}, q^{init}_{proc}\right)$*, 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 state;* $q^{init}_{proc} \in Q^{proc}$ *is an initial process state; and* $\mathcal{R}$ *is a finite set of* rules*. Rule are of the form:*

$$\begin{bmatrix} q_0 \\ \rightarrow \\ q'_0 \end{bmatrix} \quad \begin{bmatrix} q_1 \\ g_1 \rightarrow R_1 \\ q'_1 \end{bmatrix} \quad \cdots \quad \begin{bmatrix} q_n \\ g_n \rightarrow R_n \\ q'_n \end{bmatrix} \quad \begin{bmatrix} p_1 \\ g'_1 \rightarrow_T R'_1 \\ p'_1 \end{bmatrix} \quad \cdots \quad \begin{bmatrix} p_\ell \\ g'_\ell \rightarrow_T R'_\ell \\ p'_\ell \end{bmatrix}$$

such that:

- $q_0, q'_0 \in Q^{ctrl}$*;*
- *for all* $i \in \{1, \ldots, n\}$ $q_i, q'_i \in Q^{proc}$*,* $g_i \in \mathcal{G}(X)$ *and* $R_i \in 2^X$*;*
- *for all* $i \in \{1, \ldots, \ell\}$ $p_i, p'_i \in Q^{proc}$*,* $g'_i \in \mathcal{G}(X)$ *and* $R'_i \in 2^X$*;*
- *for all* $i \in \{1, \ldots, n\}$ *and* $j, k \in \{1, \ldots, \ell\}$*,* $q_i \neq p_j$*, and,* $j \neq k$ *implies* $p_j \neq p_j$*.*

For a rule of a TNT, we call rendez-vous communication the actions with $\rightarrow$, whereas actions with $\rightarrow_T$ correspond to transfers. As we shall see later a rule describes a set of transitions of the network. Finally, a *Timed Network* (*TN*) is simply a TNT where the rules do not contain transfers.

3.3.3 Operational semantics

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

- I is a finite *index set*;
- $q \in Q^{ctrl}$;

- $\mathcal{Q} : I \mapsto Q^{proc}$;
- $\mathcal{X} : I \mapsto [X \to \mathbb{R}^{\geq 0}]$.

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., $|\gamma| = |I|$. We denote by $\mathcal{C}_{\mathcal{N}}$ the set of configurations of $\mathcal{N}$. A configuration $\gamma^{init} = (I, q, \mathcal{Q}, \mathcal{X})$ is said to be *initial* if $q = q_{ctrl}^{init}$, $\mathcal{Q}(i) = q_{proc}^{init}$, and $\mathcal{X}(i)(x) = 0$ for each $i \in I$ and $x \in X$. This means that an execution of a TNT 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 giving the formal definition of the transition relation associated to $\mathcal{N}$, we explain the behaviour of a rule of the form:

$$\left[\begin{array}{c} q_0 \\ \to \\ q_0' \end{array}\right] \quad \left[\begin{array}{c} q_1 \\ g_1 \to R_1 \\ q_1' \end{array}\right] \quad \cdots \quad \left[\begin{array}{c} q_n \\ g_n \to R_n \\ q_n' \end{array}\right] \quad \left[\begin{array}{c} p_1 \\ g_1' \to_T R_1' \\ p_1' \end{array}\right] \quad \cdots \quad \left[\begin{array}{c} p_\ell \\ g_\ell' \to_T R_\ell' \\ p_\ell' \end{array}\right]$$

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 $R_1, \ldots, R_n$. Furthermore, for $i \in \{1, \ldots, \ell\}$, each process in state p_i whose clock values satisfy the guard g_i' has to move to p_i' while its clocks are reset according to R_i'. Note that for a rule to be enabled there is no requirement on the presence or absence of processes in states $p_1, \cdots, p_\ell$.

The semantics associated to the TNT $\mathcal{N}$ 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 $\longrightarrow_D$, representing transitions induced by the rules of $\mathcal{N}$ and a *timed* transition relation $\longrightarrow_T$ which characterizes the elapse of time.

We begin by describing the transition relation $\longrightarrow_D \subseteq \mathcal{C}_{\mathcal{N}} \times \mathcal{C}_{\mathcal{N}}$. For this matter, we define a transition relation $\longrightarrow_r$ for each rule $r \in \mathcal{R}$ of the TNT $\mathcal{N}$. 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 $\gamma \longrightarrow_r \gamma'$ 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).

Furthermore, for all $j \in I \setminus \{h(i) \mid i \in \{1, \ldots, n\}\}$:

4. if there exists $i \in \{1, \ldots, \ell\}$ such that $\mathcal{Q}(j) = p_i$ and if $\mathcal{X}(j) \models g'_i$, then $\mathcal{Q}'(j) = p'_i$ and $\mathcal{X}'(j) = \mathcal{X}'(j)[R'_i]$ (i.e. the transfer is applied to such processes forcing them to change their state);

5. otherwise, $\mathcal{Q}'(j) = \mathcal{Q}(j)$ and $\mathcal{X}'(j) = \mathcal{X}(j)$.

We define then the discrete transition relation $\longrightarrow_D = \bigcup_{r \in \mathcal{R}} \longrightarrow_r$.

We now provide the definition of the timed transition relation $\longrightarrow_T \subseteq \mathcal{C}_\mathcal{N} \times \mathcal{C}_\mathcal{N}$. 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 $\gamma \longrightarrow_T \gamma'$ 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 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 advance at the same rate.

Finally we define $\longrightarrow_\mathcal{N}$ to be $\longrightarrow_D \cup \longrightarrow_T$. 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.

3.3.4 State reachability problem

Similarly to the case of TAHN, we present here the state reachability problem for TNT. This problem is again parameterized by the number of processes involved in the execution, and that is why we do not impose any bounds on the size of the initial configurations.

Let $\mathcal{N} = \left(Q^{ctrl}, Q^{proc}, X, \mathcal{R}, q^{init}_{ctrl}, q^{init}_{proc}\right)$ be a TNT. We say that a configuration γ_n in $\mathcal{C}_\mathcal{N}$ is reachable in $\mathcal{N}$ if there exists an 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 $(I, q, \mathcal{Q}, \mathcal{X})$. We now define in the case of TNT the state reachability problem $\texttt{TNT}-\texttt{Reach}\,(K)$ parameterized by the number of clocks K as follows:

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

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

As said earlier, Timed Networks (without Transfer) have already been introduced in [AJ03] where results for the state reachability problem are also presented. We denote by $\texttt{TN}-\texttt{Reach}\,(K)$ the restriction of $\texttt{TNT}-\texttt{Reach}\,(K)$ to Timed Networks. These latter result can be expressed as follows:

Theorem 2. *[AJ03]*

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

As Timed Networks are a restriction of TNT, this allows us to deduce the following undecidability result.

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

We will see in the next section that the decidability result on Timed Networks with 1 single clock per process can be extended to the case of TNT by adapting the proof proposed in [AJ03].

3.3.5 Decidability of $\texttt{TNT}-\texttt{Reach}\,(1)$

We now prove that the decidability result of Theorem 2 can be extended to the case of TNT. The proof of this result is based on the following steps:

1. Define a symbolic way to represent infinite set of configurations.
2. Exhibit a well-quasi-ordering over the symbolic configurations which corresponds to set inclusion on the concrete configurations.
3. Show that it is possible to compute symbolically the predecessors of a symbolic configuration.
4. Give an algorithm which computes all the elements from which a given symbolic configuration can be reached and which terminates thanks to the wqo theory.

We will show here that these different steps developed in the case of Timed Networks can be adapted to the case of TNT.

In the rest of the section we consider a TNT $\mathcal{N} = \left(Q^{ctrl}, Q^{proc}, X, \mathcal{R}, q^{init}_{ctrl}, q^{init}_{proc}\right)$ with $|X| = 1$. Since there is a single clock, we assume a simplified clock mapping for configurations $(I, q, \mathcal{Q}, \mathcal{X})$ of $\mathcal{N}$ of the form $\mathcal{X} : I \mapsto \mathbb{R}^{\geq 0}$.

Symbolic representation of configurations.

First, note that the configurations of TNTs are the same as the ones for TNs, hence we can use the same symbolic representation for configurations as the one presented in [AJ03]. We denote by max the maximal constant occurring in the guards of $\mathcal{N}$. Furthermore, for a quasi order $\sqsubseteq$, we use the notation $a \equiv b$ whenever $a \sqsubseteq b$ and $b \sqsubseteq a$. A symbolic configuration φ for $\mathcal{N}$ is a tuple $(m, q, \mathcal{Q}, \mathcal{K}, \sqsubseteq)$ where:

- m is a natural number such that $\{1, \ldots, m\}$ represents a set of indices for the processes;
- $q \in Q^{ctrl}$ is a controller state;
- $\mathcal{Q} : \{1, \ldots, m\} \mapsto Q^{proc}$ maps indices to process states;
- $\mathcal{K} : \{1, \ldots, m\} \mapsto \{0, \ldots, max\}$ maps process indices to natural numbers smaller than the constant $\max$;
- $\sqsubseteq$ is a quasi-order on the set $\{1, \ldots, m\} \cup \{\bot, \top\}$ such that:
  - $\bot$ and $\top$ are respectively the minimal and maximal elements of $\sqsubseteq$ with $\bot \neq \top$;
  - for $j \in \{1, \ldots, m\}$, if $\mathcal{K}(j) = max$ then $j \equiv \bot$ or $j \equiv \top$;
  - for $j \in \{1, \ldots, m\}$, if $j \equiv \top$ then $\mathcal{K}(j) = max$.

We denote by $\mathcal{S}_{\mathcal{N}}$ the set of symbolic configurations for $\mathcal{N}$. The intuition behind a symbolic configuration $\varphi = (m, q, \mathcal{Q}, \mathcal{K}, \sqsubseteq)$ is that it corresponds to a set of configurations for which the controller is in state q, each process is given an index $j \in \{1, \ldots, m\}$ such that $\mathcal{Q}(j)$ is the state of the process and $\mathcal{K}(j)$ is either the integer part of the clock or max, and, the order $\sqsubseteq$ provides an order of the processes corresponding to the order of the fractional part of their respective clock. 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.

Following [AJ03], we now formalize the previous intuition by providing a formal definition of the set $[\![\varphi]\!]$ of concrete configurations represented by the symbolic configuration φ. Let $\gamma = (I, q, \mathcal{Q}, \mathcal{X})$ be a configuration of $\mathcal{N}$ and $\varphi = (m, q', \mathcal{Q}', \mathcal{K}, \sqsubseteq)$ be a symbolic configuration of $\mathcal{N}$. We have $\gamma \in [\![\varphi]\!]$ if and only if $q = q'$ and there exists an injective function $h : \{1, \ldots, m\} \mapsto I$ such that for all $j, j_1, j_2 \in \{1, \ldots, m\}$:

- $\mathcal{Q}(h(j)) = \mathcal{Q}'(j)$;
- $\min(max, \lfloor \mathcal{X}(h(j)) \rfloor) = \mathcal{K}(j)$ (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 $j_1 \not\equiv \top$ and $j_2 \not\equiv \top$ then $frac(\mathcal{X}(h(j_1))) \leq frac(\mathcal{X}(h(j_2)))$ if and only if $j_1 \sqsubseteq j_2$.

We call such an injective function h a *mapping associated to* $(\gamma, [\![\varphi]\!])$. Note that in the above definition, we do not require the number of processes in γ and in φ to be the same, but only that each process of φ can be matched with a process of γ. For a set of symbolic configurations $\Phi \subseteq \mathcal{S}_{\mathcal{N}}$, we denote by $[\![\Phi]\!]$ the set $\bigcup_{\varphi \in \Phi} [\![\varphi]\!]$.

We now equip the set of symbolic configurations $\mathcal{S}_{\mathcal{N}}$ with a quasi-order $\preceq$. Given two symbolic configurations $\varphi = (m, q, \mathcal{Q}, \mathcal{K}, \sqsubseteq)$ and $\varphi' = (m', q', \mathcal{Q}', \mathcal{K}', \sqsubseteq')$, we have $\varphi \preceq \varphi'$ if and only if $q = q'$ and there exists an injective mapping $g : \{1, \ldots, m\} \mapsto \{1, \ldots, m'\}$ such that for all $j, j_1, j_2 \in \{1, \ldots, m\}$:

- $\mathcal{Q}'(g(j)) = \mathcal{Q}'(j)$;
- $\mathcal{K}'(g(j)) = \mathcal{K}(j)$;
- $g(j) \equiv' \bot$ if and only if $j \equiv \bot$;
- $g(j) \equiv' \top$ if and only if $j \equiv \top$;
- $g(j_1) \sqsubseteq' g(j_2)$ if and only if $j_1 \sqsubseteq j_2$.

We have then the following proposition concerning this order.

Proposition 1. *[AJ03]*

1. *Given $\varphi, \varphi' \in \mathcal{S}_{\mathcal{N}}$, we have $\varphi \preceq \varphi'$ if and only if $[\![\varphi']\!] \subseteq [\![\varphi]\!]$.*
2. *$(\mathcal{S}_{\mathcal{N}}, \preceq)$ is a wqo.*

Computing the symbolic predecessors.

We now show how to compute symbolically the set of predecessors of the configurations described by a symbolic configurations. As in [AJ03], for a symbolic configuration $\varphi \in \mathcal{S}_{\mathcal{N}}$, we will see how to build finite set of symbolic configurations corresponding to the two sets $pre_D(\varphi) = \{\gamma \in \mathcal{C}_{\mathcal{N}} \mid \exists \gamma' \in [\![\varphi]\!].\gamma \longrightarrow_D \gamma'\}$ and $pre_T(\varphi) = \{\gamma \in \mathcal{C}_{\mathcal{N}} \mid \exists \gamma' \in [\![\varphi]\!].\gamma \longrightarrow_T \gamma'\}$. We will now show that it is possible to build a finite set of finite configurations Φ such that $[\![\Phi]\!] = pre_D(\varphi) \cup pre_T(\varphi)$.

First we begin with the predecessors obtained by considering the discrete transition relation. The construction is very similar to the one proposed in [AJ03] with the addition that ours allows

to deal with transfers. Following the construction proposed in this latter work, given a guard $g \in \mathcal{G}(X)$ (we recall that $|X| = 1$), a symbolic configuration $\varphi = (m, q, \mathcal{Q}, \mathcal{K}, \sqsubseteq)$ and a natural $j \in \{1, \ldots, m\}$, we define the relation $\langle \varphi, j \rangle \models g$ inductively as follows:

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

In [AJ03], the following result was provided concerning this satisfiability relation $\models$.

Lemma 1. *[AJ03] Let $\varphi = (m, q, \mathcal{Q}, \mathcal{K}, \sqsubseteq)$ be a symbolic configuration,$\gamma = (I, q, \mathcal{Q}, \mathcal{X})$ be a concrete configuration such that $\varphi \in [\![\gamma]\!]$ and g a guard in $\mathcal{G}(X)$. For all mappings h associated to $(\gamma, [\![\varphi]\!])$ and all $j \in \{1, \ldots, m\}$, we have $\langle \varphi, j \rangle \models g$ if and only if $\mathcal{X}(h(j)) \models g$.*

We now consider a rule r of $\mathcal{N}$ of the following form:

$$\begin{bmatrix} q_0 \\ \rightarrow \\ q_0' \end{bmatrix} \quad \begin{bmatrix} q_1 \\ g_1 \rightarrow R_1 \\ q_1' \end{bmatrix} \quad \cdots \quad \begin{bmatrix} q_n \\ g_n \rightarrow R_n \\ q_n' \end{bmatrix} \quad \begin{bmatrix} p_1 \\ g_1' \rightarrow_T R_1' \\ p_1' \end{bmatrix} \quad \cdots \quad \begin{bmatrix} p_\ell \\ g_\ell' \rightarrow_T R_\ell' \\ p_\ell' \end{bmatrix}$$

and a symbolic configuration $\varphi' = (m', q', \mathcal{Q}', \mathcal{K}', \sqsubseteq')$, we define the set $\texttt{Pre}(r, \varphi')$ of symbolic configurations as follows: a symbolic configuration $\varphi = (m, q, \mathcal{Q}, \mathcal{K}, \sqsubseteq)$ belongs to $\texttt{Pre}(r, \varphi')$ if and only if there exist pairwise disjoint sets $changing$, $unchanged$ and $guarding$ such that $\{1, \ldots, m'\} = changing \cup unchanged$ and $\{1, \ldots, m\} = changing \cup unchanged \cup guarding$, and a bijective function $g : (changing \cup guarding) \mapsto \{1, \ldots, n\}$ such that the following conditions are satisfied:

1. $q' = q_0'$ and $q = q_0$;
2. for all $j \in changing$, we have:
   (a) $\mathcal{Q}(j) = q_{g(j)}$;

(b) $(\varphi, j) \models g_{g(j)}$;

(c) $\mathcal{Q}'(j) = q'_{g(j)}$;

(d) if $R_{g(j)} \neq \emptyset$, $\mathcal{K}'(j) = 0$ and $j \equiv' \bot$

(e) if $R_{g(j)} = \emptyset$, $\mathcal{K}'(j) = \mathcal{K}(j)$, and, $j \equiv \bot$ iff $j \equiv' \bot$, and, $j \equiv \top$ iff $j \equiv' \top$;

3. for all $j \in \mathit{guarding}$, we have:

   (a) $\mathcal{Q}(j) = q_{g(j)}$;

   (b) $(\varphi, j) \models g_j$;

4. for all $j \in \mathit{unchanged}$, we have:

   (a) $\mathcal{Q}'(j) = \mathcal{Q}(j)$;

   (b) there does not exist $i \in \{1, \ldots, l\}$ such that $\mathcal{Q}(j) = p_i$ and $(\varphi, j) \models g'_i$;

5. for all $j, j' \in \{j \in \mathit{changing} \mid R_{g(j)} = \emptyset\} \cup \mathit{unchanged}$, we have $j \sqsubseteq j'$ iff $j \sqsubseteq' j'$.

The intuition behind this construction is the following. We want to witness the processes in $[\![\varphi']\!]$ whose states (control states and clock value) have been changed by the rule r. The indices of such processes are stored in the set $\mathit{changing}$, while in $\mathit{unchanged}$, we store the indices of the processes that are not afflicted by the rule r. Furthermore, since φ' is a symbolic configuration, there might be some processes which are needed to fire the rules r and that are not represented in φ' (but are present in $[\![\varphi']\!]$): the set $\mathit{guarding}$ characterizes such processes. The function g is used to map the indices of the processes partitioned as mentioned with the indices of the processes appearing in the rule r. Not that the only point where this construction differs from the case with no transfer presented in [AJ03] is the condition 4.(b) where we require that in the set of predecessors there should not be an unchanged process in a state which satisfies the conditions of a transfer rule (control state and guard), in fact if such process exists, it should be affected by the transfer rule and hence not remain unchanged. We have then the following lemma which states that the set $\mathtt{Pre}(r, \varphi')$ can be computed since it is finite and the union of the symbolic predecessors for each rule of the network corresponds exactly to the set of predecessors $\mathit{pre}(\varphi)$.

Lemma 2.

1. $\mathtt{Pre}(r, \varphi')$ *is finite.*

2. $\mathit{pre}_D(\varphi) = [\![\bigcup_{r \in \mathcal{R}} \mathtt{Pre}(r, \varphi)]\!]$.

Sketch of proof. We remark that if $\varphi = (m, q, \mathcal{Q}, \mathcal{K}, \sqsubseteq)$ belongs to $\mathtt{Pre}(r, \varphi')$, we have $m \leq m' + n$ thanks to the partitioning and to the fact that the function g is bijective. And by definition of symbolic configurations, we have that there exists a finite set of symbolic configurations of

the form $(m, q, \mathcal{Q}, \mathcal{K}, \sqsubseteq)$ for a fixed m. Hence $\texttt{Pre}(r, \varphi')$ is finite. The proof that $pre_D(\varphi) = [\![\bigcup_{r \in \mathcal{R}} \texttt{Pre}(r, \varphi)]\!]$ follows the same line as the one for the case without transfer that can be found in Appendix of [AJ03]. □

Note that for what concerns the set $pre_T(\varphi)$, the rules of the systems are not taken into account and hence in the case of TNT computing this set is exactly the same as in Timed Networks. We can consequently reuse the following result proved in [AJ03].

Lemma 3. *[AJ03] There exists a computable finite set of symbolic configurations Φ such that $[\![\Phi]\!] = pre_T(\varphi)$.*

According to the two previous Lemma, for a symbolic configuration φ we define the finite set of symbolic configurations $\texttt{Pre}(\varphi)$ as the set such that $[\![\texttt{Pre}(\varphi)]\!] = pre_D(\varphi) \cup pre_T(\varphi)$.

Solving $\texttt{TNT}-\texttt{Reach}\,(1)$.

We now describe an algorithm, similar to the one in [AJ03], in order to solve $\texttt{TNT}-\texttt{Reach}\,(1)$. Let $q_F \in Q^{ctrl}$ be a controller state, and let $\Phi(q_F)$ denote the set $\{(m, q, \mathcal{Q}, \mathcal{K}, \sqsubseteq) \in \mathcal{S}_{\mathcal{N}} \mid m = 0 \wedge q = q_F\}$. Note that this set is a singleton by definition of $\mathcal{S}_{\mathcal{N}}$.

Algorithm 1 *Reachable*$(\mathcal{N}, q_F)$

Require: A TNT $\mathcal{N} = \left(Q^{ctrl}, Q^{proc}, X, \mathcal{R}, q^{init}_{ctrl}, q^{init}_{proc}\right)$ with $|X| \leq 1$ and $q_F \in Q^{ctrl}$
Ensure: YES if q_F is reachable in $\mathcal{N}$, NO otherwise

$W = \Phi(q_F)$
$V = \emptyset$
while $W \neq \emptyset$ **do**
  Take φ out of W
  if $[\![\varphi]\!]$ contains an initial configuration **then**
    Return YES
  else if $\nexists \varphi' \in V$ such that $\varphi' \preceq \varphi$ **then**
    $V := V \cup \{\varphi\}$
    $W := W \cup \texttt{Pre}(\varphi)$
  end if
end while
Return NO

Proposition 2. *Given as input a TNT $\mathcal{N} = \left(Q^{ctrl}, Q^{proc}, X, \mathcal{R}, q^{init}_{ctrl}, q^{init}_{proc}\right)$ with $|X| \leq 1$ and $q_F \in Q^{ctrl}$, Algorithm 1 terminates and returns* YES *iff q_F is reachable in $\mathcal{N}$.*

Proof. First we prove the termination of the Algorithm 1 by reasoning by contradiction. If this algorithm on input $\mathcal{N} = \left(Q^{ctrl}, Q^{proc}, X, \mathcal{R}, q^{init}_{ctrl}, q^{init}_{proc}\right)$ with $|X| \leq 1$ and $q_F \in Q^{ctrl}$ does

not terminate, because of the test performed at Line 7, we deduce that there exists an infinite sequence of symbolic configurations $(\varphi_i)_{i\in\mathbb{N}}$ such that for all $j \in \mathbb{N}$ there does not exist $i \in \mathbb{N}$ verifying $i < j$ and $\varphi_i \preceq \varphi_j$. But thanks to Proposition 1 we know that $(\mathcal{S}_{\mathcal{N}}, \preceq)$ is a wqo, hence it is not possible to have an infinite sequence of symbolic configurations respecting the previous property. This allows us to deduce the termination of Algorithm 1.

We now prove the correction of Algorithm 1. First we assume that this algorithm on input $\mathcal{N} = \left(Q^{ctrl}, Q^{proc}, X, \mathcal{R}, q^{init}_{ctrl}, q^{init}_{proc}\right)$ with $|X| \leq 1$ and $q_F \in Q^{ctrl}$ returns YES. This means that there exists a finite sequence of symbolic configurations $(\varphi_i)_{1\leq i\leq \ell}$ such that $\varphi_1 \in \Phi(q_F)$, $[\![\varphi_\ell]\!]$ contains an initial configuration and $\varphi_{i+1} \in \mathtt{Pre}(\varphi_i)$ for all $i \in \{1, \ldots, \ell - 1\}$. Thanks to the definition of the operator Pre and according to the results of Lemmas 2 and 3, we deduce that there exists a sequence of configurations $(\gamma_i)_{1\leq i\leq \ell}$ verifying the following properties: $\gamma_i \in [\![\varphi_i]\!]$ for all $i \in \{1, \ldots, \ell\}$, γ_ℓ is an initial configuration and $\gamma_{i+1} \longrightarrow_{\mathcal{N}} \gamma_i$ for all $i \in \{1, \ldots, \ell - 1\}$. Since $\gamma_1 \in [\![\varphi_1]\!]$ and since $\gamma_1 \in \Phi(q_F)$, according to the definition of the operator $[\![\cdot]\!]$, we deduce that γ_1 is of the form $(I, q_F, \mathcal{Q}, \mathcal{X})$. Consequently q_F is reachable in $\mathcal{N}$.

We now assume that the algorithm on input $\mathcal{N} = \left(Q^{ctrl}, Q^{proc}, X, \mathcal{R}, q^{init}_{ctrl}, q^{init}_{proc}\right)$ with $|X| \leq 1$ and $q_F \in Q^{ctrl}$ returns NO. We denote by V_{end} the content of the set V when the algorithm terminates. Given a set of symbolic configurations Φ, we define the set $pre^*(\Phi)$ as the set of configurations γ for which there exists a finite sequence of configurations $(\gamma_i)_{1\leq \ell}$ verifying $\gamma_1 = \gamma$, $\gamma_\ell \in [\![\Phi]\!]$ and $\gamma_i \longrightarrow_{\mathcal{N}} \gamma_{i+1}$ for all $i \in \{1, \ldots, \ell-1\}$. We will prove that if $\gamma \in pre^*(\Phi(q_F))$ then $\gamma \in [\![V_{end}]\!]$. Let $\gamma \in pre^*(\Phi(q_F))$ and let $(\gamma_i)_{1\leq \ell}$ be the associated sequence of configurations. We reason by induction on ℓ. Assume $\ell = 1$, we have then $\gamma = \gamma_\ell = \gamma_1$ and hence $\gamma \in [\![\Phi(q_F)]\!]$. Note that if the algorithm returns NO, the symbolic configurations included in $\Phi(q_F)$ is included in V_{end}. Consequently we have $\gamma \in [\![V_{end}]\!]$. We assume now that $\ell > 1$. By induction hypothesis, we have for all $i \in \{2, \ldots, \ell\}$, we have $\gamma_i \in V_{end}$ consequently for all $i \in \{2, \ldots, \ell\}$ there exist a symbolic configuration φ_i such that $\varphi_i \in V_{end}$ and $\gamma_i \in [\![\varphi_i]\!]$. But then we can deduce that for all $\varphi \in \mathtt{Pre}(\varphi_2)$, either $\varphi \in V_{end}$ or there exists a symbolic configuration $\varphi' \in V_{end}$ such that $\varphi' \preceq \varphi$. This because when φ_2 is added to V, $\mathtt{Pre}(\varphi_2)$ is added to W and since the algorithm returns NO, when the algorithm terminates, the set W has been emptied and according to the test on Line 7 some of the symbolic configurations contained in it have been added to V. Furthermore thanks to to the definition of the operator Pre and according to the results of Lemmas 2 and 3, we know that there exists $\varphi_1 \in \mathtt{Pre}(\varphi_2)$ such that $\gamma_1 \in [\![\varphi_1]\!]$. Note that If $\varphi_1 \in V_{end}$ then we have $\gamma_1 \in [\![V_{end}]\!]$. Otherwise if $\varphi_1 \notin V_{end}$, then there exists $\varphi'_1 \in V_{end}$ such that $\varphi'_1 \preceq \varphi_1$, but in that case we have thanks to Proposition 1 $[\![\varphi_1]\!] \subseteq [\![\varphi'_1]\!]$ and consequently $\gamma_1 \in [\![V_{end}]\!]$. This allows us to deduce that if $\gamma \in pre^*(\Phi(q_F))$ then $\gamma \in [\![V_{end}]\!]$. Now if q_F is reachable in $\mathcal{N}$, it means that there is an initial configuration in $pre^*(\Phi(q_F))$ and consequently $[\![V_{end}]\!]$ but this is not possible according to the test performed at Line 5. Hence q_F is reachable in $\mathcal{N}$. □

This proposition allows us to state our main result about Timed Networks with Transfers where each process is equipped with a single clock.

Theorem 3. $\texttt{TNT}-\texttt{Reach}\,(1)$ *is decidable.*

3.4 Undecidability with Dense Time

In this section we show undecidability of the reachability problem for three classes of TAHNs, namely:

- $\texttt{STAR}(2)$: *star* topologies of depth 2 (one root and several rays with two nodes each) and with a single clock in each node;
- $\texttt{CLIQUE}$: *clique* topologies with two clocks per node; and
- $\texttt{BOUNDED}(5)$: *bounded path* topologies with no simple paths counting more than 5 nodes and with a single clock in each node.

In the first two cases, the undecidability result is shown through a reduction from $\texttt{TN}-\texttt{Reach}\,(2)$ (wich is undecidable by Theorem 2). In each case we will simulate a TN $\mathcal{N}$ with two clocks per process by a TAHN $\mathcal{T}$.

3.4.1 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 TAHN $\mathcal{T}$. 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 $\mathcal{T}(q)$ in $\mathcal{T}$. Furthermore, we will have a number of auxiliary states in $\mathcal{T}$ that we need to perform the simulation. As a convention, we omit state labels in the automata every time we want fresh ones.

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 $\mathcal{T}(q)$ verifying that there exists a graph $G \in \texttt{STAR}(2)$ such that q is reachable in $\mathcal{T} = \langle P, G\rangle$ iff q is reachable in $\mathcal{N}$. The root of $\mathcal{T}$ plays the role of the controller in $\mathcal{N}$, while each ray in $\mathcal{T}$ plays the role of one process in $\mathcal{N}$. The local state of a process in $\mathcal{T}$ 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 ray node and by the clock of the leaf of the ray. For technical reasons, we require that $\mathcal{T}$ has at least three rays. 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. 3.2, 3.3, and 3.4), 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 due 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 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.

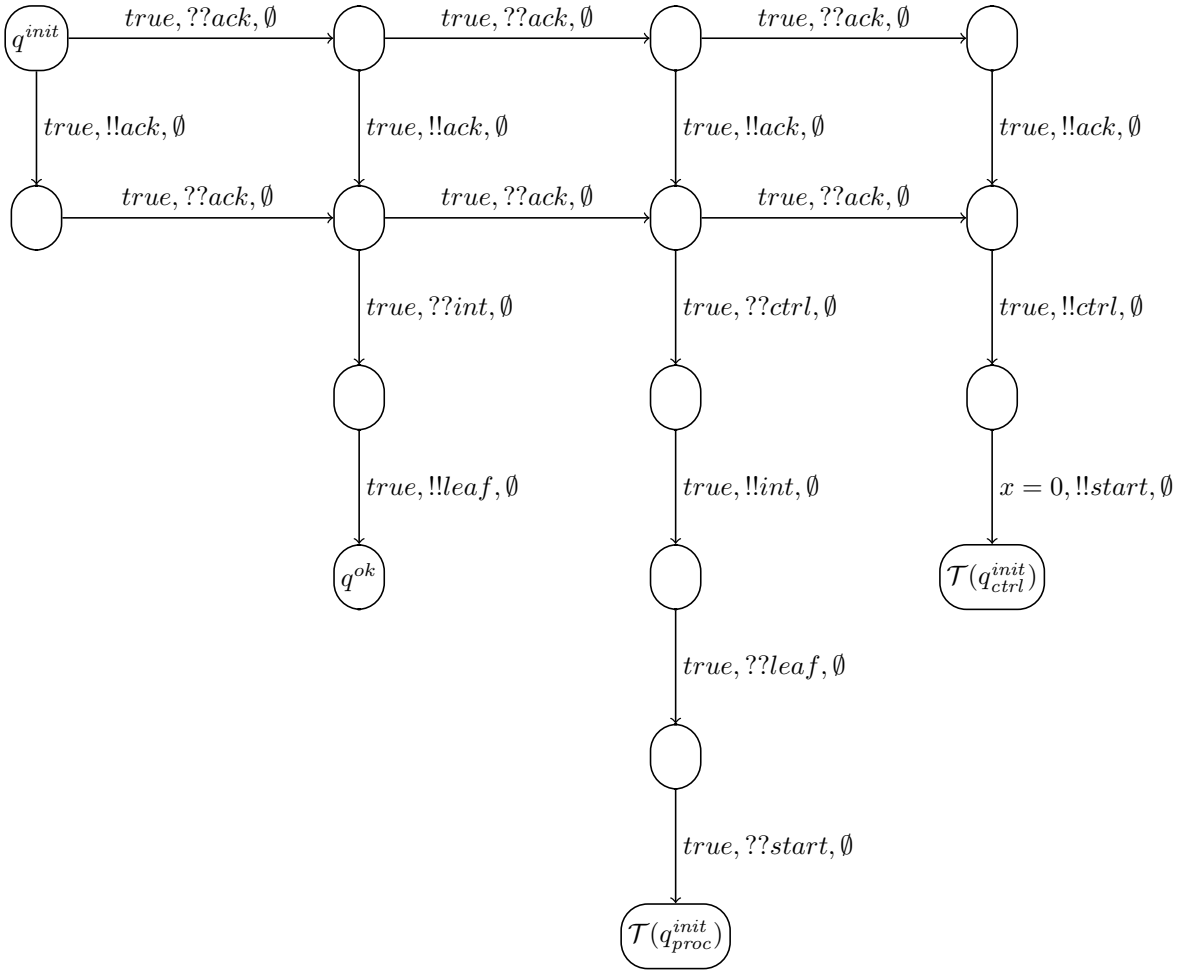


Figure 3.1: Initializing the simulation

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. 3.1) 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. 3.1, a node starts by broadcasting/receiving an ack message to/from his neighbors. ack messages 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 ray 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 a 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.

(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. A failure in a ray will bring the simulation into a deadlock state.

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

(i) It changes local state to $\mathcal{T}(q_{ctrl}^{init})$, 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 $\mathcal{T}(q_{proc}^{init})$. The rest of the nodes will remain passive throughout the rest of the simulation.

Now all the nodes are ready: the root of $\mathcal{T}$ 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 $\mathcal{N}$, and all clocks have values equal to 0.

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

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

The behaviour of the root, internal, and leaf nodes is detailed respectively in Fig. 3.2, Fig. 3.3, and Fig. 3.4. At first, the root of $\mathcal{T}$ is in the state $\mathcal{T}(q_0)$ and executes a transition to reset its clock to 0 (this is done so that it can later make sure that the simulation of the rule did not take 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 each of x_1 and x_2 separately. 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 $\mathcal{T}(q_1')$.

(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 to play the role of the first process, the root enters en error state if it happens to receive acknowledgements from several internal nodes.

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 $\mathcal{T}(q_0')$. (ii) It sends a final broadcast where the node ready for simulating the i^{th} process will now move to $\mathcal{T}(q_i')$ 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 did not take any time).

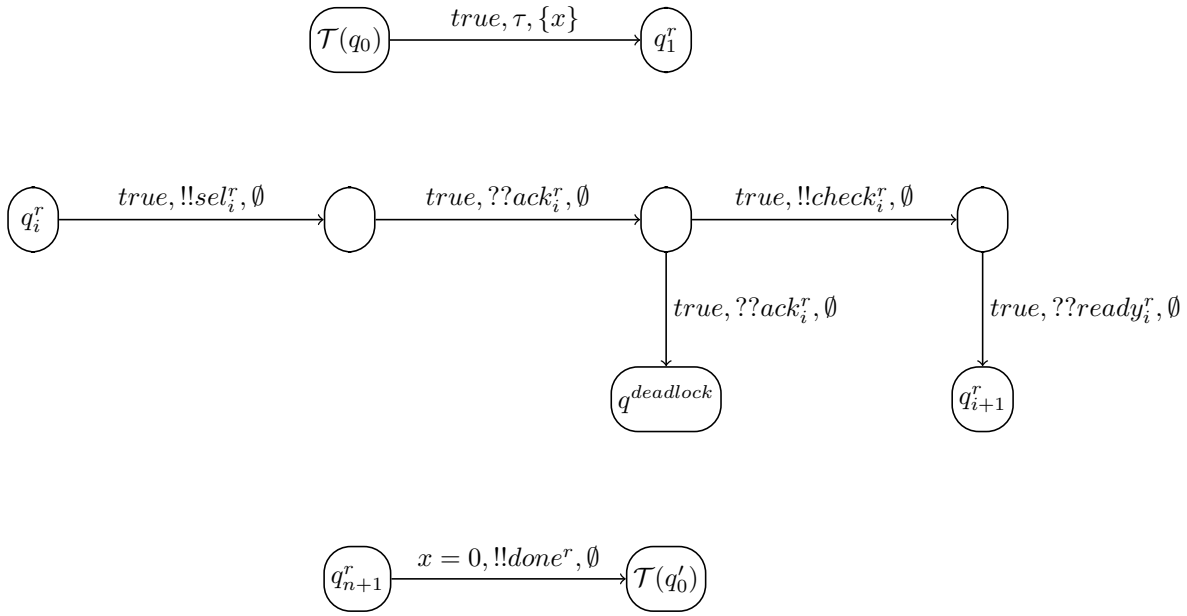


Figure 3.2: Ray selection: root node

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

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

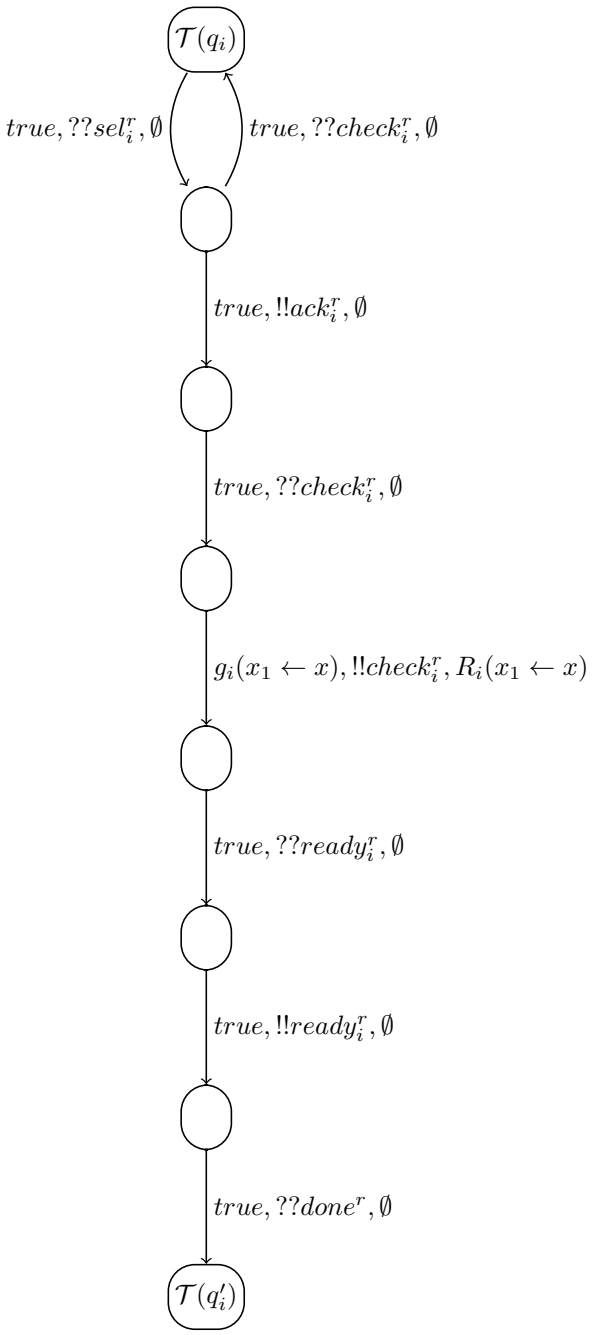


Figure 3.3: Ray selection: internal node

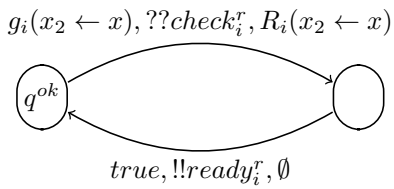


Figure 3.4: Ray selection: leaf node

Theorem 4. $\texttt{TAHN}-\texttt{Reach}\,(\texttt{STAR}(2), 1)$ *is undecidable.*

3.4.2 Cliques and Nodes with Two Clocks

We now show that the reachability problem for the clique topology is undecidable if nodes have two clocks. For this purpose, we build a protocol P with $nbclocks(P) = 2$ which will simulate $\mathcal{N}$ on 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 $\mathcal{T}(q_{ctrl}^{init})$, while all the other nodes will move to $\mathcal{T}(q_{proc}^{init})$.

Simulating Discrete Transitions Below, we show how a rule of the form of the previous subsection is simulated. In a similar manner to the case of stars, the controller node first reset 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 to try to 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 $\mathcal{T}(q_1')$. To prevent multiple nodes to play the role of the first process, the controller node enters en error state if it happens to receive acknowledgements from several nodes. 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 5. $\texttt{TAHN}-\texttt{Reach}\,(\texttt{CLIQUE}, 2)$ *is undecidable.*

3.4.3 Bounded Path Topologies

Using the result of Theorem 4 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 simple paths can have 5 vertices 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 an arbitrary graph in paths are allowed to have (at least) 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 internal transitions labelled with empty event that non-deterministically select the role of a each node: the root (control state q_0), a ray (internal) node (control state r_0) or a leaf (control state s_0) of the star topology. These three rules have following form:

$$\langle q^{init}, x = 0 \xrightarrow{\tau} \emptyset, q_0 \rangle$$
$$\langle q^{init}, x = 0 \xrightarrow{\tau} \emptyset, r_0 \rangle$$
$$\langle q^{init}, x = 0 \xrightarrow{\tau} \emptyset, s_0 \rangle$$

The behavior of the root node (state q_0) is given in Figure 3.5. It broadcasts message $root$ to notify its neighbors that it is the root. A node in state q_0 moves to an error state if it receives notifications/requests from other nodes. The combination of the two types of rules 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 message $root$, a node in state r_0 runs the protocol in Figure 3.6. Specifically, it first reacts by sending $ackroot$. This message is needed to send all of its neighbors 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 a ray node is never connected to two different root nodes. Upon reception of message $endroot$ from the root, it 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.

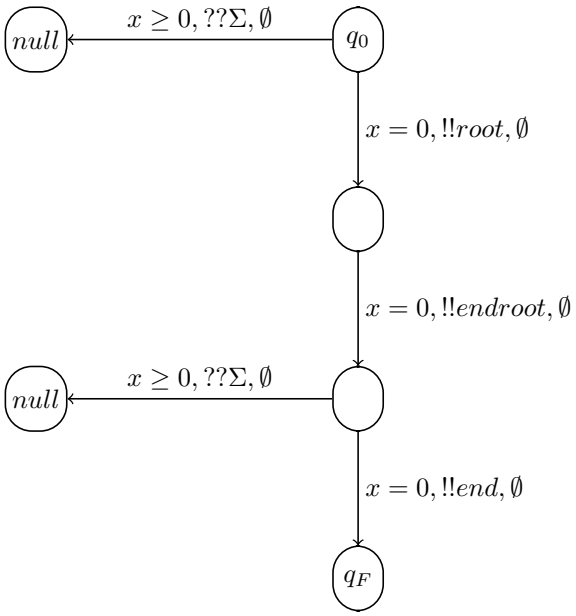


Figure 3.5: Discovery protocol: node chosen as the root

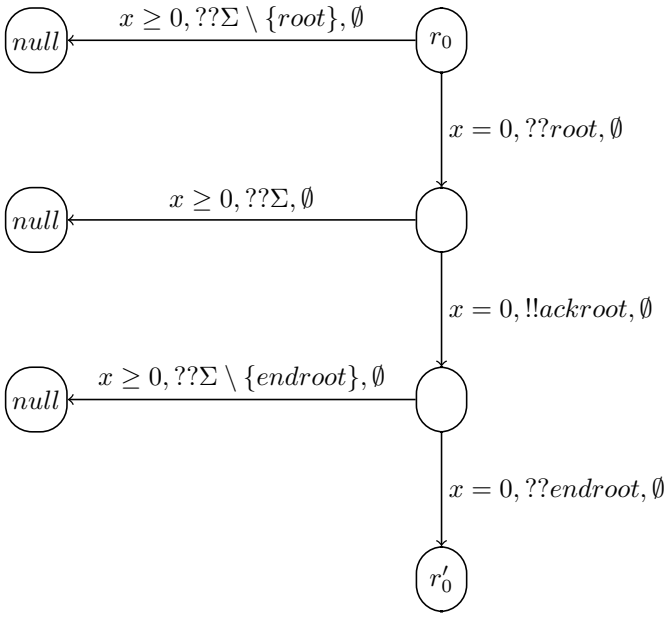


Figure 3.6: Discovery protocol: node chosen as a ray (communication with the root)

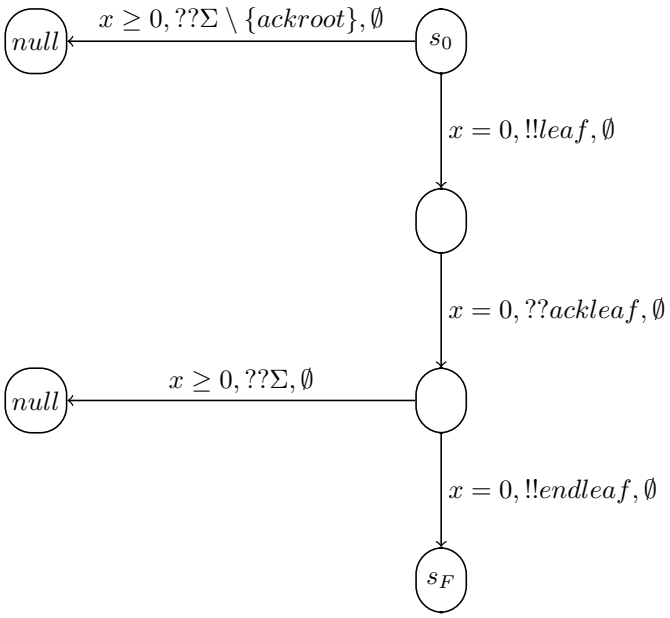


Figure 3.7: Discovery protocol: node chosen as a leaf

Figures 3.7 and 3.8 show the handshaking protocol between leaf and ray nodes. A node in state s_0 sends a message $leaf$ to its adjacent nodes. A ray 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.

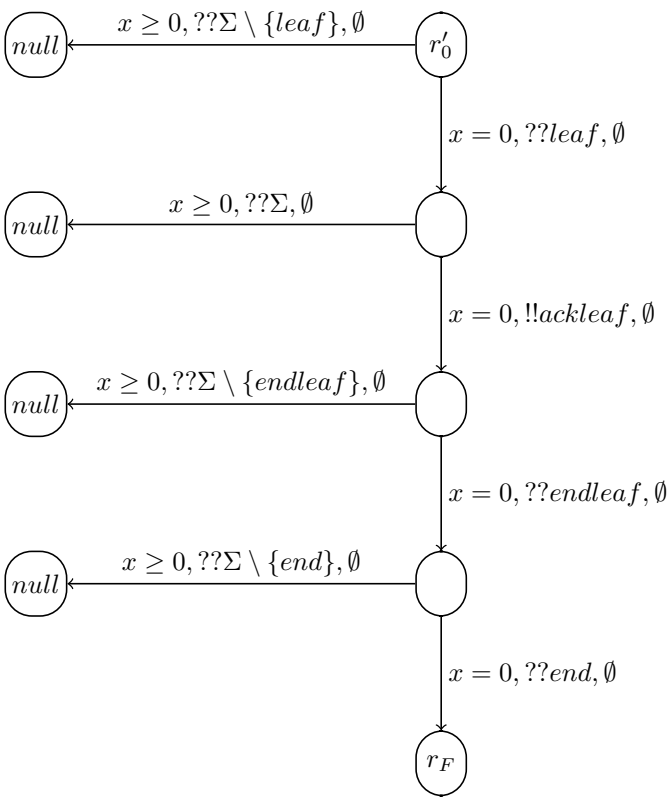


Figure 3.8: Discovery protocol: node chosen as a ray (communication with the leaf)

By construction, the following properties then holds.

Proposition 3. *We have that if $\mathcal{T} = \langle P, G \rangle$ is a TAHN such that $G \in \texttt{BOUNDED}(5)$, then all configurations $\gamma = \langle \mathcal{Q}, \mathcal{X} \rangle$ reachable in $\mathcal{T}$ satisfy the following properties:*

- *for all vertices $v \in V$ 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 vertices $v_1, v_2 \in V$ 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 vertices $v' \in V \setminus \{v_1, v_2\}$, $v' \sim v$ 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 $\mathcal{T}$ then all the nodes in the state q_F can be seen as the root node of a star of depth 2 where the rays are in state r_F, the leaves are in state q_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 4, we can now simulate the behavior of a $\mathcal{N}$ 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 6. $\texttt{TAHN}-\texttt{Reach}\,(\texttt{BOUNDED}(5), 1)$ *is undecidable.*

Proof. We reduce reachability of a TN $\mathcal{N}$ with two clocks. To this purpose, we define a new protocol P' that combines the discovery protocol and the simulation protocol described in the proof of Theorem 4. 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 4, we know there exists $\mathcal{T}'$ restricted to topologies in $\texttt{STAR}(2)$ that can correctly simulate the $\mathcal{N}$.
- From Proposition 3, any star of depth 2 can be obtained with our pre-protocol. Furthermore, the protocol guarantees the existence of an initial configuration from which we can mark (using states q_F, r_F, and s_F) a subgraph in $\texttt{STAR}(2)$.

Combining the two properties, we deduce that the there exists a $\mathcal{T}$ restricted to topologies in $\texttt{BOUNDED}(5)$ that can correctly simulate a TN $\mathcal{N}$ with two clocks per node. □

3.5 Decidability with Dense Time

We showed previously that $\texttt{TAHN}-\texttt{Reach}\,(\texttt{STAR}(2),1)$ is undecidable. In this section we consider two other classes of topologies for which reachability becomes decidable when nodes have a single clock, namely the class $\texttt{STAR}(1)$ and $\texttt{CLIQUE}$. A convenient way to prove these results is to resort to the extension of Timed Networks with Transfers presented in Section 3.3 for which control state reachability is decidable when each process is equipped with a single clock, as stated by Theorem 3.

We prove next that in TAHNs with one clock restricted to the clique topology, the reachability problem is decidable.

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

Proof. We reduce $\texttt{TAHN}-\texttt{Reach}\,(\texttt{CLIQUE},1)$ to $\texttt{TNT}-\texttt{Reach}\,(1)$. The reasoning works as follows. Since in a clique graph all nodes are connected to each other, a broadcast message sent by a node is always received by all other nodes. In other words working with a clique is like working with a multiset of processes as in a TNT. Broadcast communication can then be simulated by using TNT rules in which the sender perfoms an individual step and reception of a message is simulated by transfer actions, one for each state in which the message can be received. Furthermore, we can insert a special rule to synchronize the controller with the local state we want to reach in the TAHN.

Formally, let $P = \langle Q, X, \Sigma, \mathcal{R}, q^{init}\rangle$ be a protocol with $nbclocks(P) = 1$, and let $q^{init}, q_F \in Q$ be respectively the initial and the target state. Without loss of generality we assume that if $\langle q_1, g_1 \xrightarrow{??a} R_1, q_1'\rangle$ and $\langle q_2, g_2 \xrightarrow{??a} R_2, q_2'\rangle$ are two different rules in $\mathcal{R}$ then $q_1 \neq q_2$ (in other words from each local state there is at most one rule per message a labelled with $??a$). Note that this restriction can easily be obtained by adding empty event rules to the model. We now show how to build a Timed Network with Transfer $\mathcal{N} = \left(Q^{ctrl}, Q^{proc}, X, \mathcal{R}', q^{init}_{ctrl}, q^{init}_{proc}\right)$ with 1 clock which will simulate the behavior of P. First the set of controller states Q^{ctrl} is equal to $\{q_0^c, q_1^c\}$ (with $\{q_0^c, q_1^c\} \cap Q = \emptyset$) and the set of process states Q^{proc} is equal to Q. We assume that q_0^c is the initial controller state and q^{init} is the initial process state. The finite set of rules $\mathcal{R}'$ is the smallest set satisfying the following conditions:

- For each rule $\langle q, g \xrightarrow{\tau} R, q'\rangle$ in $\mathcal{R}$ there is a rule in $\mathcal{R}'$ of the form:

$$\begin{bmatrix} q_0^c \\ \rightarrow \\ q_0^c \end{bmatrix} \quad \begin{bmatrix} q \\ g \rightarrow R \\ q' \end{bmatrix}$$

- For each rule $\langle q, g \xrightarrow{!!a} R, q' \rangle$ in $\mathcal{R}$, there is a rule in $\mathcal{R}'$ of the form:

$$\begin{bmatrix} q_0^c \\ \rightarrow \\ q_0^c \end{bmatrix} \quad \begin{bmatrix} q \\ g \rightarrow R \\ q' \end{bmatrix} \quad \begin{bmatrix} p_1 \\ g_1 \rightarrow_T R_1 \\ p_1' \end{bmatrix} \quad \cdots \quad \begin{bmatrix} p_\ell \\ g_\ell \rightarrow_T R_\ell \\ p_\ell' \end{bmatrix}$$

  such that the set $\{\langle p_i, g_i \xrightarrow{??a} R_i, p_i' \rangle \mid 1 \leq i \leq \ell\}$ is exactly the subset of $\mathcal{R}$ containing all the rules of the form $\langle q_0, g' \xrightarrow{??a} R', q_0' \rangle$. Intuitively, this rule selects non deterministically a node which will perform a broadcast and then simulate the broadcast using the transfer.

- There is a rule in $\mathcal{R}'$ of the form:

$$\begin{bmatrix} q_0^c \\ \rightarrow \\ q_1^c \end{bmatrix} \quad \begin{bmatrix} q_F \\ (0 \leq x) \rightarrow \emptyset \\ q_F \end{bmatrix}$$

  Intuitively this rule allows the controller to detect that one of the process has reached the state q_F.

With this construction and using the semantics of both TAHN and TN with transfer nodes, one can easily prove that there is a TAHN $\mathcal{T} = \langle P, G \rangle$ such that $G \in \texttt{CLIQUE}$ and q_F is reachable in $\mathcal{T}$ iff the controller state q_1^c is reachable in $\mathcal{N}$. Hence, since $\mathcal{N}$ works over one clock, by using Theorem 3 we obtain the desired result. □

A similar positive result can be obtained for TAHN with 1 clock restricted to the star topology of depth 1. The main difference compared to the previous result is that in a star of depth 1 we have to distinguish the root (the central node) from the leaves. When the root performs a broadcast, it is transmitted to all the leaf nodes, but when a leaf performs one, only the root can receive it.

Theorem 8. $\texttt{TAHN}-\texttt{Reach}\,(\texttt{STAR}(1), 1)$ *is decidable.*

Proof. We reduce $\texttt{TAHN}-\texttt{Reach}\,(\texttt{STAR}(1), 1)$ to $\texttt{TNT}-\texttt{Reach}\,(1)$. Let $P = \langle Q, X, \Sigma, \mathcal{R}, q^{init} \rangle$ be the protocol with $nbclocks(P) = 1$ of a TAHN $\mathcal{T}$ with star topology of depth 1. We construct a TNT $\mathcal{N} = \left(Q^{ctrl}, Q^{proc}, X, \mathcal{R}', q_{ctrl}^{init}, q_{proc}^{init}\right)$ with 1 clock which will simulate the behavior of $\mathcal{T}$ as follows. Let $q^{init} \in Q$ be the initial state, and let $q_F \in Q$ be the target state for the reachability problem. As for the proof of Theorem 7 we assume that from each local state there is at most one rule per message a labelled with $??a$. First the set of controller states Q^{ctrl} is equal to $\{q_0^c, q_1^c, q_2^c\}$ (with $\{q_0^c, q_1^c\} \cap Q = \emptyset$) and the set of process states Q^{proc} is equal to $Q \cup Q_r$ where $Q_r = \{q_r \mid q \in Q\}$. The set Q_r will be used to distinguish the root from the leaves. We assume that q_0^c is the initial controller state and q^{init} is the initial process state for $\mathcal{N}$. The finite set of rules $\mathcal{R}'$ is the smallest set satisfying the following conditions:

- There is a rule in $\mathcal{R}'$ of the form:

$$\begin{bmatrix} q_0^c \\ \to \\ q_1^c \end{bmatrix} \quad \begin{bmatrix} q^{init} \\ (0 = x) \to \emptyset \\ q_r^{init} \end{bmatrix}$$

  Intuitively with this rule the timed network with broadcast selects non deterministically a node which will be the root.

- For each rule $\langle q, g \xrightarrow{\tau} R, q' \rangle$ in $\mathcal{R}$ there is a rule in $\mathcal{R}'$ of the form (local action of a leaf):

$$\begin{bmatrix} q_1^c \\ \to \\ q_1^c \end{bmatrix} \quad \begin{bmatrix} q \\ g \to R \\ q' \end{bmatrix}$$

  and a rule of the form (local action of the root):

$$\begin{bmatrix} q_1^c \\ \to \\ q_1^c \end{bmatrix} \quad \begin{bmatrix} q_r \\ g \to R \\ q'_r \end{bmatrix}$$

- For each rule $\langle q, g \xrightarrow{!!a} R, q' \rangle$ in $\mathcal{R}$, there is a rule in $\mathcal{R}'$ of the form (broadcast from the root):

$$\begin{bmatrix} q_1^c \\ \to \\ q_1^c \end{bmatrix} \quad \begin{bmatrix} q_r \\ g \to R \\ q'_r \end{bmatrix} \quad \begin{bmatrix} p_1 \\ g_1 \to_T R_1 \\ p'_1 \end{bmatrix} \quad \cdots \quad \begin{bmatrix} p_\ell \\ g_\ell \to_T R_\ell \\ p'_\ell \end{bmatrix}$$

  such that the set $\{\langle p_i, g_i \xrightarrow{??a} R_i, p'_i \rangle \mid 1 \leq i \leq \ell\}$ is exactly the subset of $\mathcal{R}$ containing all the rules of the form $\langle q_0, g' \xrightarrow{??a} R', q'_0 \rangle$.

- For each rule $\langle q, g \xrightarrow{!!a} R, q' \rangle$ in $\mathcal{R}$, there is a rule in $\mathcal{R}'$ of the form (broadcast from the root):

$$\begin{bmatrix} q_1^c \\ \to \\ q_1^c \end{bmatrix} \quad \begin{bmatrix} q \\ g \to R \\ q' \end{bmatrix} \quad \begin{bmatrix} p_{1,r} \\ g_1 \to_T R_1 \\ p'_{1,r} \end{bmatrix} \quad \cdots \quad \begin{bmatrix} p_{\ell,r} \\ g_\ell \to_T R_\ell \\ p'_{\ell,r} \end{bmatrix}$$

  such that the set $\{\langle p_i, g_i \xrightarrow{??a} R_i, p'_i \rangle \mid 1 \leq i \leq \ell\}$ is exactly the subset of $\mathcal{R}$ containing all the rules of the form $\langle q_0, g' \xrightarrow{??a} R', q'_0 \rangle$. Note that in this case only the root will receive the broadcast and we still use the transfer in order to be sure that if it can receive it, it will change its state. However only one of the transfer will be done since there is only one node labelled by a state in Q_r.

- There is a rule in $\mathcal{R}'$ of the form (a leaf reaches state q_F):

$$\begin{bmatrix} q_1^c \\ \rightarrow \\ q_2^c \end{bmatrix} \quad \begin{bmatrix} q_F \\ (0 \leq x) \rightarrow \emptyset \\ q_F \end{bmatrix}$$

  and a rule of the form (the root reaches q_F):

$$\begin{bmatrix} q_1^c \\ \rightarrow \\ q_2^c \end{bmatrix} \quad \begin{bmatrix} q_{F,r} \\ (0 \leq x) \rightarrow \emptyset \\ q_{F,r} \end{bmatrix}$$

  Intuitively this rule allows the controller to detect that one of the process has reached the state q_F.

With this construction and using the semantics of both TAHN and TN with transfer nodes, one can easily prove that there is a TAHN $\mathcal{T} = \langle P, G \rangle$ such that $G \in \texttt{STAR}(1)$ and q_F is reachable in $\mathcal{T}$ iff the controller state q_2^c is reachable in $\mathcal{N}$. Hence, since as before $\mathcal{N}$ works over one clock, by using Theorem 3 we obtain the desired result. □

3.6 Decidability with Discrete Time

In this section we study the reachability problem for Discrete Time Ad Hoc Networks (DTAHN). In this model clocks range over the natural numbers instead of the reals. Let μ' be the maximum constant used in the protocol rules and let $\mu = \mu' + 1$. When using discrete time, it is enough to restrict the evaluation of clocks to the finite range $I = [0, \mu]$. This follows from the fact that guards of the form $x > c$ remain enabled when time passes once the clock associated to variable x reaches a value greater or equal to μ, while guards of the form $x \leq c$ remain disabled. Therefore, beyond μ we do not need todistinguish between different values for the same clock (see e.g. [ADM04]). For every DTAHN $\mathcal{T}$, we can then define a finite-state process that describes the behavior of a node. We use $\mathcal{C}_{\mu,K}$ to denote configurations over undirected graphs with labels in $Q \times I^K$, where Q is the set of local states of a process, I is the interval $[0, \mu]$, and K is the number of clocks in each process.

In order to simplify the handling of transition guards, without lack of generality for a set X of clocks we encode protocol guards g over X with guards in a subset of $\mathcal{G}(X)$, namely the subset of all conjunctive formulae of the form $\bigwedge_{x \in X} x \geq a_x^g \wedge x \leq b_x^g$, with every $a_x, b_x \in [0, \mu]$. Clocks x always have an explicit lower and upper bound, eventually $x \geq 0$ or $x \leq \mu$ in case there are not any constraints on its current value. Since we already restricted clock values to the interval $[0, \mu]$, this does not affect the semantics. Disjunctions and negations may be encoded via multiple rules between the same two states of the (original) guarded transition rule.

The transition relation $\rightharpoonup_{\mathcal{T}}$ is obtained from that of TAHN by assuming that the evaluation of clock variables is a function $\mathcal{X} : V \mapsto [X \to I]$ and by replacing the time step by the discrete time step defined as follows.

Discrete Time Step: For configurations $\gamma = \langle \mathcal{Q}, \mathcal{X} \rangle$ and $\gamma' = \langle \mathcal{Q}', \mathcal{X}' \rangle$, we write $\gamma \rightharpoonup_{\mathcal{T}} \gamma'$ if, 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.

For a topology class Top and $K \geq 0$, the control state reachability problem $\texttt{DTAHN}-\texttt{Reach}\,(Top, K)$ is the natural reformulation of the one defined for TAHN.

We show next that reachability is decidable when restricting the topology to the class of bounded path graphs $\texttt{BOUNDED}(N)$ for any fixed $N > 1$. The decision procedure is obtained by resorting to the theory of well-structured transition systems [ACJT96]. The procedure is based on a symbolic backward exploration algorithm in which we use *constraints* to finitely represent sets of configurations of *variable size* taken from the class $\texttt{BOUNDED}(N)$ (the configurations may potentially belong to different TAHNs).

Ordering

We first introduce the following ordering between configurations of variable size. Given configurations $\gamma = \langle \mathcal{Q}, \mathcal{X} \rangle$ defined over $G = \langle V, E \rangle$ and $\gamma' = \langle \mathcal{Q}', \mathcal{X}' \rangle$ defined over $G' = \langle V', E' \rangle$, $\gamma \preceq \gamma'$ 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 $(h(u), h(u')) \in E'$;
- $\forall u \in V$, $\mathcal{Q}(u) = \mathcal{Q}'(h(u))$ and $\mathcal{X}(u) = \mathcal{X}'(h(u))$.

An upward closed set U satisfies the property that $U = \{\gamma' \mid \gamma \preceq \gamma',\ \gamma \in U\}$. The following property then holds.

Proposition 4. *If U is an upward closed set of configurations (of variable size), then $Pre(U) = \{\gamma \mid \gamma \rightharpoonup_{\mathcal{T}} \gamma',\ \gamma' \in U\}$ is still upward closed w.r.t. $\preceq$.*

Proof. In [DSZ10] it has been proven that, for untimed AHN, selective broadcast is monotone w.r.t. $\preceq$. We observe that DTAHN restricted to configurations in $\mathcal{C}_{\mu,K}$ can be viewed as untimed AHN extended with a time step transition. Thus, we just have to show monotonicity w.r.t. a time

step $\gamma \rightharpoonup_{\mathcal{T}} \gamma'$. Since time can always proceed, for every $\beta \succeq \gamma$ there is a configuration β' such that $\beta \rightharpoonup_{\mathcal{T}} \beta'$ with a time step. Now since h is label preserving both time steps will have the same effect on nodes identified by h and thus $\gamma' \preceq \beta'$. □

Constraints

Assume a given process definition P. A constraint is a tuple $\Phi = \langle G, \mathcal{Q}, \mathcal{X} \rangle$, where $G = \langle V, E \rangle$ is a graph in $\texttt{BOUNDED}(N)$, and $\mathcal{Q}$ and $\mathcal{X}$ are defined on the set of vertices V. We call γ_Φ the associated configuration $\langle \mathcal{Q}, \mathcal{X} \rangle$ over the graph G. The denotation of a constraint Φ is defined as the infinite set of configurations $[\![\Phi]\!] = \{\gamma' \mid \gamma_\Phi \preceq \gamma'\}$ with variable size and topology. The following proposition then holds.

Proposition 5. *Given a constraint Φ, there exists an algorithm to compute a finite set of constraints whose denotation corresponds to $Pre([\![\Phi]\!])$.*

Proof. Given a constraint Φ', the symbolic computation of predecessors of instances of Φ' is obtained as the union of the constraints obtained via the backward application of individual broadcast rules with those obtained via backward time-steps.

- The computation of the set of predecessor constraints $Pre_\tau(\Phi')$ w.r.t. to time-elapse transitions is straightforward. Since the range of clocks is restricted to the interval I, we just need to collect in the finite set $Pre_\tau(\Phi')$ all constraints obtained by subtracting the same constant value $\delta \geq 0$ s.t. the resulting clock values remain all greater or equal than zero.
- The computation of the set of predecessor constraints $Pre_{\mathcal{R}}(\Phi')$ for a set of rules $\mathcal{R}$ is obtained as the union of the sets $Pre_\rho(\Phi')$ for each individual rule ρ. The computation of $Pre_\rho(\Phi')$ is defined by the Algorithm 2. The algorithm is based on the following key property. In order to generate a basis of the upward closed set of predecessors, we can restrict ourselves to constraints in which the underlying graph has either the same structure as that in Φ' or the same structure with in addition a single node that represents a sending process. For the additional node, it is necessary to consider all possible ways to connect it to already present nodes in such a way that the graph remains in $\texttt{BOUNDED}(N)$. It is immediate to verify that all other possible extensions of the graph obtained by adding a single receiver node or more than a single node produce configurations that are redundant w.r.t. the ones generated by adding at most one sender node (it follows from the fact that broadcast rules involve a single server node and that, upon reception of a message, receivers do not influence each other).

We focus now on the description of Algorithm 2. The algorithm takes in input a single constraint Φ' and it is defined for a rule ρ. The set B initially contains only Φ'. In the code from line 2 to line 7 we add to B all constraints obtained by adding a single node labeled with the target state

Algorithm 2 $Pre_\rho(\Phi')$

Require: A protocol $\langle Q, X, \Sigma, \mathcal{R}, q^{init}\rangle$, a rule $\rho = \langle q, g \xrightarrow{!!a} R, q'\rangle \in \mathcal{R}$, a constraint $\Phi' = \langle\langle V_{\Phi'}, E_{\Phi'}\rangle, \mathcal{Q}_{\Phi'}, \mathcal{X}_{\Phi'}\rangle$.

Ensure: A finite set of constraints that denotes $Pre_\rho([\![\Phi']\!])$.

$B := \{\Phi'\}$
Pick $v_{new} \notin V_{\Phi'}$
for all $E' \subseteq (V_{\Phi'} \times \{v_{new}\}) \cup (\{v_{new}\} \times V_{\Phi'})$ s.t. E' is symmetric and in $\texttt{BOUNDED}(N)$ **do**
  for all clock evaluations $\mathcal{X}_{new}$ s.t. $\mathcal{X}_{new} = \mathcal{X}_{new}[R]$ **do**
    $B := B \cup \langle\langle V_{\Phi'} \cup \{v_{new}\}, E_{\Phi'} \cup E'\rangle, \mathcal{Q}_{\Phi'}[v_{new} \hookleftarrow q'], \mathcal{X}_{\Phi'}[v_{new} \hookleftarrow \mathcal{X}_{new}]\rangle$
  end for
end for
$B' := \emptyset$
for all $\Phi = \langle\langle V, E\rangle, \mathcal{Q}, \mathcal{X}\rangle \in B$ **do**
  for all $v \in V$ s.t. $\mathcal{Q}(v) = q' \wedge \mathcal{X}(v) = \mathcal{X}(v)[R]$ **do**
    $W := \{v' \in V \mid v \sim v'\}$
    if $\nexists v' \in W$ s.t. $\begin{cases} \exists \rho = \langle \mathcal{Q}(v'), g_{v'} \xrightarrow{??a} R_{v'}, q_{v'}\rangle \in \mathcal{R}\ s.t\ \mathcal{X}(v') \models g_{v'} \\ \nexists \rho' = \langle q'_{v'}, g'_{v'} \xrightarrow{??a} R'_{v'}, \mathcal{Q}(v')\rangle \in \mathcal{R}\ s.t.\ \mathcal{X}(v') = \mathcal{X}(v')[R'_{v'}] \end{cases}$ **then**
      for all $W' \subseteq W$ s.t. $\forall n_i \in W'. \exists \langle q_{n_i}, g_{n_i} \xrightarrow{??a} R_{n_i}, \mathcal{Q}(n')\rangle \in \mathcal{R}.\ \mathcal{X}(n_i) = \mathcal{X}(n_i)[R_{n_i}]$ **do**
        for all clock evaluations $\mathcal{X}_v$ s.t. $\mathcal{X}_v \models g$ and $\mathcal{X}_v[R] = \mathcal{X}(v)$ **do**
          for all sets $\{\mathcal{X}_{n_i} \mid n_i \in W'$, $\mathcal{X}_{n_i} \models g_{n_i}$, *and* $\mathcal{X}_{n_i}[R_{n_i}] = \mathcal{X}(n_i)\}$ **do**
            $\mathcal{Q}' := \mathcal{Q}[v \hookleftarrow q][n_1 \hookleftarrow q_{n_1}][n_2 \hookleftarrow q_{n_2}]\cdots$
            $\mathcal{X}' := \mathcal{X}[v \hookleftarrow \mathcal{X}_v][n_1 \hookleftarrow \mathcal{X}_{n_1}][n_2 \hookleftarrow \mathcal{X}_{n_2}]\cdots$
            $\Psi := \langle\langle V, E\rangle, \mathcal{Q}', \mathcal{X}'\rangle$
            if $\nexists \Phi'' \in B'$ s.t. $\gamma_{\Phi''} \preceq \gamma_\Psi$ **then**
              $B' := B' \cup \{\Psi\}$
            end if
          end for
        end for
      end for
    end if
  end for
end for
return B'

of the broadcast rule (it corresponds to a sender process). Each new constraint is obtained by a distinct set of edges connecting the new node with the existing ones. We require here that the new formed constraints are based on symmetric graphs that are still bounded path.

In the code from line 8 to line 28 we generate the constraints that represent predecessor configurations of those in $[\![\Phi']\!]$. In line 8-9 we select a constraint Φ from B, and in line 10 we consider all possible sender nodes v contained in Φ by searching nodes labeled with the target state of ρ. In line 11 we identify the set W of neighbors of v. In line 12 we insert the condition needed to check whether the set of predecessors of Φ' w.r.t. v is empty or not. Specifically, Φ' and v have non empty predecessors if there are no neighbors whose current state enables a reception rule with the same label as ρ and no other rule reception can generate that state. If the condition of line 12 is violated then at least one neighbor is in a state in which it should have reacted to the broadcast but it did not, thus the resulting configuration of sender and of its neighbours does not correspond to a postcondition for ρ. If the condition of line 12 is satisfied then we compute all possible predecessors of the selected neighborhood in line 13-23. In line 13 we select the subset of neighbors n_i for which the current state is a postcondition for a specific reception ρ_{n_i}. The other nodes simply remain in their current state. In line 15 we non-deterministically select an evaluation $\mathcal{X}_{n_i}$ for the clocks in n_i that satisfies the reception guards in ρ_{n_i} and that corresponds to the current evaluation (applied to n_i) after resetting the same clocks as those specified in ρ_{n_i}. The resulting precondition states and evaluations are associated in line 16-18 to the selected nodes so as to form the new constraint Ψ. The constraint Ψ is inserted in the set of predecessor constraints only if there are not other constraints that subsume Ψ. Subsumption is tested in line 19, if the test fails then Ψ is added to the set B' in line 20.

Since B is extended with finitely many new constraints and the set of possible choices of vertexes and rules as well as of the evaluations is finite, then all the for-loops in the algorithm always terminate (i.e. the algorithm returns B' when it is not possible to add other constraints). The correctness of the construction follows by the definition of the algorithm that precisely capture all possible ways to apply backwards a broadcast rule and its set of receptions to a minimal number of nodes that characterize an upward closet set of configurations. Minimality is ensured by the fact that the addition of more than a single node to the constraint in Φ' immediately yields a redundant constraints w.r.t. computed by the algorithm. The proof is by case analysis for graphs with two additional nodes (for larger graphs it can be reduced to this simpler case) but it is omitted for brevity. □

Theorem 9. $\texttt{DTAHN}-\texttt{Reach}\,(\texttt{BOUNDED}(N), K)$ *is decidable for any* $N \geq 1$ *and* $K \geq 0$.

Proof. From propositions 4 and 5, we can apply the general results in [ACJT96] to define a backward search algorithm working on upward closed sets of extended configurations represented by their finite basis. The finite set of constraints $\langle G, Q, X\rangle$ in which G is a single node v, $Q(v) = q$, and $X(v) \in I$ can be used as finite representation of all configurations containing the control state q. Furthermore, for a fixed $N \geq 1$ the induced subgraph relation is a well-quasi-ordering on

the class of N-bounded path graphs [Din92]. This implies that for any sequence of constraints $\Phi_0\Phi_1\ldots$ there exist $i < j$ s.t. $\Phi_i \preceq \Phi_j$. This property ensures the termination of the symbolic backward reachability algorithm. □

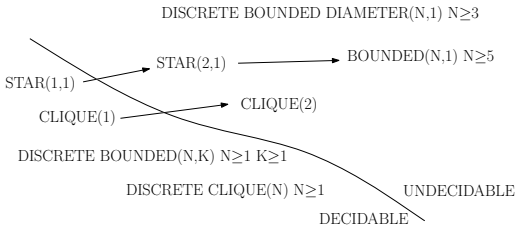


Figure 3.9: Decidability and undecidability results.

3.7 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. 3.9 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 [DSZ11].

Chapter 4

Reconfigurable Broadcast Networks

In presence of non-deterministic reconfigurations of the network topology during an execution, parameterized control state reachability becomes decidable [DSZ10]. Reconfiguration models spontaneous node movement, i.e. each node can dynamically connect (resp. disconnect) to (resp. from) any other node in the network. Furthermore, it also models the dynamic addition (resp. removal) of nodes by means of connection to the network of a previously disconnected idle node (resp. the definitive disconnection of a previously connected node). The decidability proof in [DSZ10] does not give exact complexity bounds of the problem; it simply gives a reduction to Petri net coverability, an EXPSPACE-complete problem. The precise complexity of parameterized reachability was left as an open problem in [DSZ10].

In this chapter, which is taken from [DSTZ12b, DSTZ12a], we present a comprehensive analysis of the complexity of reachability problems for reconfigurable broadcast networks. We start by generalizing the problem by considering reachability queries defined over assertions that: (i) check the presence or absence of control states in a given configuration generated by some initial configuration, and (ii) cardinality queries that define lower and upper bounds for the number of occurrences of control states in a reachable configuration. In any case the problems require, at least in principle, the exploration of an infinite-state space. Indeed they are formulated for arbitrary initial configurations, and upper bounds to the number of processes per control state are not mandatory in case (ii). We then move to the analysis of the complexity of the considered problems by showing that reachability queries for constraints that only check for the presence of a control state can be checked in polynomial time. When considering both constraints for checking presence and absence of control states the problem turns out to be NP-complete. Finally, we show that the problem becomes PSPACE-complete for cardinality queries.

Acknowledgements The author would like to thank Prof. Giorgio Delzanno, Prof. Arnaud Sangnier, and Prof. Gianluigi Zavattaro as co-authors of the articles [DSTZ12b, DSTZ12a] from

which this chapter is taken.

Related Work

As mentioned in the introduction no precise complexity bounds were given for the parameterized control state reachability problem proved decidable in [DSZ10] via a reduction to Petri net marking coverability. We attack this problem for different types of reachability queries. The interreducibility of control state reachability in models with dynamic reconfiguration, spontaneous mobility, and node-, message-, or link-failures has been formally studied in [DSZ12]. Based on the results in [DSZ12], the PTIME-algorithm presented in the current chapter can be applied not only to reconfigurable networks but also to a variety of protocols models with failures.

Symbolic backward exploration procedures for network protocols specified in graph rewriting have been presented in [JK08] (termination guaranteed for ring topologies) and [SWJ08] (approximations without termination guarantees). Decidability issues for broadcast communication in unstructured concurrent systems (or, equivalently, in fully connected networks) have been studied, e.g., in [EFM99], whereas verification of unreliable communicating FIFO systems has been studied, e.g., in [AJ93].

To our knowledge, exact algorithms (and relative complexity) for parameterized verification has not been studied in previous work on graph-based models of synchronous or asynchronous broadcast communication like [Pra95, SRS08, SRS09, NH06, EM01, God07, Mer09, SWJ08, JK08].

4.1 A Model for Reconfigurable Broadcast Networks

4.1.1 Syntax and semantics

Our model for reconfigurable broadcast networks is defined in two steps. We first define graphs used to denote network configurations and then define protocols running on each node. The label of a node denotes its current control state. Finally, we give a transition system for describing the interaction of a neighbourhood during the execution of the same protocol on each node.

Definition 5. *A Q-graph is a labeled* undirected graph $\gamma = \langle V, E, L\rangle$*, where V is a finite set of* nodes*, $E \subseteq V \times V \setminus \{\langle v, v\rangle \mid v \in V\}$ is a finite set of* edges*, and L is a labeling function from V to a set of labels Q.*

We use $L(\gamma)$ to represent all the labels present in γ (i.e. the image of the function L). The nodes belonging to an edge are called the *endpoints* of the edge. For an edge $\langle u, v\rangle$ in E, we use the

notation $u \sim_\gamma v$ and say that the vertices u and v are adjacent one to another in the graph γ. We omit γ, and simply write $u \sim v$, when it is made clear by the context.

Definition 6. *A process is a tuple $\mathcal{P} = \langle Q, \Sigma, R, Q_0 \rangle$, where Q is a finite set of control states, Σ is a finite alphabet, $R \subseteq Q \times (\{!!a, ??a \mid a \in \Sigma\}) \times Q$ is the transition relation, and $Q_0 \subseteq Q$ is a set of initial control states.*

The label $!!a$ [resp. $??a$] represents the capability of broadcasting [resp. receiving] a message $a \in \Sigma$. For $q \in Q$ and $a \in \Sigma$, we define the set $R_a(q) = \{q' \in Q \mid \langle q, ??a, q' \rangle \in R\}$ which contains the states that can be reached from the state q when receiving the message a. We assume that $R_a(q)$ is non empty for every a and q, i.e. nodes always react to broadcast messages. Local transitions (denoted by the special label τ) can be derived by using a special message m_τ such that $\langle q, ??m_\tau, q' \rangle \in R$ implies $q' = q$ for every $q, q' \in Q$ (i.e. receivers do not modify their local states).

Given a process $\mathcal{P} = \langle Q, \Sigma, R, Q_0 \rangle$, in the corresponding Reconfigurable Broadcast Network (RBN) a configuration is a Q-graph and an initial configuration is a Q_0-graph. We use Γ [resp. Γ_0] to denote the set of configurations [resp. initial configurations] associated to $\mathcal{P}$. Note that even if Q_0 is finite, there are infinitely many possible initial configurations (the number of Q_0-graphs). We assume that each node of the graph is a process that runs a common predefined protocol defined by a communicating automaton with a finite set Q of control states. Communication is achieved via selective broadcast, which means that a broadcasted message is received by the nodes which are adjacent to the sender. Non-determinism in reception is modeled by means of graph reconfigurations. We next formalize this intuition.

Given a process $\mathcal{P} = \langle Q, \Sigma, R, Q_0 \rangle$, a reconfigurable broadcast network is defined by the transition system $RBN(\mathcal{P}) = \langle \Gamma, \rightarrow, \Gamma_0 \rangle$ where the transition relation $\rightarrow \subseteq \Gamma \times \Gamma$ is such that: for $\gamma, \gamma' \in \Gamma$ with $\gamma = \langle V, E, L \rangle$, we have $\gamma \rightarrow \gamma'$ iff $\gamma' = \langle V, E', L' \rangle$ and one of the following conditions holds:

Broadcast $E' = E$ and $\exists v \in V$ s.t. $\langle L(v), !!a, L'(v) \rangle \in R$ and $L'(u) \in R_a(L(u))$ for every $u \sim v$, and $L(w) = L'(w)$ for any other node w.

Graph reconfiguration $E' \subseteq V \times V \setminus \{\langle v, v \rangle \mid v \in V\}$ and $L = L'$.

We use $\rightarrow^*$ to denote the reflexive and transitive closure of $\rightarrow$. RBN is an adequate formalism to abstractly represent broadcast communication with features like spontaneous mobility, node-, message- and link-failures.

4.1.2 Parameterized Reachability Problems

Given a process $\mathcal{P} = \langle Q, \Sigma, R, Q_0 \rangle$, a *cardinality constraint* φ over $\mathcal{P}$ is a formula which defines lower and upper bounds for the number of occurrences of each control state in a configuration.

The formulae are defined by the following grammar, where $a \in \mathbb{N}$, $q \in Q$, and $b \in (\mathbb{N} \setminus \{0\}) \cup \{+\infty\}$:

$$\varphi ::= a \leq \#q < b \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi$$

We denote by CC the class of cardinality constraints, by CC[≥ 1] the class in which negation is forbidden and atomic proposition have only the form $\#q \geq 1$ (there exists at least one occurrence of q), and finally by CC[$\geq 1, = 0$] the class of cardinality constraints as in CC[≥ 1] but where atoms can also be of the form $\#q = 0$. Given a configuration $\gamma = \langle V, E, L\rangle$ of $\mathcal{P}$ and $q \in Q$, we denote by $\#\gamma(q)$ the number of vertices in γ labeled by q, that is $\#\gamma(q) = |\{v \in V \mid L(v) = q\}|$. The satisfaction relation $\models$ for atomic formulas is defined as follows $\gamma \models a \leq \#q < b$ iff $a \leq \#\gamma(q) < b$. It is defined in the natural way for compound formulas.

We are now ready to state the cardinality reachability problem (CRP):

Input: A process $\mathcal{P}$ with $RBN(\mathcal{P}) = \langle \Gamma, \rightarrow, \Gamma_0\rangle$ and a cardinality constraint φ.

Output: Yes, if $\exists \gamma_0 \in \Gamma_0$ and $\gamma_1 \in \Gamma$ s.t. $\gamma_0 \rightarrow^* \gamma_1$ and $\gamma_1 \models \varphi$; no, otherwise.

If the answer to this problem is yes, we will write $\mathcal{P} \models \Diamond\varphi$. Note that when dealing with the complexity of this problem we will suppose that the size of the input is the size of the process defined by the product of the number of states times the number of edges added to the size of the formula in which the integer values are encoded in unary.

We use the term *parameterized* to remark that the initial configuration is not fixed a priori. In fact, the only constraint that we put on the initial configuration is that the nodes have labels taken from Q_0 without any information on their number or connection links. As a special case we can define the control state reachability problems studied in [DSZ10] as the CRP for the simple constraint $\#q \geq 1$ (i.e. is there a reachable configuration in which the state q is exposed?). Similarly, we can define the target reachability problem studied in [DSZ10] as an instance of CRP in which control states that must not occur in a target configuration are constrained by formulas like $\#q = 0$.

According to our semantics, the number of nodes stays constant in each execution starting from the same initial configuration. As a consequence, when fixing the initial configuration γ_0, we obtain finitely many possible reachable configurations. Thus, checking if there exists γ_1 reachable from a given γ_0 s.t. $\gamma_1 \models \varphi$ for a constraint φ is a decidable problem. On the other hand, checking the parameterized version of the reachability problem is in general much more difficult. E.g. consider constraints of the form $\#q \geq 1$: CRP is undecidable for a semantics without non-deterministic graph reconfigurations [DSZ10]. In [DSZ10] it is also proved that CRP for the same class of constraints is decidable. However, the proposed decidability proof is based on a reduction to the problem of coverability in Petri nets. Since no lower-bound was provided, the precise complexity of CRP with simple constraints was left as an open problem that we close in this work by showing that it is PTIME-complete.

4.2 CRP restricted to constraints in CC[≥ 1]

In this section, we study CRP restricted to CC[≥ 1]. These constraints characterize configurations in which a given set of control states is present but they can express neither the absence of states nor the number of their occurrences. We first give a lower bound for this problem.

Proposition 6. *CRP restricted to CC[≥ 1] is* PTIME*-hard.*

Proof. The proof is based on a LOGSPACE-reduction from the Circuit Value Problem (CVP) which is know to be PTIME-complete [Lad75]. CVP is defined as follows: given an acyclic Boolean circuit with k input variables, m boolean gates (of type and, or, not), a single output variable and a truth assignment for the input variables, is the value of the output equal to a given boolean value? The protocol $\mathcal{P}$ built from the CVP instance has an initial state for each of the input variables which broadcasts its truth assignment, and another one for each gate of the input circuit. In the sub-protocol associated to individual gates, a process waits for messages representing inputs and then broadcasts messages representing outputs in such a way that CVP is satisfied iff $\mathcal{P} \models \Diamond \# ok \geq 1$, where ok is a state reached only when the last gate produces the expected output.

Formally, let us assume an instance of CVP C with input/output/intermediate value names taken from a finite set VN. We denote by $v_1, \ldots, v_k \in VN$ the inputs and by $v \in VN$ the output. Furthermore, each gate g is represented by its signature $g(\odot, i_1, i_2, o)$ with $i_1, i_2, o \in VN$ and $\odot \in \{\vee, \wedge\}$ or by $g(\neg, i, o)$ with $i, o \in VN$. Finally, let $b_1, \ldots, b_k \in \{true, false\}$ be a truth assignment for the inputs and $b \in \{true, false\}$ the value for the output to be tested.

The process $\mathcal{P}_C$ associated to C has two types of initial states: q_0 (init nodes), and g (gate nodes) for each gate g of C. A node in state q_0 broadcasts (an arbitrary number of) messages that model the initial assignments to input variables. Since the assignment is fixed, broadcasting these messages several times (or receiving them from different initial nodes) does not harm the correctness of the encoding. When receiving an evaluation for their inputs (from an initial node or another gate node), a gate node evaluates the corresponding boolean function and then repeatedly broadcasts the value of the corresponding output. Since C is acyclic, once computed, the output value remains always the same (i.e. recomputing it does not harm). Finally, reception of a value v for output z sends a q_0 node into state ok. Reachability of an output value v reduces then to CRP for the process $\mathcal{P}_C$ with ok the control state to be reached.

Process rules are defined as follows. For $i \in \{1, \ldots, k\}$, we have rules $\langle q_0, !!(v_i = b_i), q_0 \rangle$ and $\langle q_0, ??(v = b), ok \rangle$. They model the assignment of value v_i to input x_i and reception of output value v. For gate $g(\odot, i_1, i_2, o)$ and for each assignment $\alpha = \langle b'_1, b'_2 \rangle$ (with $b'_1, b'_2 \in \{true, false\}$) of values to $\langle i_1, i_2 \rangle$ (a constant number for each gate), we associate the sub-protocol in Figure 4.1. We omit self-loops associated to receptions for which there are no explicit rules. We use a similar encoding for a not gate.

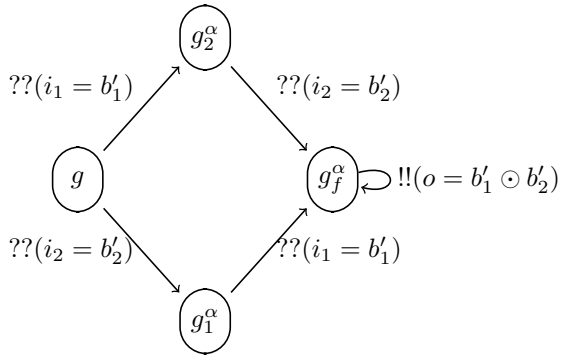


Figure 4.1: Sub-protocol for gate $g(\odot, i_1, i_2, o)$

Consider now the resulting process $\mathcal{P}_C = \langle Q, \Sigma, R, \{q_0\} \cup \{g \mid g \text{ is a gate in } C\}\rangle$ with corresponding transition system $RBN = \langle \Gamma, \rightarrow, \Gamma_0\rangle$. There exists $\gamma \in \Gamma_0$ and γ' in Γ s.t. $\gamma \rightarrow^* \gamma'$ and $\gamma' \models \#ok \geq 1$ iff b is the value for v in C with input values $b_1, \ldots, b_k$. □

We now show that CRP restricted to CC[≥ 1] is in PTIME. We first observe that, in order to decide if control state q can be reached, we can focus our attention on initial configurations in which the topology is fully connected (i.e. graphs in which all pairs of nodes are connected). Indeed, graph reconfigurations can be applied to non-deterministically transform a topology into any other one.

Another key observation is that if the control state q is reached once from the initial configuration γ_0, then it can be reached an arbitrary number of times by considering larger and larger initial configurations γ'_0. More specifically, the initial configuration γ'_0 is obtained by replicating several times the initial graph γ_0. The replicated parts are then connected in all possible ways (to obtain a fully connected topology). We can then use dynamic reconfiguration in order to mimic parallel executions of the original system and reach a configuration with several repeated occurrences of state q.

For what concerns constraints in CC[≥ 1] this property of CRP avoids the need of counting the occurrences of states. We just have to remember which states can be generated by repeatedly applying process rules. As a consequence, in order to define a decision procedure for checking control state reachability we can take the following assumptions: (i) forget about the topology underlying the initial configuration; (ii) forget about the number of occurrences of control states in a configuration; (iii) consider a single symbolic path in which at each step we apply all possible rules whose preconditions can be satisfied in the current set and then collect the resulting set of computed states.

We now formalize the previous observations. Let $\mathcal{P} = \langle Q, \Sigma, R, Q_0\rangle$ be a process with $RBN(\mathcal{P}) = \langle \Gamma, \rightarrow, \Gamma_0\rangle$ and let $\texttt{Reach}(\mathcal{P})$ be the set of reachable control states equals to $\{q \in Q \mid \exists \gamma \in \Gamma_0. \exists \gamma' \in \Gamma.$ s.t. $\gamma \rightarrow^* \gamma'$ and $q \in L(\gamma')\}$. We will now prove that Algorithm 4.1 computes

Input: $\mathcal{P} = \langle Q, \Sigma, R, Q_0 \rangle$ a process
Output: $S \subseteq Q$ the set of reachable control states in $RBN(\mathcal{P})$
  $S := Q_0$
  $oldS := \emptyset$
  while $S \neq oldS$ **do**
    $oldS := S$
    for all $\langle q_1, !!a, q_2 \rangle \in R$ such that $q_1 \in oldS$ **do**
      $S := S \cup \{q_2\} \cup \{q' \in Q \mid \langle q, ??a, q' \rangle \in R \wedge q \in oldS\}$
    end for
  end while

Listing 4.1: Computing the set of control states reachable in a RBN

$\texttt{Reach}(\mathcal{P})$. Let S be the result of the Algorithm 4.1 (note that this algorithm necessarily terminates because the **while**-loop is performed at most $|Q|$ times). We have then the following lemma.

Lemma 4. *The two following properties hold:*

(i) *There exist two configurations* $\gamma_0 \in \Gamma_0$ *and* $\gamma \in \Gamma$ *such that* $\gamma_0 \rightarrow^* \gamma$ *and* $L(\gamma) = S$.

(ii) $S = \texttt{Reach}(\mathcal{P})$.

Proof. We first prove (i). We denote by $S_0, S_1, \ldots, S_n$ the content of S after each iteration of the loop of the Algorithm 4.1. We recall that an undirected graph $\gamma = \langle V, E, L \rangle$ is complete if $\langle v, v' \rangle \in E$ for all $v, v' \in V$. We will now consider the following statement: for all $j \in \{0, \ldots, n\}$, for all $k \in \mathbb{N}$, there exists a complete graph $\gamma_{j,k} = \langle V, E, L \rangle$ in Γ verifying the two following points:

1. $L(\gamma_{j,k}) = S_j$ and for each $q \in S_j$, the set $\{v \in V \mid L(v) = q\}$ has more than k elements (i.e. for each element q of S_j there are more than k nodes in $\gamma_{j,k}$ labeled with q),
2. there exits $\gamma_0 \in \Gamma_0$ such that $\gamma_0 \rightarrow^* \gamma_{j,k}$.

To prove this statement we reason by induction on j. First, for $j = 0$, the property is true, because for each $k \in \mathbb{N}$, the graph $\gamma_{0,k}$ corresponds to the complete graphs where each of the initial control states appears at least k times. We now assume that the property is true for all naturals smaller than j (with $j < n$) and we will show it is true for $j + 1$. We define C_a as the set $\{\langle \langle q_1, !!a, q_2 \rangle, \langle q, ??a, q' \rangle \rangle \in R \times R \mid q_1, q \in S_j\}$ and M its cardinality. Let $k \in \mathbb{N}$ and let $N = k + 2 * k * M$. We consider the graph $\gamma_{j,N}$ where each control state present in S_j appears at least N times (such a graph exists by the induction hypothesis). From $\gamma_{j,N}$, we build the graph $\gamma_{j+1,k}$ obtained by repeating k times the following operations:

- for each pair $\langle\langle q_1, !!a, q_2\rangle, \langle q, ??a, q'\rangle\rangle \in C_a$, select a node labeled by q_1 and one labeled by q and update their label respectively to q_2 and q' (this simulates a broadcast from the node labeled by q_1 received by the node labeled q in the configuration in which all the other nodes have been disconnected thanks to the reconfiguration rule and reconnected after). Note that the two selected nodes can communicate because the graph is complete.

By applying these rules it is then clear that $\gamma_{j,N} \rightarrow^* \gamma_{j+1,k}$ and also that $\gamma_{j+1,k}$ verifies the property 1 of the statement. Since by induction hypothesis, we have that there exists $\gamma_0 \in \Gamma_0$ such that $\gamma_0 \rightarrow^* \gamma_{j,N}$, we also deduce that $\gamma_0 \rightarrow^* \gamma_{j+1,k}$, hence the property 2 of the statement also holds. From this we deduce that (i) is true.

To prove (ii), from (i) we have that $S \subseteq \texttt{Reach}(\mathcal{P})$ and we now prove that $\texttt{Reach}(\mathcal{P}) \subseteq S$. Let $q \in \texttt{Reach}(\mathcal{P})$. We show that $q \in S$ by induction on the minimal length of an execution path $\gamma_0 \rightarrow^* \gamma$ such that $\gamma_0 \in \Gamma_0$ and $q \in L(\gamma)$. If the length is 0 then $q \in Q_0$ hence also $q \in S$. Otherwise, let $\gamma' \rightarrow \gamma$ be the last transition of the execution. We have that there exists $q_1 \in L(\gamma')$ such that $\langle q_1, !!a, q\rangle \in R$ [or $q_1, q_2 \in L(\gamma')$ such that $\langle q_1, !!a, q_3\rangle, \langle q_2, ??a, q\rangle \in R$]. By induction hypothesis we have that $q_1 \in S$ [or $q_1, q_2 \in S$]. By construction, we can conclude that also $q \in S$. □

Since constraints in CC[≥ 1] check only the presence of states and do not contain negation, given a configuration γ and a constraint φ in CC[≥ 1] such that $\gamma \models \varphi$, we also have that $\gamma' \models \varphi$ for every γ' such that $L(\gamma) \subseteq L(\gamma')$. Moreover, given a process $\mathcal{P}$, by definition of $\texttt{Reach}(\mathcal{P})$ we have that $L(\gamma) \subseteq \texttt{Reach}(\mathcal{P})$ for every reachable configuration γ, and by Lemma 4 there exists a reachable configuration γ_f such that $L(\gamma_f) = \texttt{Reach}(\mathcal{P})$. Hence, to check $\mathcal{P} \models \Diamond\varphi$ it is sufficient to verify whether $\gamma_f \models \varphi$ for such a configuration γ_f. This can be done algorithmically as follows: once the set $\texttt{Reach}(\mathcal{P})$ is computed, check if the boolean formula obtained from φ by replacing each atomic constraint of the form $\#q \geq 1$ by *true* if $q \in \texttt{Reach}(\mathcal{P})$ and by *false* otherwise is valid. This allows us to state the following theorem.

Theorem 10. *CRP restricted to CC[≥ 1] is* PTIME*-complete.*

Proof. The lower bound is given by Proposition 6. To obtain the upper bound, it suffices to remark that the Algorithm 4.1 is in PTIME since it requires at most $|Q|$ iterations each one requiring at most $|R|^2$ look-ups (of active broadcast/receive transitions) for computing new states to be included, and also that evaluating the validity of a boolean formula can be done in polynomial time. □

4.3 CRP restricted to constraints in CC[$\geq 1, = 0$]

We consider now decidability and complexity of CRP for constraints in CC[$\geq 1, = 0$]. This kind of queries can be used to specify that a given control state is not present in a configuration (using

atomic constraints of the form $\#q = 0$).

Proposition 7. *CRP for constraints in CC[$\geq 1, = 0$] is* NP*-hard.*

Proof. The proof is based on a reduction of the boolean satisfiability problem (SAT), which is known to be NP-complete. The rationale is as follows. The encoding of the SAT instance for a boolean formula Φ with variables in V is based on a protocol with only local transitions from a single initial state into states that encode truth assignments in $\{v, \overline{v} \mid v \in V\}$. A CC[$\geq 1, = 0$] query is then built in order to guarantee that there are no contradicting assignments to variables. The query also ensures that the selected assignments satisfy the formula Φ, where positive literals v are replaced by $\#v \geq 1$ and negative literals $\neg v$ are replaced by $\#v = 0$.

Let Φ be a boolean formula in conjunctive normal form over the set of variables $V = \{v_1, \ldots, v_k\}$. We define a process $\mathcal{P}$ with initial state q_0 and the following set of rules $R = \{\langle q_0, \tau, v\rangle \mid v \in V\} \cup \{\langle q_0, \tau, \overline{v}\rangle \mid v \in V\}$. From Φ, we build a constraint $\varphi \wedge \psi$ where φ is the formula obtained from Φ by replacing each positive literal v by $\#v \geq 1$ and each negative literal $\neg v$ by $\#\overline{v} \geq 1$ and $\psi = \bigwedge_{i=1}^{k} (\#v_i \geq 1 \wedge \#\overline{v_i} = 0) \vee (\#v_i = 0 \wedge \#\overline{v_i} \geq 1)$. The former constraint is the natural encoding of the input propositional formula whereas the latter assigns a consistent interpretation to the control state labels v_i and $\overline{v_i}$ as assignments to the propositional variable v_i. The constraint $\varphi \wedge \psi$ is a formula in CC.

A node in the initial state q_0 makes a guess for the boolean valuation of a variable v by moving to state v [resp. to $\overline{v}$] if the associated chosen value is $true$ [resp. $false$]. The formula ψ ensures that no contradictory valuation is generated by stating that for each variable v in V only one type of control state v or $\overline{v}$ is chosen. Assume that the formula Φ is satisfiable and let $\{b_1, \ldots, b_k\} \in \{true, false\}^k$ be an interpretation over the variables $\{v_1, \ldots, v_k\}$ that satisfies it. From an initial configuration γ_0 with k nodes, it is possible to reach a configuration γ such that $\gamma \models \psi$ and for all $1 \leq i \leq k$ if $b_i = true$ then $\gamma' \models \#v_i \geq 1$ else $\gamma' \models \#v_i = 0$. $\gamma \models \varphi \wedge \psi$ clearly holds here. Vice versa, if there exists a computation that reaches a configuration that satisfies $\varphi \wedge \psi$, then we have $m \geq k$ nodes whose labels correspond to a consistent interpretation of the variables in V and which satisfies Φ. □

We will now give an algorithm in NP to solve CRP for constraints in CC[$\geq 1, = 0$]. As for Algorithm 4.1, this new algorithm works on sets of control states. The algorithm works in two main phases. In a first phase it generates an increasing sequence of sets of control states that can be reached in the considered process definition. At each step the algorithm adds the control states obtained from the application of the process rules to the current set of labels. Unlike the Algorithm 4.1, this new algorithm does not merge different branches, i.e. application of distinct rules may lead to different sequences of sets of control states. In a second phase the algorithm only removes control states applying again process rules in order to reach a set of control states that satisfies the given constraint.

Input: $\mathcal{P} = \langle Q, \Sigma, R, Q_0 \rangle$ a process and φ a constraint over $\mathcal{P}$ in CC[$\geq 1, = 0$]
Output: Does $\mathcal{P} \models \Diamond\varphi$?
guess $S_0, \ldots, S_m, T_1, \ldots, T_n \subseteq Q$ with $m, n \leq |Q|$
if $S_0 \not\subseteq Q_0$ **then return false**
for all $i \in \{0, \ldots, m-1\}$ **do**
if $S_{i+1} \notin \texttt{postAdd}(\mathcal{P}, S_i)$ **then return false**
end for
$T_0 = S_m$
for all $i \in \{0, \ldots, n-1\}$ **do**
if $T_{i+1} \notin \texttt{postDel}(\mathcal{P}, T_i)$ **then return false**
end for
If T_n satisfies φ **then return true else return false**

Listing 4.2: Solving CRP for constraints in CC[$\geq 1, = 0$]

Definition 7. *For a process $\mathcal{P} = \langle Q, \Sigma, R, Q_0 \rangle$ and a set $S \subseteq Q$, we define the operator $\texttt{postAdd}(\mathcal{P}, S) \subseteq 2^Q$ as follows: $S' \in \texttt{postAdd}(\mathcal{P}, S)$ if and only if the two following conditions are satisfied:*

(i) $S \subseteq S'$ *and*

(ii) for all $q' \in S' \setminus S$, there exists a rule $\langle q, !!a, q' \rangle \in R$ such that $q \in S$ (q' is produced by a broadcast) or there exist rules $\langle p, !!a, p' \rangle$ and $\langle q, ??a, q' \rangle \in R$ such that $q, p \in S$ and $p' \in S'$ (q' is produced by a reception).

In other words, all the states in $S' \in \texttt{postAdd}(\mathcal{P}, S)$ are either in S or states obtained from the application of broadcast/reception rules to labels in S. Similarly, we define the operator $\texttt{postDel}(\mathcal{P}, S)$ as follows:

Definition 8. *Given a process $\mathcal{P} = \langle Q, \Sigma, R, Q_0 \rangle$ and a set $S \subseteq Q$, $\texttt{postDel}(\mathcal{P}, S) \subseteq 2^Q$ is the set such that $S' \in \texttt{postDel}(\mathcal{P}, S)$ if and only if $S' \subseteq S$ and one of the following conditions hold:*

(i) either $S \setminus S' = \emptyset$, or

(ii) $S \setminus S' = \{q\}$ and there exists a rule $\langle q, !!a, q' \rangle \in R$ such that $q' \in S'$ (q is consumed by a broadcast), or

(iii) $S \setminus S' = \{q\}$ and there exist two rules $\langle p, !!a, p' \rangle, \langle q, ??a, q' \rangle \in R$ such that $p, p', q' \in S'$ (q is consumed by a reception).

Finally, we say that a set $S \subseteq Q$ satisfies an atom $\#q = 0$ if $q \notin S$ and it satisfies an atom $\#q \geq 1$ if $q \in S$; satisfiability for composite boolean formulae of CC[$\geq 1, = 0$] is then defined in the natural way. We have then the following Lemma.

Lemma 5. *There is an execution of Algorithm 4.2 which answers YES on input $\mathcal{P}$ and φ iff $\mathcal{P} \models \Diamond\varphi$.*

Proof. Let $\mathcal{P} = \langle Q, \Sigma, R, Q_0 \rangle$ a process with $RBN(\mathcal{P}) = \langle \Gamma, \rightarrow, \Gamma_0 \rangle$ and φ a constraint over $\mathcal{P}$ in CC. First we assume that the Algorithm 4.2 answers YES on input $\mathcal{P}$ and φ. This means that there exists $S_0, \ldots, S_m, T_0, T_1, \ldots, T_n$ such that $1 \leq m, n \leq |Q|$ and $S_0 \subseteq Q_0$, and for all $i \in \{0, \ldots, m-1\}$, $S_{i+1} \in \texttt{postAdd}(\mathcal{P}, S_i)$ and $T_0 = S_m$ and for all $i \in \{0, \ldots, n-1\}$, $T_{i+1} \in \texttt{postDel}(\mathcal{P}, T_i)$. We will now prove that there exists two configurations $\gamma_0 \in \Gamma_0$ and $\gamma \in \Gamma$ such that $\gamma_0 \rightarrow^* \gamma$ and $L(\gamma) = T_n$. First, by reasoning the same way we did in the proof of Lemma 4, we can deduce that for any $k \in \mathbb{N} \setminus \{0\}$, there exists $\gamma_0 \in \Gamma_0$ and a complete graph $\gamma_k = \langle V, E, L \rangle$ in Γ such that $L(\gamma_k) = S_m$ and for every $q \in S_m$ the set $\{v \in V \mid L(v) = q\}$ has more than k elements. Now we are going to prove that for any $j \in \{0, \ldots, n\}$, for all $k \in \mathbb{N} \setminus \{0\}$, there is a complete graph $\gamma_{j,k}$ such that:

1. $L(\gamma_{j,k}) = T_j$ and for each $q \in S_j$, the set $\{v \in V \mid L(v) = q\}$ has more than k elements (i.e. for each element q of S_j there are more than k nodes in $\gamma_{j,k}$ labelled with q);
2. there exists $\gamma_0 \in \Gamma_0$ such that $\gamma_0 \rightarrow^* \gamma_{j,k}$.

To prove this statement we reason by induction on j. For $j = 0$, since the statement holds for S_m, it holds also for $T_0 = S_m$. We now assume that the property is true for all naturals smaller than j (with $j < n$) and we will show it is true for $j + 1$. We consider now the set $T_j \setminus T_{j+1}$ (assuming it is not empty, otherwise the property trivially holds). By property of the operator `postDel`, we have $T_{j+1} \subseteq T_j$. Let $k \in \mathbb{N}$. Given the graph $\gamma_{j,k+1}$, we now build another graph $\gamma_{j+1,k}$. The reason why we get a smaller k is that, when consuming nodes via a reception rule, the nodes enabled to produce the required message will decrease by one because of the broadcast rule, possibly decreasing the minimum number of occurrences of processes per local state. This is not an issue at all however, because thanks to the inductive hypothesis we can pick any k big enough to prevent subsequent steps from failing due to lack of broadcasting processes. The construction proceeds as follows.

- If $T_j \setminus T_{j+1} = \{q\}$ and there exists a rule $\langle q, !!a, q' \rangle \in R$ such that $q' \in T_{j+1}$, then this rule is applied to all the nodes labelled by q; first each node is isolated with the reconfiguration rule, then the broadcast rule is performed and then the complete graph is rebuilt. Note that the application of this rule consecutively will only increase the number of nodes labelled by q' which were already present in $\gamma_{j,k+1}$.

- If $T_j \setminus T_{j+1} = \{q\}$ and there exist two rules $\langle p, !!a, p' \rangle, \langle q, ??a, q' \rangle \in R$ such that $p, p', q' \in T_{j+1}$ (q is consumed by a broadcast), then all the nodes labelled by q are isolated together with a node labelled by p so that all these nodes are connected, then p broadcast a sending all the other nodes in q' and finally the complete graph is rebuilt; as a consequence there is no more nodes labelled by q, the number of nodes labelled by q' and p' have increased and the number of nodes labelled by p has decreased of one unit.

By applying these rules it is then clear that $\gamma_{j,k+1} \rightarrow^* \gamma_{j+1,k}$ and also that $\gamma_{j+1,k}$ verifies the property 1 of the statement. Since by induction hypothesis, we have that there exists $\gamma_0 \in \Gamma_0$ such that $\gamma_0 \rightarrow^* \gamma_{j,k+1}$, we also deduce that $\gamma_0 \rightarrow^* \gamma_{j+1,k}$, hence the property 2 of the statement also holds. Hence if the Algorithm 4.2 returns YES on input $\mathcal{P}$ and φ, we deduce that there exist a reachable configuration $\gamma \in \Gamma$ such that $L(\gamma) = T_n$ and since T_n satisfies φ, we also have that $\gamma \models \varphi$, hence $\mathcal{P} \models \Diamond\varphi$.

We now assume that there exists two configurations $\gamma_0 \in \Gamma_0$ and $\gamma \in \Gamma$ such that $\gamma_0 \rightarrow^+ \gamma$ (the case $\gamma_0 = \gamma$ can be easily verified) and $\gamma \models \varphi$. Hence there exists $\gamma_1, \ldots, \gamma_k \in \Gamma$ such that $\gamma_0 \rightarrow^+ \gamma_1 \ldots \rightarrow^+ \gamma_k$ with $\gamma_k = \gamma$ and for all $i \in \{1, \ldots, k\}$, exactly one broadcast rule has been applied between γ_i and γ_{i+1}. From this execution we build a sequence of set of control states $(S'_i)_{0 \leq i \leq k}$ such that $S'_0 = L(\gamma_0)$ and for all $0 \leq i \leq k-1$, $S'_{i+1} = S'_i \cup L(\gamma_i)$. By definition of the broadcast rule and of the operator `postAdd`, we deduce that $S'_{i+1} \in \texttt{postAdd}(\mathcal{P}, S'_i)$. From this sequence, we can furthermore extract a subsequence $(S_i)_{0 \leq i \leq m}$ such that for all $0 \leq i \leq m-1$, $S_{i+1} \in \texttt{postAdd}(\mathcal{P}, S_i)$ and $S_{i+1} \neq S_i$ and for all $0 \leq j \leq k$, there exists $0 \leq i \leq m$ such that $S'_j = S_i$. Since we have $S_i \subset S_{i+1}$ for all $0 \leq i \leq m-1$, we deduce that necessarily $m \leq |Q|$. Now we build another sequence of control states $(T'_i)_{0 \leq i \leq k}$ such that $T'_0 = S_m$ and for all $0 \leq i \leq k-1$, $T'_{i+1} = T'_i \setminus E_i$ where for all $0 \leq i \leq k-1$, $E_i = \{q \in L(\gamma_i) \mid \nexists j > i \text{ s.t. } q \in L(\gamma_j)\}$. In other words, to build T'_{i+1} from T'_i we delete the control states q that are present in γ_i and will never be present in any γ_j for $j > i$. We recall that by construction for all $1 \leq i \leq k$, we have $L(\gamma_i) \subseteq T'_0$ and hence by construction of the sequence $(T'_i)_{0 \leq i \leq k}$ we have necessarily $L(\gamma) = T'_k$. By definition of the broadcast rule and of the operator `postDel` we also deduce that $T'_{i+1} \in \texttt{postDel}^{|E_i|}(\mathcal{P}, T'_i)$, where we apply `postDel` $|E_i| - 1$ times for each state deleted by a reception rule (iff $|E_i| - 1 \geq 1$) and one last time to delete the state of the broadcast rule. From this sequence, we can furthermore extract a subsequence $(T_i)_{0 \leq i \leq n}$ such that for all $0 \leq i \leq n-1$, $T_{i+1} \in \texttt{postDel}(\mathcal{P}, T_i)$ and $T_{i+1} \neq T_i$ and for all $0 \leq j \leq k$, there exists $0 \leq i \leq n$ such that $T'_j = T_i$. Since we have $T_{i+1} \subset T_i$ for all $0 \leq i \leq n-1$, we deduce that necessarily $n \leq |Q|$ and also we have $T(n) = L(\gamma)$. Since $\gamma \models \varphi$, we deduce that T_n satisfies φ and consequently we have proved that there is an execution of Algorithm 4.2 which answers YES on input $\mathcal{P}$ and φ. □

It is then clear that each check performed by the Algorithm 4.2 (i.e. $S_0 \subseteq Q_0$ and $S_{i+1} \in \texttt{postAdd}(\mathcal{P}, S_i)$ and $T_{i+1} \in \texttt{postAdd}(\mathcal{P}, T_i)$ and T_n satisfies φ) can be performed in polynomial time in the size of the process $\mathcal{P}$ and of the formula φ and since m and n are smaller than the

number of control states in $\mathcal{P}$, we deduce the following theorem (the lower bound being given by Proposition 7).

Theorem 11. *CRP for constraints in CC[$\geq 1, = 0$] is* NP*-complete.*

4.4 Complexity of CRP in Full CC

In this section we will show that CRP for the entire class of cardinality constraints CC is PSPACE-complete. First we prove the lower bound.

Proposition 8. *CRP is* PSPACE*-hard.*

Proof. Given a 1-safe net $N = \langle P, T, \vec{m_0} \rangle$ and a marking $\vec{m_1}$, we encode the reachability problem as a CRP problem for the process $\mathcal{P}$ and cardinality constraint φ defined next. For each place $p \in P$, we introduce control states p_1 and p_0 to denote the presence or absence of the token in p, respectively. Furthermore, we introduce a special control state ok. The control state is used to control the net simulation. Transitions of the controller are depicted in the upper part of Figure 4.2.

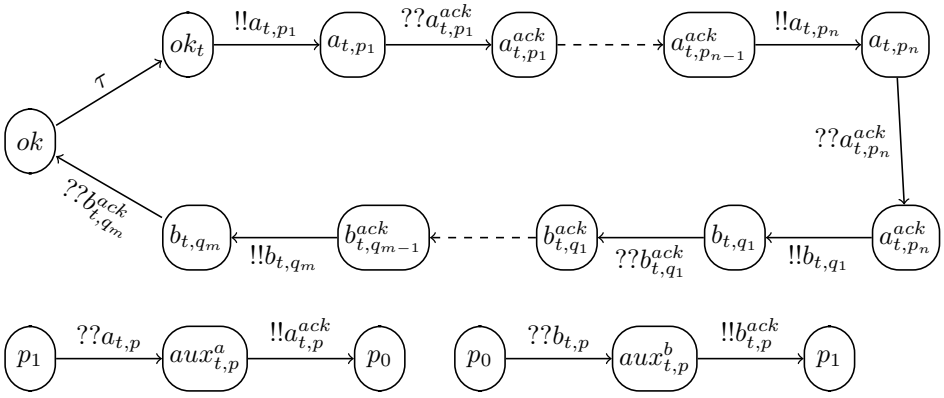


Figure 4.2: Simulation of a transition t with ${}^\bullet t = \{p_1, \ldots, p_n\}$ and $t^\bullet = \{q_1, \ldots, q_m\}$

The first rule of the controller selects the current transition to simulate. The simulation of the transition t with ${}^\bullet t = \{p_1, \ldots, p_n\}$ and $t^\bullet = \{q_1, \ldots, q_m\}$ is defined via two sequences of messages (we denote ${}^\bullet t$ and $t^\bullet$ as sets instead of multisets because we are considering a 1-safe net and it is hence not possible that a transition consumes or produces more than one token for each place). The first one is used to remove the token from $p_1, \ldots, p_n$, whereas the second one is used to put the token in $q_1, \ldots, q_m$. To guarantee that every involved place reacts to the protocol —i.e. messages are not lost— the controller waits for an acknowledgement from each of them. Transitions of places are depicted in the lower part of Figure 4.2. It is not restrictive to assume that there is only one token in the initial marking $\vec{m_0}$ (otherwise we add an auxiliary initial place

and a transition that generates $\vec{m_0}$ by consuming the initial token). Let p^0 be such a place. We define the initial states Q_0 of the process $\mathcal{P}$ as $\{p_1^0, ok\} \cup \{p_0 \mid p \in P \setminus \{p^0\}\}$, in order to initially admit control states representing the controller, the presence of the initial token, and the absence of tokens in other places. The reduction does not work if there are several copies of controller nodes and/or place representations (i.e. $p_1, p_0, \ldots$) interacting during a simulation (interferences between distinct nodes representing controllers/places may lead to incorrect results). However we can ensure that the reduction is accurate by checking the number of occurrences of states exposed in the final configuration: it is sufficient to check that only one controller and only one node per place in the net are present. Besides making this check, the cardinality constraint φ should also verify that the represented net marking coincides with $\vec{m_1}$. Namely, we define φ as follows:

$$\begin{aligned}\varphi = & \bigwedge_{p \in \vec{m_1}, t \in T} \big(\#p_1 = 1 \wedge \#p_0 = 0 \wedge \#aux_{t,p}^a = 0 \wedge \#aux_{t,p}^b = 0\big) \wedge \\ & \bigwedge_{q \notin \vec{m_1}, t \in T} \big(\#q_1 = 0 \wedge \#q_0 = 1 \wedge \#aux_{t,q}^a = 0 \wedge \#aux_{t,q}^b = 0\big) \wedge \#ok = 1 \wedge \\ & \bigwedge_{t \in T} (\#ok_t = 0) \wedge \bigwedge_{t \in T, q \in P} \big(\#a_{t,q} = 0 \wedge \#b_{t,q} = 0 \wedge \#a_{t,q}^{ack} = 0 \wedge \#b_{t,q}^{ack} = 0\big)\end{aligned}$$

Since the number of nodes stays constant during an execution, the post-condition specified by φ is propagated back to the initial configuration. Therefore, if the protocol satisfies CRP for φ, then in the initial configuration there must be one single controller node with state ok, and for each place p one single node with either state p_1 or state p_0. Under this assumption, it is easy to check that a run of the protocol corresponds precisely to a firing sequence in the 1-safe net. Thus an execution run satisfies φ if and only if the corresponding firing sequence reaches the marking $\vec{m_1}$. □

We now show that there exists an algorithm to solve CRP in PSPACE. The main idea is to use a symbolic representation of configurations in which the behavior of a network is observed exactly for a fixed number of nodes only. For all the other nodes, we only maintain the control state they are labeled with and not their precise number.

Without loss of generality, we consider for simplicity only processes with $Q_0 = \{q_0\}$, as multiple initial states can be encoded through local transitions from q_0. Given a process $\mathcal{P} = \langle Q, \Sigma, R, \{q_0\}\rangle$ and a cardinality constraint φ over $\mathcal{P}$ we denote by $\mathrm{val}(\varphi) \in \mathbb{N}$ the largest natural constant that appears in φ. We then denote by $\mathrm{psize}(\varphi)$ the natural $|Q| * \mathrm{val}(\varphi)$. Intuitively $\mathrm{psize}(\varphi)$ is the number of witness nodes we keep track of: we reserve $\mathrm{val}(\varphi)$ processes to each control state that may appear in φ.

A symbolic configuration for $\mathcal{P}$ and φ is then a pair $\theta = \langle v, S\rangle$ where $v \in Q^{\mathrm{psize}(\varphi)}$ is a vector of $\mathrm{psize}(\varphi)$ elements of Q and $S \subseteq Q$. For $q \in Q$, we then write $\#v(q)$ to indicate the number of occurrences of q in the vector v. Note that by definition $0 \leq \#v(q) \leq \mathrm{psize}(\varphi)$ for every $q \in Q$

and that $\sum_{q \in Q} \#v(q) = \text{psize}(\varphi)$. This allows us to describe the set of configurations $[\![\theta]\!] \subseteq \Gamma$ characterized by a symbolic configuration $\theta = \langle v, S \rangle$ as follows: we have $\gamma \in [\![\theta]\!]$ if and only $\#\gamma(q) > \#v(q)$ for every $q \in S$ and $\#\gamma(q) = \#v(q)$ for every $q \in Q \setminus S$. Hence a symbolic configuration $\theta = \langle v, S \rangle$ represents all the configurations such that the number of occurrences of a control state q is greater than the number of occurrences of q in v if $q \in S$, or equal when $q \notin S$. We will say that a symbolic configuration θ satisfies the cardinality constraint φ, written $\theta \models \varphi$, iff $\gamma \models \varphi$ for all $\gamma \in [\![\theta]\!]$. We use Θ to represent the set of symbolic configurations.

We make the following non restrictive assumptions: there is no constraint on the unique initial state q_0 in the cardinality constraints, the only outgoing transitions from the state q_0 are local transitions (labelled with τ), in the symbolic configurations $\langle v, S \rangle$ we always have $q_0 \in S$, and the initial configuration θ_0 is $\langle (q_0, \ldots, q_0), \{q_0\} \rangle$. The most important assumption is the first one about the absence of constraints on q_0: it is needed to guarantee the correctness of our symbolic procedure. For instance, consider a process $\mathcal{P} = \langle \{q_0\}, \Sigma, R, \{q_0\} \rangle$ and a cardinality constraint φ of the form $1 \leq \#q_0 < 2$. We have then $\text{psize}(\varphi) = 2$ and the symbolic configurations are of the form $\langle (q_0, q_0), S \rangle$. It is then obvious that all the symbolic configurations do not satisfy φ while the initial concrete configuration with only one node does. The above assumptions are not restrictive because given a process $\mathcal{P} = \langle Q, \Sigma, R, \{q_0\} \rangle$ and a cardinality constraint φ, we can define a new process $\mathcal{P}' = \langle Q', \Sigma, R', \{q_{init}\} \rangle$ where $Q' = Q \cup \{q_{init}\}$ and $R' = R \cup \{\langle q_{init}, \tau, q_0 \rangle\}$, i.e. q_{init} is a new initial state from which the process is enabled to go to q_0 thanks to a local transition. As there is no constraint in φ about q_{init} it is immediate to prove the following Lemma:

Lemma 6. *$\mathcal{P} \models \Diamond\varphi$ if and only if $\mathcal{P}' \models \Diamond\varphi$.*

Proof. Assume $\mathcal{P} \models \Diamond\varphi$, then there are a configuration γ_0 and a configuration γ of $\mathcal{P}$ such that $\gamma_0 \rightarrow^* \gamma$ and $\gamma \models \varphi$. In $\mathcal{P}'$ we can take the initial configuration γ_0' with the same number of nodes as in γ_0, assume that all the node begin to fire the transition $\langle q_{init}, \tau, q_0 \rangle$ (hence all the nodes will be labeled by q_0) and then perform the same execution as in $\mathcal{P}$, hence $\mathcal{P}' \models \Diamond\varphi$. Now assume that $\mathcal{P}' \models \Diamond\varphi$, hence there are an initial configuration γ_0' and a configuration γ' of $\mathcal{P}'$ such that $\gamma_0' \rightarrow^* \gamma'$ and $\gamma' \models \varphi$. It is possible to simulate in $\mathcal{P}$ almost the same execution (without the use of the rule $\langle q_{init}, \tau, q_0 \rangle$) by taking as initial configuration in $\mathcal{P}$ the configurations which have as cardinality the number of γ' which are not labelled by q_{init} and thus obtaining an execution which leads to a configuration which satisfies ϕ. □

We now define a relation on the symbolic configurations to represent the effect that process rules have on symbolic configurations. Let $\mathcal{P} = \langle Q, \Sigma, R, \{q_0\} \rangle$ be a process, φ a cardinality constraint and θ the associated set of symbolic configurations. For each rule $r \in R$ of form $\langle q, !!a, q' \rangle$, we define the symbolic transition relation $\rightsquigarrow_r \subseteq \Theta \times \Theta$ as follows, we have $\langle v, S \rangle \rightsquigarrow_r \langle v', S' \rangle$ if and only if at least one of the two following conditions holds:

1. *(broadcast from a state in v)* there exists $i \in \{1, \ldots, \text{psize}(\varphi)\}$ such that $v[i] = q$ and $v'[i] = q'$ (i.e. the sending process switches state according to r) and:

- for all $j \in \{1, \ldots, \mathrm{psize}(\varphi)\} \setminus \{i\}$ we have either $v[j] = v'[j]$ or there exists $\langle q_r, ??a, q'_r \rangle \in R$ such that $v[j] = q_r$ and $v'[j] = q'_r$ (i.e. other processes in the pool may or may not react to the broadcast);
- for each $q_s \in Q \setminus \{q_0\}$:
  - if $q_s \in S' \setminus S$ then there exists $q'_s \in S$ and $\langle q'_s, ??a, q_s \rangle \in R$,
  - if $q_s \in S \setminus S'$ then there exists $q'_s \in S'$ and $\langle q_s, ??a, q'_s \rangle \in R$.

2. *(broadcast from a state in S)* we have $q \in S$ and $q' \in S'$ (note that we could have that $q \in S'$ or $q \notin S'$), and the following conditions hold:
   - for all $j \in \{1, \ldots, \mathrm{psize}(\varphi)\}$ we have either $v[j] = v'[j]$ or there exists $\langle q_r, ??a, q'_r \rangle \in R$ such that $v[j] = q_r \wedge v'[j] = q'_r$;
   - for each $q_s \in Q \setminus \{q, q'\}$, we have:
     - if $q_s \in S' \setminus S$ then there exists $\langle q'_s, ??a, q_s \rangle \in R$ with $q'_s \in S$,
     - if $q_s \in S \setminus S'$ then there exists $\langle q_s, ??a, q'_s \rangle \in R$ with $q'_s \in S'$.

We denote by $\rightsquigarrow \subseteq \Theta \times \Theta$ the relation such that $\theta \rightsquigarrow \theta'$ if and only if there exists a rule $r \in R$ such that $\theta \rightsquigarrow_r \theta'$, and $\rightsquigarrow^*$ represents its reflexive and transitive closure. The intuition behind this construction is that we do not perform any abstraction on the states present in the vector v but only on the states present in S, this because the states present in v are used as witnesses to satisfy the cardinality constraint φ.

As an example, for $\mathrm{psize}(\varphi) = 5$, let $\langle (q_1, q_2, q_0, q_0, q_0), \{q_0, q_1, q_2\} \rangle$ be a symbolic configuration, and let $\langle q_1, !!a, q'_1 \rangle$ and $\langle q_2, ??a, q'_2 \rangle$ be two transition rules. With a broadcast from a process in the vector we may reach, among others:

- $\langle (q'_1, q_2, q_0, q_0, q_0), \{q_0, q_1, q_2\} \rangle$,
- $\langle (q'_1, q'_2, q_0, q_0, q_0), \{q_0, q_1, q_2, q'_2\} \rangle$, or
- $\langle (q'_1, q'_2, q_0, q_0, q_0), \{q_0, q_1, q'_2\} \rangle$,

whereas a broadcast from a process in the set may lead to:

- $\langle (q_1, q_2, q_0, q_0, q_0), \{q_0, q_1, q_2, q'_1\} \rangle$,
- $\langle (q_1, q_2, q_0, q_0, q_0), \{q_0, q_1, q'_1, q'_2\} \rangle$,
- $\langle (q_1, q'_2, q_0, q_0, q_0), \{q_0, q_1, q_2, q'_1, q'_2\} \rangle$, or
- $\langle (q_1, q'_2, q_0, q_0, q_0), \{q_0, q'_1, q'_2\} \rangle$.

We will now prove that the symbolic configurations are well-suited to solve CRP. First, we show that if a symbolic configuration which satisfies φ is reachable from the initial symbolic configuration, then there is a concrete configuration reachable from an initial configuration in γ_0 which also satisfies φ. This ensures a sound reasoning on symbolic configurations.

Lemma 7. *If there exists $\theta \in \Theta$ such that $\theta_0 \rightsquigarrow^* \theta$ and $\theta \models \varphi$, then $\mathcal{P} \models \Diamond\varphi$.*

Sketch of proof. For a symbolic $\theta = \langle v, S\rangle$ in Θ and $N \in \mathbb{N}$, we denote by $[\![\theta]\!]_N = \{\gamma \in [\![\theta]\!] \mid \forall q \in S.\#\gamma(q) > (N + \#v(q))\}$, i.e. the set of configurations which belong to $[\![\theta]\!]$ in which for each $q \in S$, there are at least N vertices in state q (in addition to those already in the vector v). Note that with this definition $[\![\theta]\!]_0 = [\![\theta]\!]$. We then can prove the following property: given $\theta \in \Theta$ such that $\theta_0 \rightsquigarrow^* \theta$, there exists $N \in \mathbb{N}$ such that for all $\gamma \in [\![\theta]\!]_N$, there exists an initial configuration $\gamma_0 \in \Gamma_0$ such that $\gamma_0 \to^* \gamma$. To show that this property is true, we reason by induction on the length of the execution choosing the N adequately at each step of the induction. Then if there exists $\theta \in \Theta$ such that $\theta_0 \rightsquigarrow^* \theta$ and $\theta \models \varphi$, then there exists $\gamma_0 \in \theta_0$ and $\gamma \in [\![\theta]\!]$ such that $\gamma_0 \to^* \gamma$, and by the definition of $\models$ for symbolic configuration we deduce also that $\gamma \models \varphi$. Hence $\mathcal{P} \models \Diamond\varphi$. □

More formally, in order to prove Lemma 7, we first prove Lemma 8.

Lemma 8. *For all $\theta \in \Theta$ such that $\theta_0 \rightsquigarrow^* \theta$, there exists $N \in \mathbb{N}$ such that for all $\gamma \in [\![\theta]\!]_N$, there exists an initial configuration $\gamma_0 \in \Gamma_0$ such that $\gamma_0 \to^* \gamma$.*

Proof. Let $\theta \in \Theta$ such that $\theta_0 \rightsquigarrow^* \theta$. Then there exists $\theta_1, \ldots, \theta_m$ in Θ such that $\theta_0 \rightsquigarrow \theta_1 \rightsquigarrow \ldots \rightsquigarrow \theta_m$ and $\theta_m = \theta$. We will prove the statement of the Lemma by induction on the length of this symbolic execution. The base case, when $m = 0$, is obvious since we have $[\![\theta_0]\!] \subseteq \Gamma_0$, it is then sufficient to take $N = 0$. Now let $m > 0$ and assume the property is true for all $i \in \{0, \ldots, m-1\}$, we will prove it holds for θ_m. First, by hypothesis of induction, we have that there exists N_{m-1} such that for all $\gamma \in [\![\theta_{m-1}]\!]_{N_{m-1}}$, there exists an initial configuration $\gamma_0 \in \Gamma_0$ such that $\gamma_0 \to^* \gamma$. Since $\theta_{m-1} \rightsquigarrow \theta_m$, there exists a rule $r \in R$ of the form $\langle q_1, !!a, q_2\rangle$ such that $\theta_{m-1} \rightsquigarrow_r \theta_m$. We now proceed by a case analysis on the application of the rule r to obtain the symbolic configuration $\theta_m = \langle v', S'\rangle$ from $\theta_{m-1} = \langle v, S\rangle$.

First, assume the sending process is in the vector v. There exists $i \in \{1, \ldots, \mathrm{psize}(\varphi)\}$ such that $v[i] = q_1$ and $v'[i] = q_2$. Let $N_m = (N_{m-1}+1)*(|S\backslash S'|+1)$ (where $|S\backslash S'|$ denotes the cardinality of the set $S \setminus S'$). Note that $N_m > N_{m-1}$. We will prove that for all $\gamma' \in [\![\theta_m]\!]_{N_m}$ there exists $\gamma \in [\![\theta_{m-1}]\!]_{N_{m-1}}$ such that $\gamma \to^+ \gamma'$. From γ', we will build a configuration γ in $[\![\theta_{m-1}]\!]_{N_{m-1}}$ such that $\gamma \to^+ \gamma'$. First, we need to divide the set of control states Q in different subsets. For each $q' \in S' \setminus S$, we know there exists $\langle q, ??a, q'\rangle \in R$ with $q \in S$; note that there might exist more than one such transition, but among them we choose one and we denote by $origin(q')$ the corresponding state q. Similarly for each $q \in S \setminus S'$, we know there exists $\langle q, ??a, q'\rangle \in R$ with $q' \in S'$; note that there might exist more than one such transition, but among them we choose

one and we denote by $destination(q)$ the corresponding state q'. For $q \in S$, we then define the set $From(q) = \{q' \in S' \setminus S \mid q = origin(q')\}$. Intuitively this set characterizes the control states which are newly appearing in S' and their presence is due to the reception of a from a vertex in state q. Similarly for $q' \in S'$ we define $To(q') = \{q'' \in S \setminus S' \mid q' = destination(q'')\}$, which intuitively characterizes the control state that disappears from S and their associated vertex changes their state to q'. The configuration γ should then verify the following requirements, for each $q \in Q$:

1. If $q \in S \cap S'$, then:

$$\#\gamma(q) = \#v(q) + \#\gamma'(q) - \#v'(q) + \sum_{q' \in From(q)} [(\#\gamma'(q') - \#v'(q')) - |To(q')| * (N_{m-1} + 1)]$$

2. If $q \in S \setminus S'$, then:

$$\#\gamma(q) = \#v(q) + (N_{m-1} + 1) + \sum_{q' \in From(q)} [(\#\gamma'(q') - \#v'(q')) - |To(q')| * (N_{m-1} + 1)]$$

3. If $q \notin S$, then $\#\gamma(q) = \#v(q)$

Note that the number of labels of vertices labelled by a given control state $q \in Q$ is the only relevant information that need to be considered because then thanks to the reconfiguration rule of the RBN, we can obtain any labeled graphs (i.e. the topology can be changed in any wished direction).

Let us now check that the requirements for γ ensure that $\gamma \in [\![\theta_{m-1}]\!]_{N_{m-1}}$, that is that we have for each $q \in Q$, if $q \notin S$, then $\#\gamma(q) = \#v(q)$ (this is guaranteed by the requirement **3.** on γ) and if $q \in S$, then $\#\gamma(q) > (N_{m-1} + \#v(q))$. We will now prove this last point. Let $q \in S$. First we will show that $\sum_{q' \in From(q)} [(\#\gamma'(q') - \#v'(q')) - |To(q')| * (N_{m-1} + 1)] \geq 0$. Let $q' \in From(q)$, since $q' \in S'$, we have that $\#\gamma'(q') - \#v'(q') > N_m$, hence by definition of N_m we deduce that $\#\gamma'(q') - \#v'(q') > (N_{m-1} + 1) * (|S \setminus S'| + 1)$, consequently $[(\#\gamma'(q') - \#v'(q')) - |To(q')| * (N_{m-1} + 1)] > (N_{m-1} + 1) * (|S \setminus S'| + 1 - |To(q')|)$. But since $To(q') \subseteq (S \setminus S')$, this allows us to deduce that $[(\#\gamma'(q') - \#v'(q')) - |To(q')| * (N_{m-1} + 1)] > 0$. We can now prove that for all $q \in S$, we have $\#\gamma(q) > (N_{m-1} + \#v(q))$. If $q \in S \setminus S'$, thanks to requirement **2.** and the previous consideration, it is obvious that $\#\gamma(q) > (N_{m-1} + \#v(q))$. If $q \in S \cap S'$, then thanks to requirement **1.** and to the previous consideration, we have that $\#\gamma(q) > \#v(q) + \#\gamma'(q) - \#v'(q)$, but since $q \in S'$, we have that $\#\gamma'(q) - \#v'(q) > N_m > N_{m-1}$, hence $\#\gamma(q) > \#v(q) > N_{m-1}$. This allows us to deduce that the configuration $\gamma \in [\![\theta_{m-1}]\!]_{N_{m-1}}$.

It remains now to check that $\gamma \rightarrow^+ \gamma'$. The main idea is that first we begin to use the non-deterministic reconfiguration to obtain a graph as we want in order to make the nodes react correctly to the broadcast. We will now explain why requirements **1.** to **3.** ensure that $\gamma \rightarrow^+ \gamma'$.

We will focus our attention on the vertices labelled with control states in S. Assume there is a control state q in S but not in S' (i.e. we analyze requirement 2.) For this control state we put at least $\#v(q) + (N_{m-1} + 1)$, one assumption we made is that the $(N_{m-1} + 1)$ will change their label to $destination(q)$, consequently thanks to this we are sure that there are at least $|To(destination(q))| * (N_{m-1}+1)$ vertices labelled by $destination(q)$ in γ'. Then however it might be the case that in γ, we need more nodes labelled by q because these nodes will change their label into some node q' that do not belong to S but to S'. This is the case for the nodes q' such that $q = origin(q')$, and the quantity of nodes labelled by q we need to add is precisely for each such label q': $[(\#\gamma'(q') - \#v'(q')) - |To(q')| * (N_{m-1} + 1)]$, i.e. the number of nodes labelled by q' in γ' to which is subtracted the number of nodes in γ labelled by a state in $S \setminus S'$ that change their state to q' (this quantity being $|To(q')| * (N_{m-1}+1)$ as we have already mentioned). For the control state $q \in S \cap S'$, the reasoning is similar. Hence what we have shown is that we choose the requirements **1.** and **2.** to ensure that $\gamma \rightarrow^+ \gamma'$. Finally since $\gamma \in [\![\theta_{m-1}]\!]_{N_{m-1}}$, by induction hypothesis, we have that there exists $\gamma_0 \in \Gamma_0$ such that $\gamma_0 \rightarrow^* \gamma$ and consequently we also deduce that $\gamma_0 \rightarrow^* \gamma'$. The case where the broadcast is performed by a node whose control state is taken from S can be treated in a similar way. □

We can now formally prove Lemma 7.

Proof of Lemma 7. Assume there exists $\theta \in \Theta$ such that $\theta_0 \rightsquigarrow^* \theta$ and $\theta \models \varphi$. From Lemma 8, we know that there exists $\gamma_0 \in \theta_0$ and $\gamma \in [\![\theta]\!]$ such that $\gamma_0 \rightarrow^* \gamma$, and by the definition of $\models$ for symbolic configuration we deduce also that $\gamma \models \varphi$. Consequently we have $\mathcal{P} \models \Diamond\varphi$. □

We will now show that a reasoning on symbolic configurations leads to completeness, in other words that if there is a reachable configuration that satisfies the cardinality constraint φ, then there is a reachable symbolic configuration that satisfies φ.

Lemma 9. *If $\mathcal{P} \models \Diamond\varphi$, then there exists $\theta \in \Theta$ such that $\theta_0 \rightsquigarrow^* \theta$ and $\theta \models \varphi$.*

Sketch of proof. In order to prove this Lemma, we need to introduce some auxiliary notations. Given a configuration $\gamma \in \Gamma$, we define $\uparrow_{q_0} \gamma$ as the set $\{\gamma' \in \Gamma \mid \forall q \in Q \setminus \{q_0\}.\ \#\gamma'(q) = \#\gamma(q)\}$. The above definition is needed because we could reach a configuration γ which does not have enough processes to be represented by a symbolic configuration, but we can complete it by adding new vertices labelled by the initial state q_0 in order to solve the problem. We can then prove the following property by induction on the length of the concrete execution: for $\gamma_0 \in \Gamma_0$ and $\gamma \in \Gamma$ such that $\gamma_0 \rightarrow^* \gamma$, for all $\theta \in \Theta$ verifying $\uparrow_{q_0} \gamma \cap [\![\theta]\!] \neq \emptyset$, we have $\theta_0 \rightsquigarrow^* \theta$. Basically, this property stipulates that given a reachable configuration γ, each symbolic configuration θ whose semantics $[\![\theta]\!]$ contains γ (modulo processes in state q_0) is also reachable.

The next step consists in proving that if $\gamma \in \Gamma$ is a configuration satisfying $\gamma \models \varphi$ then there exists $\theta \in \Theta$ such that $\uparrow_{q_0} \gamma \cap [\![\theta]\!] \neq \emptyset$ and $\theta \models \varphi$. This can be proved providing an algorithm

that builds $\theta = \langle v, S\rangle$ such that, for each $q \in Q$, either the processes in state q can be exactly represented within v only when $\#\gamma(q) \leq \mathrm{val}(\varphi)$, or $\#v(q) = \mathrm{val}(\varphi)$ and $q \in S$ when $\#\gamma(q) > \mathrm{val}(\varphi)$ (i.e. v is not large enough, recall that, apart for the states q_0 used to fill the "holes" in v, we reserve only up to $\mathrm{val}(\varphi)$ processes per state in v). Consider, e.g., a process with states $Q = \{q_0, q_1, q_2\}$, the formula $\varphi = 0 \leq \#q_1 < 3 \wedge 1 \leq \#q_2 < +\infty$ and the configuration with five processes $\gamma = \langle q_1, q_2, q_2, q_2, q_2\rangle$ such that $\gamma \models \varphi$. The symbolic configuration θ obtained is then $\langle (q_1, q_2, q_2, q_2, q_0, q_0, q_0, q_0, q_0), \{q_0, q_2\}\rangle$.

Since $\mathcal{P} \models \Diamond\varphi$, there exists an initial configuration $\gamma_0 \in \Gamma_0$ and a configuration $\gamma \in \Gamma$ such that $\gamma_0 \to^* \gamma$ and $\gamma \models \varphi$. By the second property we know there exists $\theta \in \Theta$ such that $\uparrow_{q_0} \gamma \cap [\![\theta]\!] \neq \emptyset$ and $\theta \models \varphi$, and the first property allows us to say that $\theta_0 \rightsquigarrow^* \theta$. □

Again, in order to have a formal proof for Lemma 9, we first prove some preliminary results.

Lemma 10. *Let $\gamma_0 \in \Gamma_0$ and $\gamma \in \Gamma$ such that $\gamma_0 \to^* \gamma$. Then for all $\theta \in \Theta$ such that $\uparrow_{q_0} \gamma \cap [\![\theta]\!] \neq \emptyset$, we have $\theta_0 \rightsquigarrow^* \theta$.*

Proof. Let $\gamma_0 \in \Gamma_0$ and $\gamma \in \Gamma$ such that $\gamma_0 \to^* \gamma$. Then there exists $\gamma_1, \ldots, \gamma_m$ in Γ such that $\gamma_0 \to \gamma_1 \to \ldots \to \gamma_m$ and $\gamma_m = \gamma$. We will prove the statement of the Lemma by induction on the length of this symbolic execution. The base case, when $m = 0$, is obvious since if $\uparrow_{q_0} \gamma_0 \cap [\![\theta]\!] \neq \emptyset$ implies that $\theta = \theta_0$ (in fact, we know that the only control state present in γ_0 is q_0).

Now let $m > 0$ and assume the property is true for all $i \in \{0 \ldots, m-1\}$, we will prove it holds for γ_m. Let θ_m such that $\uparrow_{q_0} \gamma_m \cap [\![\theta_m]\!] \neq \emptyset$. Since $\gamma_{m-1} \to \gamma_m$, either a reconfiguration rule or broadcast rule is applied to go from γ_{m-1} to γ_m. Assume a reconfiguration rule is applied, then only the edges change between γ_{m-1} and γ_m, hence we have $\uparrow_{q_0} \gamma_{m-1} \cap [\![\theta_m]\!] \neq \emptyset$ and by induction hypothesis we deduce that $\theta_0 \rightsquigarrow^* \theta$.

We now suppose that there exists a rule $r = \langle q, !!a, q'\rangle \in R$ such that $\gamma_{m-1} \to_r \gamma_m$. We assume that $\theta_m = \langle v_m, S_m\rangle$, $\gamma_{m-1} = \langle V, E, L\rangle$ and $\gamma_m = \langle V', E', L'\rangle$ and we will now build a symbolic configuration $\theta_{m-1} = \langle v_{m-1}, S_{m-1}\rangle$ such that $\uparrow_{q_0} \gamma_{m-1} \cap [\![\theta_{m-1}]\!] \neq \emptyset$ using that $\uparrow_{q_0} \gamma_m \cap [\![\theta_m]\!] \neq \emptyset$ and that $\gamma_{m-1} \to_r \gamma_m$. First we need to introduce some notations. Since we have $\uparrow_{q_0} \gamma_m \cap [\![\theta_m]\!] \neq \emptyset$, for each $i \in \{1, \ldots, \mathrm{psize}(\varphi)\}$, if $v_m[i] \neq q_0$, we can associate a unique vertex $node(i)$ in V' to it such that $L'(node(i)) = v_m[i]$ (on the domain of the elements of v different than q_0 the function $node$ is hence injective). Note that since $\gamma_{m-1} \to_r \gamma_m$, we have by definition of the transition relation in RBN, $V = V'$ and $E = E'$. We build the vector $v_{m-1} \in Q^{\mathrm{psize}(\varphi)}$, as follows, for each $i \in \{1, \ldots, \mathrm{psize}(\varphi)\}$:

- If $v_m[i] = q_0$ then $v_{m-1}[i] = q_0$;
- Otherwise, $v_{m-1}[i] = L(node(i))$.

After we build the set $S_{m-1} \subseteq Q$ as follows: $S_{m-1} = \{q_0\} \cup \{q \in Q \setminus \{q_0\} \mid \#\gamma_{m-1}(q) > \#v_{m-1}(q)\}$. By construction and using the definitions of $\rightarrow$ and $\rightsquigarrow$, we deduce easily the two following properties: $\uparrow_{q_0} \gamma_{m-1} \cap [\![\theta_{m-1}]\!] \neq \emptyset$ and $\theta_{m-1} \rightsquigarrow \theta_m$. Hence using the hypothesis of induction, we have also $\theta_0 \rightsquigarrow^* \theta_{m-1}$ which allows us to deduce that $\theta_0 \rightsquigarrow^* \theta_m$. □

Basically, Lemma 10 says that given a reachable configuration γ, each symbolic configuration θ whose semantics $[\![\theta]\!]$ contains γ (modulo processes in state q_0) is also reachable. We next formalize that such a θ actually exists whenever γ satisfies φ.

Lemma 11. *Let $\gamma \in \Gamma$ be a configuration such that $\gamma \models \varphi$. There exists $\theta \in \Theta$ such that $\uparrow_{q_0} \gamma \cap [\![\theta]\!] \neq \emptyset$ and $\theta \models \varphi$.*

Proof. Given a process $\mathcal{P} = \langle Q, \Sigma, R, \{q_0\}\rangle$, a formula φ, and a configuration $\gamma \in \Gamma$ such that $\gamma \models \varphi$, we build a symbolic configuration $\theta = (v, S) \in \Theta$ where $S = \{q_0\} \cup \{q \in Q \mid q \neq q_0,\ \#\gamma(q) > val(\varphi)\}$, v has size $\mathrm{psize}(\varphi)$, $\#v(q_0) \geq 0$, and for all $q \in Q \setminus \{q_0\}$, $\#v(q) = val(\varphi)$ if $q \in S$, or $\#v(q) = \#\gamma(q)$ otherwise. It is straightforward, by construction, that $\mathrm{psize}(\varphi) = |Q| * val(\varphi)$ is big enough to let the vector v contain all the generated processes, because each $q \in Q \setminus \{q_0\}$ will occur at most $val(\varphi)$ times, and q_0 will fill in the rest. Furthermore, for each $\gamma' \in [\![\theta]\!]$, $\#\gamma'(q_0) > 0$ and for every $q \in Q \setminus \{q_0\}$, either $q \notin S$ and $\#\gamma'(q) = \#\gamma(q)$, or $q \in S$ and $\#\gamma(q) > \#v(q)$, meaning that $\uparrow_{q_0} \gamma \cap [\![\theta]\!] \neq \emptyset$. We will now proceed by induction on the structure of the formula to show that $\gamma \models \varphi$ if and only if $\theta \models \varphi$.

Atom Let $\varphi = a \leq \#q < b$. By definition we have that $\gamma \models \varphi$ iff $a \leq \#\gamma(q) < b$. If $a \leq \#\gamma(q) < b$, then, for all $\gamma' \in [\![\theta]\!]$, either $\#\gamma(q) \leq val(\varphi)$ and $\#v(q) = \#\gamma(q) = \#\gamma'(q)$, or $\#\gamma'(q) > val(\varphi) \geq a$ and $b = +\infty$; in both cases, $a \leq \#\gamma'(q) < b$ (i.e. $\theta \models \varphi$). If $\theta \models \varphi$, then for every $\gamma' \in [\![\theta]\!]$, $a \leq \#\gamma'(q) < b$. By construction, $\#\gamma'(q)$ is either equal to $\#\gamma(q)$ when $b \neq +\infty$, or it is greater or equal than $\#\gamma(q)$ otherwise, meaning that $a \leq \#\gamma(q) < b$.

Conjunction Let $\varphi = \varphi_1 \wedge \varphi_2$. By definition, $\gamma \models \varphi$ iff $\gamma \models \varphi_1$ and $\gamma \models \varphi_2$, but from the inductive hypothesis $\gamma \models \varphi_1$ iff $\theta \models \varphi_1$ and $\gamma \models \varphi_2$ iff $\theta \models \varphi_2$, thus $\theta \models \varphi$ iff $\gamma \models \varphi$.

Disjunction The proof is almost identical to the one for conjunction.

Negation By definition, $\gamma \models \neg\varphi_1$ iff $\gamma \not\models \varphi_1$. We conclude that $\gamma \not\models \varphi_1$ iff $\theta \not\models \varphi_1$ by application of the inductive hypothesis.

□

We can now formally prove Lemma 9.

Proof of Lemma 9. Since $\mathcal{P} \models \Diamond\varphi$, there exist an initial configuration $\gamma_0 \in \Gamma_0$ and a configuration $\gamma \in \Gamma$ such that $\gamma_0 \rightarrow^* \gamma$ and $\gamma \models \varphi$. By Lemma 11 we know there exists $\theta \in \Theta$ such that $\uparrow_{q_0} \gamma \cap [\![\theta]\!] \neq \emptyset$ and $\theta \models \varphi$ and thanks to Lemma 10 we have that $\theta_0 \rightsquigarrow^* \theta$. □

We will now explain why CRP is in PSPACE. The main idea is that we can reason on the graph of symbolic configurations. Note that by definition, since $\Theta = Q^{\text{psize}(\varphi)} \times 2^Q$, the total number of symbolic configurations is $|\Theta| = |Q|^{\text{psize}(\varphi)} * 2^{|Q|}$. Furthermore, checking whether a symbolic configuration satisfies a cardinality constraint can be done in PTIME and checking whether two symbolic configurations belong to the symbolic transition relation $\rightsquigarrow$ can also be done in PTIME.

Lemma 12. *The following two properties hold:*

(i) *For any $\theta \in \Theta$ and any formula φ, $\theta \models \varphi$ can be decided in* PTIME.

(ii) *For all $\theta, \theta' \in \Theta$, $\theta \rightsquigarrow \theta'$ is decidable in* PTIME.

Proof. We first prove part (i) of the Lemma. Let $\psi = a \leq \#q < b$ be an atom of φ, and let $\theta = (v, S) \in \Theta$. Then, $\theta \models \psi$ if $b \neq +\infty$, $q \notin S$, and $a \leq \#v(q) < b$ or if $b = +\infty$ and $\#v(q) \geq a$. All of these tests can be done in PTIME in the size of φ. After assigning a truth value to each atom, the truth of the whole formula can be computed in PTIME, as an instance of the Circuit Value Problem.

Let us now proceed to prove part (ii). The proof will follow the definition of $\rightsquigarrow \subseteq \Theta \times \Theta$. Let $\theta = (v, S)$ and $\theta' = (v', S')$ be two symbolic configurations, and let $\langle q, !!a, q' \rangle \in R$ be a broadcast rule. First, let us consider the case of a broadcast from the vector v.

For all $i \in \{1, \ldots, \text{psize}(\varphi)\}$, there are three possibilities:

- $v[i]$ is the only sending process such that $v[i] = q$ and $v'[i] = q'$ (constant time);
- there exists a rule $\langle q_r, ??a, q'_r \rangle \in R$ such that $v[i] = q_r$; and $v'[i] = q'_r$ (linear in $|R|$);
- $v[i] = v'[i]$ (constant time).

Checking the conditions on all the elements in vector v takes overall no more than $|R| * \text{psize}(\varphi)$ steps. For each $q_s \in Q$, we have two conditions to check on the sets S and S': if $q_s \in S' \setminus S$, then there is a $q'_s \in S$ such that $\langle q'_s, ??a, q_s \rangle \in R$; if $q_s \in S \setminus S'$, then there is a $q'_s \in S$ such that $\langle q_s, ??a, q'_s \rangle \in R$. Both membership of states and existence of rules can be computed in linear time in the size of the process, thus checking the conditions for S and S' is in PTIME too. The case of a broadcast from the set S can be handled in a similar way. The Lemma follows from observing that all we need to do, is to repeat the PTIME check for all broadcast rules $r \in R$. □

```
Input: A process P = ⟨Q, Σ, R, {q_0}⟩, a formula φ over P.
Output: Does P ⊨ ◇φ?
  θ ← θ_0
  for i = 0 to |Q|^psize(φ) * 2^|Q|
    guess θ' ∈ Θ
    if θ ⇝ θ' then
      if θ' ⊨ φ
        return true
      else
        θ ← θ'
      end if
    end if
  end for
  return false
```

Listing 4.3: Solving CRP

The PSPACE algorithm in Listing 4.3 (which is in reality an NPSPACE algorithm) at each step guesses a new symbolic configuration, checks whether it is reachable from the previous guessed one and verifies whether it satisfies φ. When it encounters a symbolic configuration that satisfies φ, it returns that $\mathcal{P} \models \Diamond\varphi$. Note that this algorithm needs only to store the number of configurations it has seen until now, and when this number reaches $|Q|^{\mathrm{psize}(\varphi)} * 2^{|Q|}$, it means that the algorithm have seen all the symbolic configurations. Hence to store this number and the current and next symbolic configurations, the algorithm needs polynomial space (a number smaller than $|Q|^{\mathrm{psize}(\varphi)} * 2^{|Q|}$ can be stored into a counter which requires at most $\mathrm{psize}(\varphi) * \log(|Q|) + |Q| \log(2)$ space). Finally, Lemma 7 and Lemma 9 ensure us that such an algorithm is sound and complete and from Proposition 8 we have also a lower bound for CRP. Hence we deduce the main result of this work.

Theorem 12. *CRP is* PSPACE*-complete.*

Chapter 5

Asynchronous Broadcast Networks

We present (un)decidability and complexity results for the coverability problem of Asynchronous Broadcast Networks (ABN), a mathematical model of distributed systems in which processes interact via topology-dependent and asynchronous communication. Our formal model of asynchronous broadcast communication combines three main features: a graph representation of a network configuration decoupled from the specification of individual process behaviour, a topology-dependent semantics of synchronization, and the use of local mailboxes to deliver messages to individual nodes. Our main abstraction comes from considering protocols defined via a communicating finite-state automaton replicated on each node of the network.

Our analysis is carried out with different policies to handle buffers, namely unordered bags (an abstraction of a tuple space), and perfect or lossy FIFO channels. Our technical contribution is as follows. We first show that, in contrast with the synchronous case discussed in [DSZ10, DSZ11], coverability is decidable when local buffers are unordered. For the proof, we first give a reduction to the restricted case of fully connected topologies. We then solve the coverability problem through a reduction to the Cardinality Reachability Problem for Reconfigurable Broadcast Networks, the PTIME-complete problem shown in Section 4.2

When mailboxes are ordered buffers, we obtain undecidability already in the case of fully connected topologies. The undecidability proof is based on a non-trivial encoding of the set of operations of a two counter machine in form of a cooperation protocol between distinct nodes. The protocol consists of different phases, each one is defined over a distinct set of control messages. The difficulty of the encoding comes from the fact that it is not possible to infer well-formedness properties for the content of the mailbox of an individual node. Thus it is not possible to encode the current value of the counters using the current content of a set of mailboxes. The current value of the counters is represented however in the flow of messages consumed by a pair of nodes elected in a preliminary phase of the protocol, which, in turn, completes successfully only under certain conditions on the sequence of consumed control messages. The coverability prob-

lem is decidable when introducing non-deterministic message losses. The results again follows from a reduction to the Cardinality Reachability Problem for RBN.

In an extended model in which a node can test if its mailbox is empty, we obtain undecidability with unordered bags and both arbitrary or fully-connected topologies. For this reduction we need to control the interferences due to the simultaneous communication with several neighbours. We exploit here the emptiness test in order to enforce the well-formedness of the mailboxes of nodes involved in the simulation of counter machines.

To our knowledge, the present work, published in [DT12, DT13a], shows the first complexity analysis for (parameterized) coverability in formal models of asynchronous broadcast communication.

Acknowledgements The author would like to thank Prof. Giorgio Delzanno, co-author of the articles [DT12, DT13a] from which this chapter is taken.

Related Work

The literature on formal models of distributed protocols spans from approaches based on process algebra and Petri nets to approaches based on automata or term rewriting. We focus our attention on infinite-state models for which verification analysis present good computational properties.

In the context of extensions of Petri nets, the Broadcast Protocols (BP) of [EN98] introduced features like synchronous broadcast communication over a pool of identical processes. Every process can listen to any message. For this property, BPs have been used to model cache coherence protocols in [Del03]. Broadcast communication is used to model the propagation of a message in a shared bus.

Coverability in BPs is strictly related to marking coverability in Petri nets with transfer or reset arcs [ADB11]. While the problem is non-elementary for PBs [EFM99, DEP99], we will show that in the positive fragments of ABN we have polynomial time procedures. These results are based on similar procedures derived for RBN.

When individual processes are distributed over graphs of arbitrary shape, coverability becomes undecidable as shown in [DSZ10]. The model described in [DSZ10] is called AHN (Ad Hoc Networks) is viewed as an ideal model for Ad Hoc communication. To obtain decidability of coverability, it is necessary to restrict the conformation of the communication links. For instance, decidability holds for special classes of graphs like bounded path graphs under the induced graph ordering [DSZ10, DSZ11]. Decidability also holds in presence of communication failures (e.g. message loss) or interferences [DSZ12], and with dynamic reconfiguration of the communication topology as in the RBN (Reconfigurable Broadcast Networks) model in [DSTZ12b]. In RBN reconfiguration is modelled via a non-deterministic rule that can modify the topology of the

graph in arbitrary way.

The PTIME decision procedure in [DSTZ12b] is similar to the labelling algorithms used for parameterized verification of synchronous systems in [GS92]. In the timed case coverability becomes undecidable already in special types of star topologies [ADR$^+$11].

Other formal models of broadcast communication have been proposed in [Pra95, SRS10, EM01, FvGH$^+$12b]. Verification of unreliable communicating FIFO systems have been studied in [AJ96, CFI96]. In [CS08] the authors consider different classes of topologies with mixed lossy and perfect channels.

The complexity of the verification procedures for lossy FIFO channel systems and broadcast protocols (transfer and reset nets) is discussed in [Sch10]. A classification of the expressive power of different infinite-state models including lossy FIFO channel systems and broadcast protocols is discussed in [ADB11].

This chapter presents a revised and extended version of the paper appeared at LATA 2013 [DT13a]. Compared to [DT13a], we have expanded the description of the model and of the semantics, the section on related work, and included technical proofs for all the presented results.

5.1 Asynchronous Broadcast Network (ABN)

In this section we formally define our asynchronous model for broadcast communication. A configuration is defined as a labelled graph. Nodes correspond to processes running a common, pre-defined protocol. Each node has a local message buffer used to collect messages sent by neighbours.

A protocol is specified via a finite-state automaton with send and receive operations that correspond to write [resp. read] on remote [resp. local] buffers. Communication is topology-dependent, anonymous and asynchronous: when a process at node n sends a message a, the process does not block, and the message is added to the local mailbox of all of its neighbours without explicit information about the sender (i.e. messages do not contain node identifiers).

Formally, we consider a finite set Σ of messages, and different disciplines for handling the mailbox (message buffer), e.g., unordered mailboxes that we represent as bags over Σ, and ordered mailboxes that we represent as words over Σ.

In order to deal in a uniform way with different mailbox types we define a transition system parametric on the data structures used to model mailboxes. More specifically, we consider a mailbox structure $\mathbb{M} = \langle \mathcal{M}, del?, add, del, [] \rangle$, where $\mathcal{M}$ is a denumerable set of elements denoting possible mailbox contents; for $a \in \Sigma$ and $m \in \mathcal{M}$, $add(a, m)$ denotes the mailbox obtained by adding a to m, $del?(a, m)$ is true if a can be removed from m; $del(a, m)$ denotes the mailbox

obtained by removing a from m when possible, undefined otherwise. Finally, $[] \in \mathcal{M}$ denotes the empty mailbox. We call an element a of m $visible$ when $del?(a, m) = true$. Their specific semantics and corresponding properties change with the type of mailbox considered.

Definition 9. *A protocol is defined by a process $\mathcal{P} = \langle Q, \Sigma, R, q_0 \rangle$, where Q is a finite set of control states, Σ is a finite message alphabet, $Act = \{\tau\} \cup \{!!a, ??a \mid a \in \Sigma\}$, $R \subseteq Q \times Act \times Q$ is a set of transition rules, $q_0 \in Q$ is an initial control state.*

The label τ represents the capability of performing an internal action, and the label $!!a$ [$??a$] represents the capability of broadcasting [receiving] a message $a \in \Sigma$.

Definition 10. *Configurations are undirected $(Q \times \mathcal{M})$-graphs. A $(Q \times \mathcal{M})$-graph γ is a tuple $\langle V, E, L \rangle$, where V is a finite set of nodes, $E \subseteq V \times V$ is a finite set of edges (such that E is symmetric and $\forall v \in V.(v, v) \notin E$), and $L : V \to (Q \times \mathcal{M})$ is a labelling function.*

In the rest of the chapter, for an edge $\langle u, v \rangle$ in E, we use the notation $u \sim_\gamma v$ and say that the vertices u and v are adjacent to one another in γ. We omit γ, and simply write $u \sim v$, when it is made clear by the context. We use $L(\gamma)$ to represent the set of labels in γ. The set of all configurations is denoted Γ, while $\Gamma_0 \subseteq \Gamma$ is the set of all initial configurations, in which nodes always have the same label $\langle q_0, [] \rangle$.

Given the labelling L and the node v s.t. $L(v) = \langle q, m \rangle$, we define $L_s(v) = q$ (state component of $L(v)$) and $L_b(v) = m$ (buffer component of $L(v)$). Furthermore, for $\gamma = \langle V, E, L \rangle \in \Gamma$, we use $L_s(\gamma)$ to denote the set $\{L_s(v) \mid v \in V\}$.

Definition 11. *For $\mathbb{M} = \langle \mathcal{M}, del?, add, del, [] \rangle$, an Asynchronous Broadcast Network (ABN) associated to $\mathcal{P}$ is a tuple $\mathcal{T}(\mathcal{P}, \mathbb{M}) = \langle \Gamma, \Rightarrow_{\mathbb{M}}, \Gamma_0 \rangle$, where $\Rightarrow_{\mathbb{M}} \subseteq \Gamma \times \Gamma$ is the transition relation defined next. For $\gamma = \langle V, E, L \rangle$ and $\gamma' = \langle V, E, L' \rangle$, $\gamma \Rightarrow_{\mathbb{M}} \gamma'$ holds iff one of the following conditions on L and L' holds:*

Local *There exists $v \in V$ such that $(L_s(v), \tau, L'_s(v)) \in R$, $L_b(v) = L'_b(v)$, and $L(u) = L'(u)$ for each $u \in V \setminus \{v\}$.*

Broadcast *There exists $v \in V$ and $a \in \Sigma$ such that $(L_s(v), !!a, L'_s(v)) \in R$, $L_b(v) = L'_b(v)$ and for every $u \in V \setminus \{v\}$*

- *if $u \sim v$ then $L'_b(u) = add(a, L_b(u))$ and $L_s(u) = L'_s(u)$,*
- *otherwise $L(u) = L'(u)$.*

Receive *There exists $v \in V$ and $a \in \Sigma$ such that $(L_s(v), ??a, L'_s(v)) \in R$, $del?(a, L_b(v))$ is satisfied, $L'_b(v) = del(a, L_b(v))$, and $L(u) = L'(u)$ for each $u \in V \setminus \{v\}$.*

The rationale behind the previous definition is as follows. A local transition only affects the state of the process that executes it. The selected node updates its local state, while the state of all other nodes remain unchanged. In a broadcast transition the sender node changes state and adds the message to the mailbox of all of its neighbours. Notice that broadcast is never blocking for the sender. Receivers can read the message in different instants. This models asynchronous communication. A reception of a message a is blocking for the receiver whenever the buffer is empty or the visible elements are all different from a. If a is visible in the mailbox, the message is removed and the process moves to the next state. Furthermore, it is easy to show that, when needed, a set $Q_0 \subseteq Q$ of initial states for $\mathcal{P}$ can be modelled by introducing a fresh initial state with outgoing local transitions to each $q \in Q_0$.

An *execution* is a sequence $\gamma_0\gamma_1 \ldots$ such that γ_0 is an initial configuration, and $\gamma_i \Rightarrow_{\mathbb{M}} \gamma_{i+1}$ for $i \geq 0$. We use $\Rightarrow^*_{\mathbb{M}}$ to denote the reflexive and transitive closure of $\Rightarrow_{\mathbb{M}}$. We drop $\mathbb{M}$ when the mailbox type is clear from the context.

Example 2. *In the figures from 5.2 to 5.4 we show a few executions of an hypothetical protocol (in Figure 5.1) in order to give some examples on the runtime behaviour of an ABN. We represent with natural numbers the different local control states of individual process, and with square brackets the contents of mailboxes, where the leftmost item is to be considered the first one in case of ordered queues. Depending on the policy considered, some possibilities may be forbidden.*

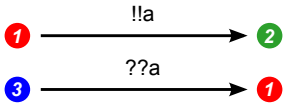


Figure 5.1: Rules used in the examples from Figure 5.2 to 5.4

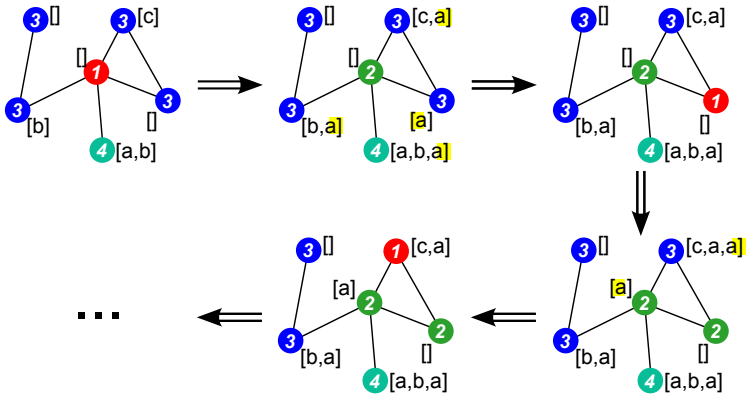


Figure 5.2: Execution with Bag mailboxes

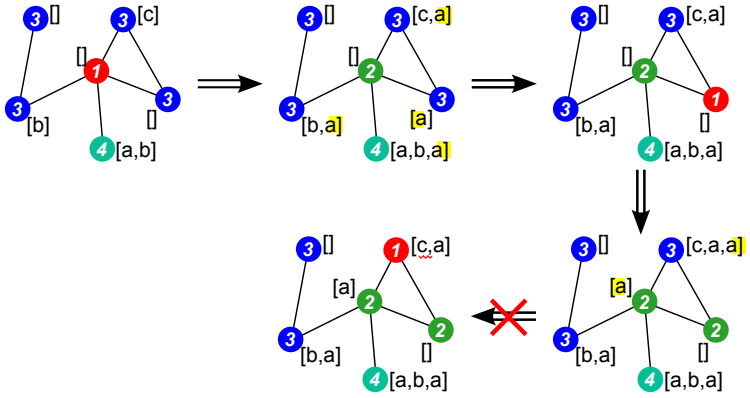


Figure 5.3: Execution with $FIFO$ mailboxes

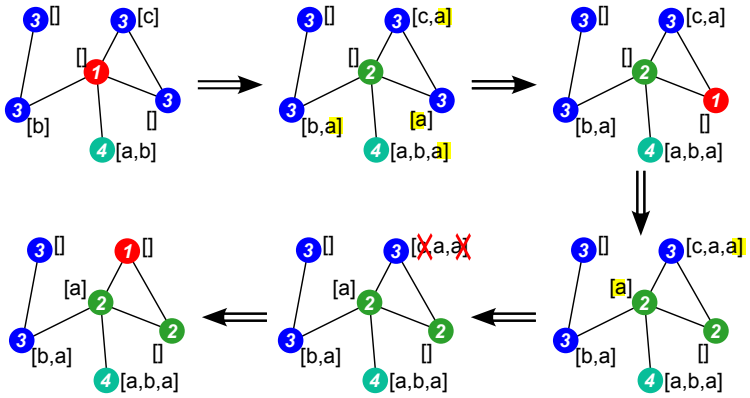


Figure 5.4: Execution with $LFIFO$ mailboxes

Decision Problem

The *Coverability Problem* parametric on the mailbox structure $\mathbb{M}$, abbreviated as $COV(\mathbb{M})$, is defined as follows.

Definition 12. *Given a protocol $\mathcal{P}$ with transition system $\mathcal{T}(\mathcal{P}, \mathbb{M}) = \langle \Gamma, \Rightarrow_{\mathbb{M}}, \Gamma_0 \rangle$ and a control state q, the coverability problem $COV(\mathbb{M})$ states: are there two configurations $\gamma_0 \in \Gamma_0$ and $\gamma_1 \in \Gamma$ such that $\gamma_0 \Rightarrow^*_{\mathbb{M}} \gamma_1$ and $q \in L_s(\gamma_1)$?*

In other words we require that a graph γ_q with a singleton node labelled q covers a reachable configuration γ_1, i.e., γ_q is a subgraph of γ_1. We often use the terminology γ_0 reaches state q as an abbreviation for $\gamma_0 \Rightarrow^*_{\mathbb{M}} \gamma_1$ and $q \in L_s(\gamma_1)$ for some configuration γ_1. Besides being parametric on the mailbox structure, our decision problem is parametric on the shape of the initial configuration. As mentioned in the introduction, this feature models in a natural way verification problems for protocols with partial information about the structure of the network.

5.1.1 Relations with the RBN Model

In the rest of the chapter we will often refer to the synchronous model called RBN, described in Chapter 4. We remark that protocols in RBN follow the same syntax used in ABN, but configurations are simply Q-graphs, i.e., graphs in which nodes have labels in Q via the labelling function L (there is no mailbox associated to individual nodes). In the following sections, we will exploit verification algorithms designed for RBN for solving coverability in ABN. In particular, the problem of interest is the one presented for the Cardinality Reachability problem (CRP) restricted to positive queries (Section 4.2), which is shown to be PTIME-complete.

5.2 Unordered Mailboxes

In this section we study the coverability problems for ABNs in which mailboxes are unordered buffers modelled as bags over the finite message alphabet Σ. The mailbox structure Bag is defined as follows.

Definition 13. *Bag is the denumerable set of bags over Σ, $add(a, m) = [a] \oplus m$ (multiset sum of the singleton $[a]$ and m), $del?(a, m) = true$ iff $m(a) > 0$, $del(a, m) = m \ominus [a]$ (multiset removal of $[a]$ from m), and $[] \in Bag$ is the empty bag $[]$.*

The operational semantics follows from the general definitions. Let us consider the instance $COV(Bag)$ of the coverability problem. For synchronous broadcast, coverability is undecidable for arbitrary topologies [DSZ10]. We show next that coverability is in PTIME for unordered mailboxes.

Terminology: For the ease of notation, we use $\mathcal{T}^{\mathcal{K}}(\mathcal{P}, \mathbb{M})$ [resp. $COV_{fc}(\mathbb{M})$] to denote the restriction of $\mathcal{T}(\mathcal{P}, \mathbb{M})$ [resp. $COV(\mathbb{M})$] to fully connected configurations only, i.e., configurations such that $u \sim_\gamma v$ for each pair of distinct nodes $u, v \in V$.

We prove the results in two different steps. We first show that, for the purpose of deciding $COV(Bag)$, we can focus on fully connected topologies only. We then show a reduction from $COV_{fc}(Bag)$ to the Cardinality Reachability Problem for RBN.

5.2.1 Focusing on Fully Connected Topologies

To prove the first property, we need some auxiliary lemmas stating properties of ABN computations. First of all, we show that the application of ABN rules is monotonic w.r.t. the content of the mailbox in each node.

Lemma 13. *Let $\mathcal{P} = \langle Q, \Sigma, R, q_0 \rangle$ be a protocol and let $\gamma = \langle V, V \times V, L \rangle$ and $\gamma' = \langle V, V \times V, L' \rangle$ be two configurations from the transition system $\mathcal{T}(\mathcal{P}, Bag)$ such that $\gamma \Rightarrow_{\mathbb{M}} \gamma'$ by applying some rule $r \in R$ to a node $v \in V$. Given another configuration $\delta = \langle V, V \times V, K \rangle$ such that, for all $u \in V$, $L_s(u) = K_s(u) \wedge L_b(u) \subseteq^b K_b(u)$, we can apply r at node v in order to reach any configuration $\delta' = \langle V, V \times V, K' \rangle$ such that $L'_s(u) = K'_s(u)$ and $L'_b(u) \subseteq^b K'_b(u)$ for all $u \in V$.*

Proof. The proof is by cases on the type of r. Specifically, for all $u \in V$ we have that

- if $r = (L_s(v), \tau, L'_s(v))$ then $L'_b(u) = K'_b(u)$ for every $u \in V$;
- if $r = (L_s(v), !!a, L'_s(v))$ then $L'_b(v) = K'_b(v)$ and $K'_b(u) = add(a, K_b(u))$ for every $u \in V \setminus \{v\}$;
- if $r = (L_s(v), ??a, L'_s(v))$ then $del?(a, K_b(v))$ holds because $L_b(v) \subseteq^b K_b(v)$ and by hypothesis $del?(a, L_b(v))$ is satisfied, $L'_b(v) = del(a, K_b(v))$, and $L'_b(u) = K'_b(u)$ for every remaining $u \in V \setminus \{v\}$.

□

We now show how to find fully connected topologies, starting from coverability in arbitrary graphs. We exploit the fact that mailboxes are unordered to ignore messages sent along links that are not present in a given topology. Formally, the following property holds.

Lemma 14. *Given an ABN protocol $\mathcal{P} = \langle Q, \Sigma, R, q_0 \rangle$ and a state $q \in Q$, if there exists an arbitrary topology from which we can reach state q, then there exists a fully connected topology from which we can also reach q.*

Proof. We show by induction that, given an execution $\gamma_0\gamma_1 \dots \gamma_k$ in $\mathcal{T}(\mathcal{P}, Bag)$ where $q \in L_s(\gamma_k)$ and $\gamma_0 = \langle V, E, L^0 \rangle$ we can build another one $\delta_0\delta_1 \dots \delta_k$ in $\mathcal{T}^{\mathcal{K}}(\mathcal{P}, Bag)$ by starting from $\delta_0 = \langle V, V \times V, L^0 \rangle$ and replicating each rule application as shown in the previous scheme.

- The base case is immediate since initial configurations are graphs in which nodes have the same label, i.e., we can always find a sufficiently large fully connected initial configuration that contains any initial configuration (of arbitrary topology).
- For the inductive step, let $\gamma_0\gamma_1 \dots \gamma_k\gamma_{k+1}$ be an execution in $\mathcal{T}(\mathcal{P}, Bag)$. From the inductive hypothesis, we know that there exists $\delta_0\delta_1 \dots \delta_k$ such that the mailboxes in the nodes of δ_i contain the mailboxes in the nodes of γ_i for every i. To conclude the proof, we extend the property to $k+1$ steps by choosing an adequate, w.r.t. lemma 13, successor configuration δ_{k+1} of δ_k.

The thesis then follows by observing that, because of the construction, the set of control states in δ_i is the same as those in γ_i for $0 \leq i \leq k$ and in particular $q \in K_s(\gamma_k)$. □

From Lemma 14 and by observing that if there exists a fully connected initial configuration that reaches a configuration in which state q occurs, then coverability is solved, we obtain the following characterization of coverability in arbitrary graphs.

Lemma 15. *Given an ABN protocol $\mathcal{P} = \langle Q, \Sigma, R, q_0 \rangle$ and a state $q \in Q$, there exists an arbitrary topology from which we can reach state q if and only if there exists a fully connected topology from which we can also reach q.*

5.2.2 Reduction to Cardinality Reachability in RBN

We now define a formal link between ABN and RBN. Again we need some auxiliary properties, this time on the computations with the synchronous broadcast semantics of RBN.

In the first lemma we show that for every state q that occurs in some intermediate state during a computation σ of a RBN we can build an execution σ' that contains (in a sense specified below) the same sequence of steps as those in σ and that exposes q in its final configuration.

Lemma 16. *Let $\pi = \theta_0\theta_1 \dots \theta_n$ be an execution of an RBN with nodes in V and $\theta_n = \langle V, E, L \rangle$. For every state q that appears in at least one of the configurations of π, there exists another execution $\pi' = \theta'_0\theta'_1 \dots \theta'_m$ with nodes in $V' \supseteq V$ and $\theta'_m = \langle V', E', L' \rangle$ such that $L'(v) = L(v)$ for all $v \in V$ and there exists $v' \in V' \setminus V$ such that $L'(v') = q$.*

Proof. Let $\theta_0 = \langle V, E_0, L_0 \rangle$. We take $V' = V \cup (V \times \{1\})$ nodes and we show how to build the execution π' by having two replicas of the given execution π running in parallel. We start

from the initial configuration $\theta'_0 = \langle V', E'_0, L'_0 \rangle$ such that $L'_0(v) = L'_0((v,1)) = L_0(v)$ and $E'_0 = \{(v_1, v_2), ((v_1,1),(v_2,1)) \mid (v_1, v_2) \in E_0\}$. It follows from the semantics of the RBN that we have an execution $\theta'_0\theta'_1 \dots \theta'_n$ where the nodes $v \in V$ mimics π step by step, and leaves every other $v' \in V' \setminus V$ in its initial state. Now let $i \in \{0, \dots, n\}$ be the index of the first configuration θ_i that exposes a process in state q. The other half of the network, $V' \setminus V$, may now proceed mimicking the execution $\theta_0 \dots \theta_i$ without involving the nodes in V. We conclude that the Lemma holds by remarking that $\pi' = \theta'_0 \dots \theta'_n \dots \theta'_{n+i}$ satisfies all requirements. □

The following main lemma formally relates coverability in ABN to the Cardinality Reachability Problem in RBN.

Lemma 17. *Given an ABN protocol $\mathcal{P} = \langle Q, \Sigma, R, q_0 \rangle$ and a state $q \in Q$ let $\mathcal{P}'$ be the RBN protocol with the same rules but with $\{q_0\}$ as singleton set of initial states. Then, there exists an execution of $\mathcal{P}'$ that satisfies CRP with Cardinality query $q \geq 1$ if and only if there exists an execution of $\mathcal{P}$ satisfying $COV_{fc}(Bag)$.*

Proof. The rationale behind the proof is as follows. On one side we can arbitrarily delay message receptions to simulate link deletion (1). Vice versa, we can exploit reconfigurations and the possibility of adding nodes to the initial configuration to simulate asynchronous receipts using dynamically created links and synchronous messages (2).

(1) Formally, let $\theta_0 \to \theta_1 \to \dots \to \theta_m$ be an execution of $\mathcal{P}'$ such that $\theta_i = \langle V, E'_i, L'_i \rangle$ for all $i \in \{1, \dots, m\}$. We now proceed by induction on m to prove that there exists an execution $\gamma_0 \Rightarrow \dots \Rightarrow \gamma_n = \langle V, E, L^n \rangle$ such that $\forall v \in V$, $L^n_s(q) = L'_m(q)$.

- For $m = 0$, the configuration $\gamma_0 \in \Gamma_0$ with vertices V trivially satisfies the thesis.
- For $m \geq 1$, from the inductive hypothesis we know that given an execution $\theta_0 \to \dots \to \theta_{m-1}$ of $\mathcal{P}'$ we have another one $\gamma_0 \Rightarrow \dots \Rightarrow \gamma_k$ of $\mathcal{P}$ such that $L^k_s(q) = L'_{m-1}(q)$ for all $v \in V$.

  We want to prove that, given $\theta_m \in \Theta$ such that $\theta_{m-1} \to \theta_m$, there exists $\gamma = \langle V, E, L \rangle \in \Gamma$ such that $\gamma_k \Rightarrow^* \gamma$ and $L_s(\gamma) = L'(\theta_m)$. In the case when $\theta_{m-1} \to \theta_m$ because of a graph reconfiguration, then the thesis follows immediately by considering that $\gamma_k \Rightarrow^* \gamma$. In all other cases there exists a rule $r = \langle q, !!a, q' \rangle \in R$ such that $\theta_{m-1} \to \theta_m$ thanks to a broadcast by a node $v \in V$. We build the execution $\gamma_k \Rightarrow \gamma_{k+1} \Rightarrow \dots \Rightarrow \gamma_{k+1+r}$ where $\gamma_k \Rightarrow \gamma_{k+1}$ corresponds to an application of rule r to the same node v that also occurs in γ_k. As a consequence, $L_b(u)$ contains the message a for each node $u \sim v$ in γ_{k+1}. The remaining r transitions are now reception steps, one for each of the r neighbours $u \in V$ of v that are enabled to react to the message sent by v. Finally, by choosing for the r steps the same exact reception rules that fired during the synchronous broadcast step, we obtain that for $\gamma = \gamma_{k+1+r} = \langle V, E, L \rangle$ we have that $L_s(v) = L'_m(v)$ for all $v \in V$. All

those rules are necessarily enabled because the local state of individual processes by the inductive hypothesis matches the one in θ_{m-1} and, since their mailboxes are unordered, processes may indefinitely ignore previously received messages.

(2) In order to prove the second implication we will show how we can rely on graph reconfigurations (instead of unordered mailboxes) in order to simulate the execution of an ABN. Let $\gamma_0 \Rightarrow \gamma_1 \Rightarrow \cdots \Rightarrow \gamma_n = \langle V, E, L^n \rangle$ be an execution of $\mathcal{P}$. We show by induction on n that there exists an execution $\theta_0 \to \theta_1 \to \cdots \to \theta_m$ of $\mathcal{P}'$ (with $\theta_i = \langle V', E'_i, L'_i \rangle$ for all $i \in \{1, \ldots, m\}$), such that $V \subseteq V'$ and $\forall v \in V$, $L'_m(v) = L^n_s(v)$. In both semantics the sets of nodes cannot change during an execution, thus we will call V the set of nodes of the ABN and V' the set of nodes of the RBN.

- For $n = 0$, every configuration $\theta_0 \in \Theta_0$ with vertices $V' \supseteq V$ satisfies the thesis.
- Now, let us consider $n \geq 1$. We know by hypothesis that given an execution $\gamma_0 \Rightarrow^* \gamma_{n-1}$ of $\mathcal{P}$, there exists an execution $\theta_0 \to^* \theta_{m'}$ of $\mathcal{P}'$ such that $V \subseteq V'$ and $\forall v \in V$, $L'_{m'}(v) = L^{n-1}_s(v)$, and we want to prove that, given $\gamma_n \in \Gamma$ such that $\gamma_{n-1} \Rightarrow \gamma_n$, there exists an execution $\theta'_0 \to^* \theta'_{m''}$ of $\mathcal{P}'$ such that if $\theta'_{m''} = \langle V'', E'', L'' \rangle$ then $V \subseteq V''$ and $L''(v) = L^n_s(v)$ for all $v \in V$. There are three cases to consider, namely local step, asynchronous broadcast and receive. In the first two cases, it suffices to note that by first removing all edges of $\theta_{m'}$ through a reconfiguration, the execution of the same step as in the ABN does not involve any other node. The configuration $\theta_{m'+2} = \langle V', E'_{m'+2}, L'_{m'+2} \rangle$ reached through these two steps, thanks to the inductive hypothesis, easily satisfies all requirements. In the third case, reception steps, we will need a slightly more complex construction. Let $r = \langle q_1, ??a, q_2 \rangle \in R$ be the rule fired during the step $\gamma_{n-1} \Rightarrow \gamma_n$. Since $\gamma_0 \ldots \gamma_n$ is an execution, we know that there exist a natural $i \in \{0, \ldots, n\}$ and a state $q_3 \in L_s(\gamma_i)$ such that there exists a rule $r' = \langle q_3, !!a, q_4 \rangle \in R$. By applying the inductive hypothesis, some configuration of the execution $\theta_0 \ldots \theta_{m'}$ must also expose a node in state q_3, thus, by Lemma 16, we have another execution $\theta'_0 \ldots \theta'_{m''}$ with nodes in $V'' \supseteq V'$ and $\theta'_{m''} = \langle V'', E'', L'' \rangle$ such that $L''(v) = L'_{m'}(v)$ for all $v \in V'$ and there exists $u \in V'' \setminus V'$ such that $L''(u) = q_3$. We can then reconfigure the set of edges of $\theta'_{m''}$ to $\{(u, v), (v, u)\}$ and let this new node u to synchronize through r' with the node $v \in V$ that fires the reception rule r in the ABN.

□

The reduction is done in constant time, since there is no need of modifying the protocol specification.

5.2.3 Complexity of Coverability

We can now apply the above proved properties to infer the complexity of coverability in ABN.

Theorem 13. $COV(Bag)$ *is* PTIME*-complete.*

Proof. Thanks to Lemmas 15 and 17 and to the PTIME algorithm for coverability in RBN [DSTZ12b], we know that $COV(Bag)$ is in PTIME. Completeness follows from a reduction of the Circuit Value Problem (CVP) [Lad75] to $COV(Bag)$. Given an acyclic circuit G composed by a finite set of gates and a fixed evaluation of its inputs, CVP consists in evaluating G in the inputs. The reduction is based on a protocol in which a special node broadcasts the evaluation of a single input (in form of a message with label $true/false$ and an index associated to the corresponding variable). Gates (i.e. Boolean operations like and/or/not) are simulated by processes running on nodes. For each gate, we have nodes that receive the inputs, evaluate the gate, and broadcast their output to the other nodes. A special node intercepts the $true$ message corresponding to the output of the whole circuit and moves in an acceptance state. Regardless the type of communication topology, number of nodes simulating each gate, and possible delays, coverability of the acceptance state corresponds to satisfiability of the circuit G w.r.t. the given assignment. □

5.3 FIFO Mailboxes

In this section we move to ABN with perfect FIFO buffers as communication media. In this context we instantiate the mailbox structure $FIFO$ as follows.

Definition 14. *The set of mailbox contents $\mathcal{M}$ is defined as Σ^*. Furthermore, the mailbox operations are defined as:*

- $add(a, m) = m \cdot a$ *(concatenation of a and m);*
- $del?(a, m) = true$ *iff* $m = a \cdot m'$*;*
- $del(a, m)$ *is the string m' whenever $m = a \cdot m'$, undefined otherwise;*
- *finally, $[] \in \mathcal{M}$ is the empty string ϵ.*

Our main result is that coverability becomes undecidable with this type of mailboxes, Theorem 14. The proof is based on a reduction from termination of counter machines. The correctnesss of the encoding requires a number of intermediate properties that allow us to formally define a representation of the current value of counters via the control flow of messages in a network.

The rationale behind the reduction of the halting problem of two-counter machines to coverability is as follows. We first use an election protocol that assigns fixed roles (controller/slave) to a pair of adjacent nodes. Since the initial configuration is not fixed a priori our election protocol does not forbid the election of multiple pairs of controller/slave nodes, but we only require that at least one pair is elected in order to succeed. The controller/slave nodes set up their mailboxes in order to use them as overlapping circular queues. Messages represent the current value (in unary) of the counters. The simulation is guided by the controller. The slave forwards all received messages back to the controller. As an example, to check that x_1 is zero, the controller reads all messages in the mailbox and checks that in between two successive reads of the marker for x_1 there are no units. We use interference to denote an unwanted message occurring in the mailbox of a controller/slave node. Since the network topology is not fixed a priori, a key point of the whole construction is the capability of controlling interferences with other nodes, e.g., avoiding the adjacency between multiple controllers and slaves. For this purpose, we use special control messages to coordinate the different phases and exploit the FIFO mailboxes in order to enforce the simulation to get into a deadlock state whenever the same control message is received more than once.

5.3.1 Formal Definition and Properties

Formally, let $\mathcal{M} = \langle Loc, Inst, \ell_0 \rangle$ be a two-counter machine. The encoding of $\mathcal{M}$ is defined via an ABN protocol $\mathcal{P}_\mathcal{M} = \langle Q, \Sigma, R, q_0 \rangle$, where each location $\ell \in Loc$ corresponds to a state $\mathcal{P}(l) \in Q$, and each instruction $r \in Inst$ corresponds to a set of auxiliary states and rules. The i-th counter is represented by the FIFO mailboxes of two collaborating processes that forward each other the units u_i and a distinguished token t_i which marks the beginning of the circular queue. The protocol is split in three phases: election, initialization and simulation. The alphabet Σ is partitioned in Σ_e, the messages exchanged during the election, and Σ_s, the messages used for the simulation. We include a special *hello* message in Σ_s. This message is used in the simulation of subtractions and zero tests (done via non atomic steps) in order to check the auxiliary node used to maintain the circular queue is always active during a simulation execution.

Election Since all processes start the protocol in the same initial state, the first thing is to distinguish their roles and make sure that all required communication links are present. This is the purpose of the election phase, during which processes try to build pairs of communicating nodes. An active process may be either a controller or a slave. The duty of the controller is to orchestrate the whole simulation; the purpose of the slave is to bounce back every simulation message received from the controller. We do not care of how pairs will be able to proceed to the next phase, as long as their minimum connectivity requirements are fulfilled: each of them will try to independently carry on its own simulation, or deadlock due to interferences.
Figure 5.5 shows the election protocol. At first every node non-deterministically chooses an

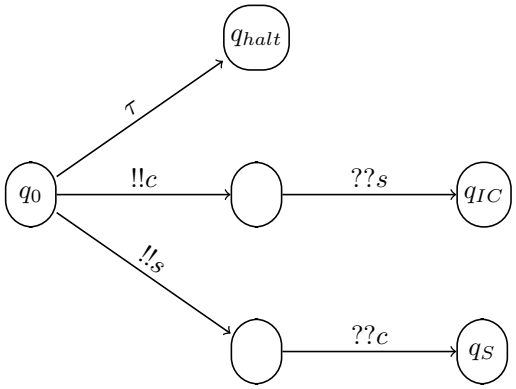


Figure 5.5: Distributed election protocol

active role and announces it to its neighbours ($!!c$ or $!!s$) or shuts down itself by going to q_{halt}. After choosing a role, we run a subprotocol that ensures the existence of the interconnection between controller and slave: a controller needs to receive the announce from a slave, and vice versa. The slave reaches the state q_S that starts its main loop, while the controller goes to an intermediate initialization state q_{IC}. At the end of the election protocol we have the following properties.

Lemma 18. *If a node is in state q_{IC} [resp. q_S], then at least one of its neighbours is already in state q_S [resp. q_{IC}] or it can possibly move only to q_S [resp. q_{IC}].*

Proof. In order for a node n to reach state q_{IC}, n has to read message s from the buffer, but this can happen only if there exists a neighbour m of n that sent s. If that is the case, then m will either move to state q_S by consuming the message c previously sent by n or it will idle forever without taking part in the simulation (e.g. because the visible message in its mailbox is s and not c). For a node in state q_S we can apply a symmetric reasoning. □

Notice that if two neighbours are respectively in state q_{IC} and q_S, there is no guarantee that their mailboxes will be empty. E.g. a node in state q_{IC} may contain a message c sent by another controller. In the rest of the simulation we will enforce a sanity check on the mailboxes in order to ensure that spurious control messages block the execution. This idea is formalized in Lemma 19, but we need to introduce some definitions first. A node in any local state reached after q_S or q_{IC} is said to be *in simulation*. A node *produces* [resp. *consumes*] a (non-empty) string $w \in \Sigma^*$ between q and q' if the corresponding automaton, starting from the local state q, executes a sequence of transitions leading to q' while broadcasting [resp. receiving] the messages in w in the same order. We are now ready to state the lemma.

Lemma 19. *Any string w consumed by a node n in simulation belongs to Σ_s^*. Furthermore, there is a single neighbour m which is in simulation and produces every w consumed by n.*

Proof. We enforce this property by construction. After successfully completing an election (i.e. reaching q_S or q_{IC}) n begins the simulation, during which it only sends and reads messages from Σ_s, and it has no transitions from states used during the simulation back to states used for the election. Thanks to this, the first property holds. For what concerns the second property, we observe that any node in simulation has been able to consume just one Σ_e-message, during the election, and that the strings it can produce are words $e' \cdot w'$ with $e' \in \Sigma_e$ and $w' \in \Sigma_s^*$. Thanks to Lemma 18 and to the fact that w is not empty, at least neighbour m in simulation has to exist. Whenever another neighbour m' completes the election and starts the simulation, then n will receive two (interleaved) strings $e_1 \cdot w_1$ and $e_2 \cdot w_2$, with $e_1, e_2 \in \Sigma_e$ and $w_1, w_2 \in \Sigma_s^*$, respectively from m and m'. Therefore, because of the first property, we can conclude that n will not be able to consume any more messages as soon as e_2 becomes visible in its mailbox: any string w consumed by n in simulation must be a substring of w_1. □

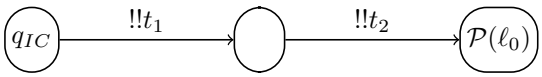


Figure 5.6: Initialization protocol

Initialization Once all nodes have been elected, it is time for the controller to initialize the representation of the counters. This is simply achieved by creating two tokens t_1 and t_2 to be used as markers for the circular queues representing the counters (Figure 5.6). At this point the controller is able to move to the encoding of the initial location of $\mathcal{M}$, $\mathcal{P}(\ell_0)$.

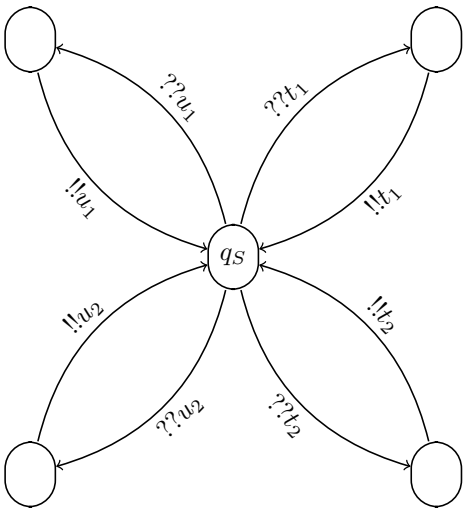


Figure 5.7: Main loop of the slave node q_S

Slave nodes When everything is ready, the nodes proceed to the simulation phase. The slave node starts in q_s, a sort of idle state which manages the forwarding of messages of the counters (Figure 5.7). By applying Lemma 19 we know that each slave node from q_s to q_s consumes a string of messages which is always produced by the same controller node, thus it will never forward any third-party messages.

We are ready now to discuss the simulation of each operation.

Increment In order to increment the j-th counter the controller needs to put one more unit u_j in the circular queue, so a simple $!!u_j$ is enough.

Subtraction Subtraction is achieved by removing a u_j token from the loop (Figure 5.8); if x_j is already at 0, the controller will loop forever (receiving and sending the tokens) without being able to proceed.

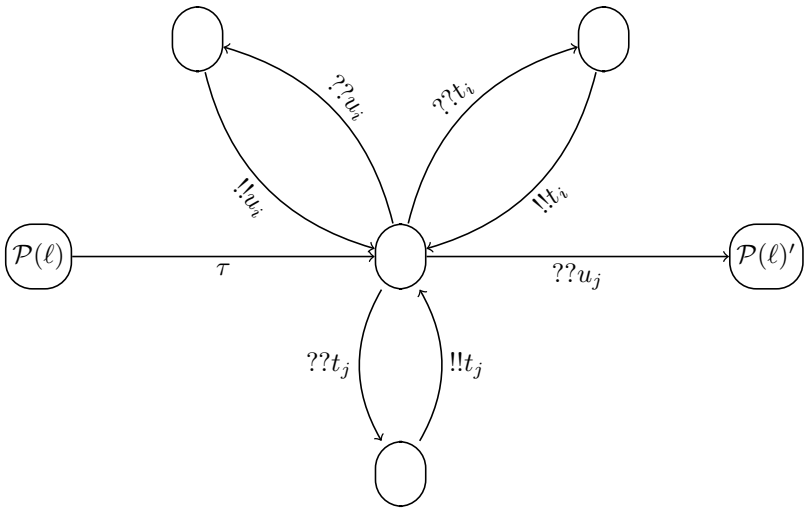


Figure 5.8: Subtraction from counter j, with $i \neq j$

Testing for zero Testing for zero x_j can be done by checking for a sequence of two messages t_j in a row – or going to deadlock in any other case. Of course both operations must carefully keep forwarding messages for the other counter x_i (Figure 5.9).

Subtraction and zero testing are not atomic, and to be sure that the slave was alive during the whole execution the reduction applies the greeting protocol shown in Figure 5.10 after the initialization and after each operation, so that for each instruction of the form $(\ell, op, \ell') \in Inst$ the system may reach the $\mathcal{P}(\ell')$ state only after exchanging hello messages; in case of interferences

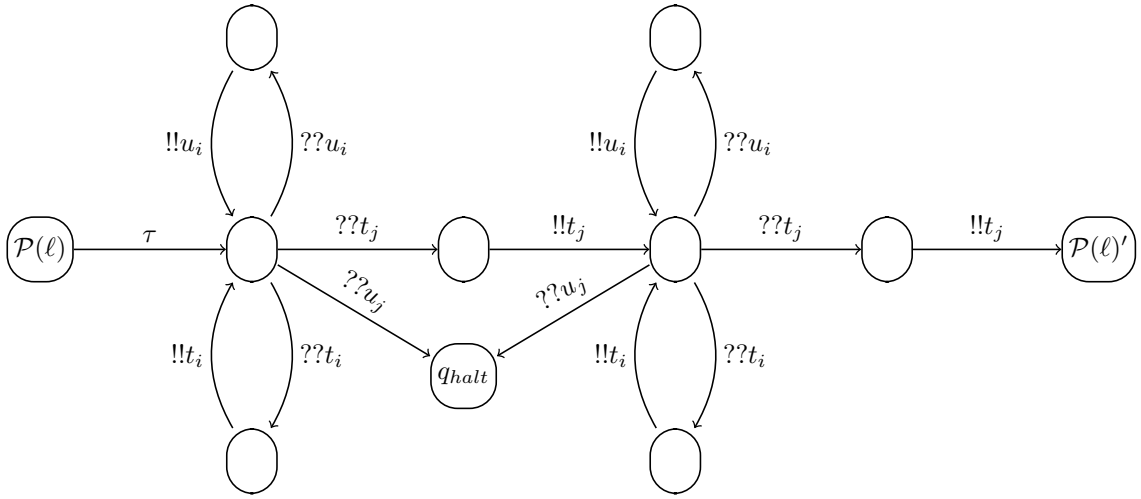


Figure 5.9: Testing for zero counter j, with $i \neq j$

deadlocks are propagated to the controller before completing the simulation of the transition.

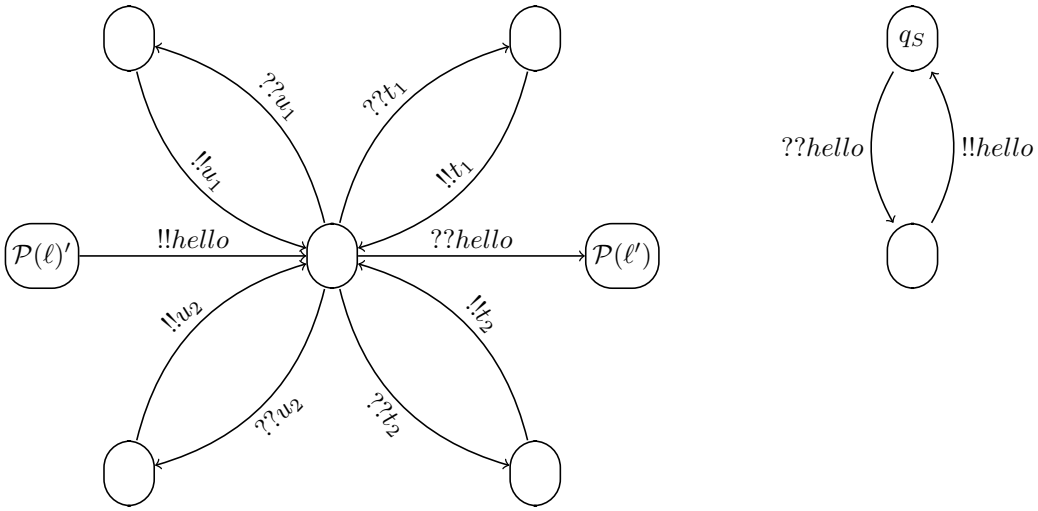


Figure 5.10: Greeting protocol

Furthermore, the greeting protocol allows us to extract an exact representation of a counter to reason about, as stated in the following lemma.

Lemma 20. *For each instruction $(\ell, op, \ell') \in Inst$, the string consumed (and produced) by a controller node between $\mathcal{P}(\ell)'$ and $\mathcal{P}(\ell')$ is a representation of the current state of the two counters (the value of the counter i is equal to the number of occurrences of units u_i).*

Proof. The only difference between the string consumed and the one produced between $\mathcal{P}(\ell)'$

and $\mathcal{P}(\ell')$ is in the position of the unique $hello$ message, which is situated at the beginning of the produced string and at the end of the consumed one. Apart from that, the rest of the two strings will be the same, because the controller broadcasts each message immediately after receiving it in the same order. Since the slave also forwards messages as they appear in his mailbox, a run of the greeting protocol does not change the representation of the counters and does not add new messages into the circular queue ($hello$ is first produced and then consumed by the controller itself).

Thanks to Lemma 19 we know that the controller always exchanges messages with the same slave, and because of the introduction in the circular queue of a unique $hello$ token it can stop consuming messages as soon as it receives it again, when he read each message in the queues exactly once. Therefore, if w is the string consumed by the controller between $\mathcal{P}(\ell)'$ and $\mathcal{P}(\ell')$, then the number of u_1 messages contained in w is the current value of the counter x_1, and the same applies for u_2 and x_2. □

We are finally ready to prove the main undecidability result of this section.

Theorem 14. *$COV(FIFO)$ and $COV_{fc}(FIFO)$ are undecidable.*

Proof. We show by induction on the number of instructions $k \in \mathbb{N}$ in the execution of a two-counter machine that for each $(\ell, op, \ell') \in Inst$ the simulation either leads the system to a deadlock state or it moves the controller to $\mathcal{P}(\ell')$ while maintaining a consistent representation of the counters w.r.t. the corresponding state of the two-counter machine. For the base case $k = 0$, after the initialization we reach $\mathcal{P}(\ell_0)$ only if the controller node is paired with a single slave (Lemma 19) and the first execution of the greeting protocol succeeded, exposing a configuration of the circular queue in which both counters have value 0 (the controller only produced one t_1 and one t_2 message so far, and no units). Assume now that we simulated $i > 0$ instructions, and we are going to simulate the $(k + 1)$-th one, $(\ell, op, \ell') \in Inst$. For the inductive hypothesis, we know that we reached $\mathcal{P}(\ell)$ if and only if the last execution of the greeting protocol showed consistent values for x_1 and x_2. There are three cases to consider, depending on the nature of the instruction. An increment $x_i{+}{+}$ is encoded through a single broadcast of a new unit, therefore if the previous greeting showed c_i units u_i and d_j units u_j then the one leading the controller from $\mathcal{P}(\ell)'$ to $\mathcal{P}(\ell')$ will expose $c_i + 1$ units u_i and d_j units u_j for $i \neq j$. A decrement $x_i - -$ works by forwarding to the slave every message in $\{t_i, t_j, u_j\}$ and by consuming a single u_i message, leading again to a consistent state of the representation of the counters in the greeting after the decrement. Finally, in case of a test $x_i == 0$ the controller is able to reach the next greeting only by producing the same string consumed from $\mathcal{P}(\ell)$ to $\mathcal{P}(\ell)'$: by construction, the only cases which may lose some messages in the circular queue are those leading to the q_{halt} state, which is a deadlock.

The reduction is still valid for fully connected topologies. □

5.4 Lossy FIFO Mailboxes

We now consider coverability for ABNs in which mailboxes are lossy FIFO channels, i.e., channels in which messages may non-deterministically be lost. Given a protocol $\mathcal{P}$, a configuration γ of $\mathcal{T}^{\mathcal{K}}(\mathcal{P}, LFIFO)$ is a multiset of pairs $\langle q, m\rangle$ where $q \in Q$ and $m \in \Sigma^*$. To model non-deterministic loss of messages, we modify the operational semantics by introducing lossy steps.

We first need to define the ordering $\preccurlyeq$ between configurations.

Definition 15. *For $\gamma = \langle V, V \times V, L\rangle$ and $\gamma' = \langle V', V' \times V', L'\rangle$ $\gamma \preccurlyeq \gamma'$ iff there exists an injection $h : V \to V'$ s.t. $L_s(v) = L_s(h(v))$ and $L_b(v) \prec L_b(h(v))$ for each $v \in V$, where $\prec$ denotes the subword relation, namely, for $w, w' \in \Sigma^*$, $w \prec w'$ iff there exists an injective and strictly monotone mapping $h : |w| \to |w'|$ s.t. $w_i = w'_{h(i)}$ for $i : 1, \ldots, |w|$, where v_i denotes the i-th symbol in the word v.*

Intuitively, $\gamma \preccurlyeq \gamma'$ means that γ is obtained from γ' by removing nodes (and all corresponding edges) and messages from the buffers.

Definition 16. *We modify the transition relation $\Rightarrow$ to include lossy steps before and after each transition in the original system as follows: $\gamma \longmapsto \gamma'$ iff there exists η and ν s.t. $\eta \preccurlyeq \gamma$, $\eta \Rightarrow \nu$, and $\gamma' \preccurlyeq \nu$.*

The ordering $\preccurlyeq$ is a simulation relation and is also a well-quasi ordering. These two properties pave the way for a possible application of the theory of well-structured transition systems to solve coverability [DT12], but that would not give a proper upper bound to complexity. Instead, in the rest of the section, we use a reduction to RBN-coverability in order to obtain better complexity results.

Let $\mathcal{P} = \langle Q, \Sigma, R, q_0\rangle$ be a protocol and let $\gamma = \langle V, E, L\rangle$ and $\gamma' = \langle V, E, L'\rangle$ be two configurations from the transition system $\mathcal{T}(\mathcal{P}, LFIFO)$ such that $\gamma \Rightarrow \gamma'$ by applying some rule $r \in R$ to a node $v \in V$. We first show that $\preccurlyeq$ is a simulation relation.

Lemma 21. *Given a configuration $\delta = \langle V, V \times V, K\rangle$ s.t. $\gamma \preccurlyeq \delta$ we can apply r at node v in order to reach a configuration $\delta' = \langle V, V \times V, K'\rangle$ such that $\gamma' \preccurlyeq \delta'$.*

Proof. The proof is by cases on the type of r. Specifically, for all $u \in V$ we have that

- if $r = (L_s(v), \tau, L'_s(v))$ then, since the semantics applies lossy steps, $L'_b(v) \prec L_b(v)$ for every $u \in V$ (messages can get lost).
- if $r = (L_s(v), !!a, L'_s(v))$ then $L'_b(v) \prec K'_b(v)$ and $K'_b(u) \prec add(a, K_b(u))$ for every $u \in V \setminus \{v\}$.

- if $r = (L_s(v), ??a, L'_s(v))$ then $del?(a, K_b(v))$ holds because $L_b(v) \prec K_b(v)$ and by hypothesis $del?(a, L_b(v))$ is satisfied, $L'_b(v) \prec del(a, K_b(v))$, and $L'_b(u) \prec K'_b(u)$ for every remaining $u \in V \setminus \{v\}$.

In all three cases by applying the same transition to δ and assuming that no message is lost in δ we obtain a configuration δ' such that $\gamma' \preccurlyeq \delta'$. □

As for unordered mailbox we show that we can focus our attention on fully connected topologies, only.

Lemma 22. *There exists an execution of $\mathcal{P}$ that satisfies $COV(LFIFO)$ if and only if there is one of $\mathcal{P}$ satisfying $COV_{fc}(LFIFO)$.*

Proof. Given an execution $\gamma_0\gamma_1 \dots \gamma_k$ in $\mathcal{T}(\mathcal{P}, LFIFO)$ where $q \in L_s(\gamma_k)$ and $\gamma_0 = \langle V, E, L^0 \rangle$, we can now show by induction that we can build another one $\delta_0\delta_1 \dots \delta_k$ in $\mathcal{T}^{\mathcal{K}}(\mathcal{P}, LFIFO)$ by starting from $\delta_0 = \langle V, V \times V, L^0 \rangle$ and replicating each rule application as shown in the previous scheme.

- The base case is immediate since initial configurations are graphs in which nodes have the same label, i.e., we can always find a sufficiently large fully connected initial configuration that contains any initial configuration (of arbitrary topology).
- For the inductive step, let $\gamma_0\gamma_1 \dots \gamma_k\gamma_{k+1}$ be an execution in $\mathcal{T}(\mathcal{P}, LFIFO)$. From the inductive hypothesis, we know that there exists $\delta_0\delta_1 \dots \delta_k$ with $\gamma_i \preccurlyeq \delta_i$ for $i : 1, \dots, k$. To conclude the proof, we can now apply lemma 21 to extend the property to $k + 1$ steps by choosing an adequate (according to the Lemma 21) successor configuration δ_{k+1} of δ_k.

The thesis then follows by observing that, because of the construction, the set of control states in δ_i is the same as those in γ_i for $0 \leq i \leq k$ and in particular $q \in K_s(\gamma_k)$. □

We are now at the most tricky part of the proof that consists in proving that $COV_{fc}(LFIFO)$ can be reduced to CRP. The coverability problem for lossy FIFO mailboxes has a property in common with the one for bags, that is in both cases processes are able to ignore incoming messages indefinitely; this is achieved by either leaving the message in the multiset or by deleting it from the lossy FIFO queue. Let $\mathcal{P}$ be an ABN protocol, and let $\mathcal{P}'$ be the corresponding RBN protocol derived as in Section 5.2.

Lemma 23. *There exists an execution of $\mathcal{P}'$ that satisfies CRP if and only if there is one of $\mathcal{P}$ satisfying $COV_{fc}(LFIFO)$.*

Proof. Let $\theta_0 \to \theta_1 \to \cdots \to \theta_m$ be an execution of $\mathcal{P}'$ such that $\theta_i = \langle V, E'_i, L'_i \rangle$ for all $i \in \{1, \ldots, m\}$. We now proceed by induction on m to prove that there exists an execution $\gamma_0 \Rightarrow \cdots \Rightarrow \gamma_n = \langle V, E, L^n \rangle$ such that $\forall v \in V$, $L^n_s(q) = L'_m(q)$ and such that $L^i_b(v)$ has always at most one message.

- For $m = 0$, the configuration $\gamma_0 \in \Gamma_0$ with vertices V trivially satisfies the thesis.
- For $m \geq 1$, from the inductive hypothesis we know that given an execution $\theta_0 \to \cdots \to \theta_{m-1}$ of $\mathcal{P}'$ we have another one $\gamma_0 \Rightarrow \cdots \Rightarrow \gamma_k$ of $\mathcal{P}$ such that $L^k_s(q) = L'_{m-1}(q)$ for all $v \in V$ and such that all queues have at most one message. We want to prove that, given $\theta_m \in \Theta$ such that $\theta_{m-1} \to \theta_m$, there exists $\gamma = \langle V, E, L \rangle \in \Gamma$ such that $\gamma_k \Rightarrow^* \gamma$ and $L_s(\gamma) = L'(\theta_m)$. In the case when $\theta_{m-1} \to \theta_m$ because of a graph reconfiguration, then the thesis follows immediately by considering that $\gamma_k \Rightarrow^* \gamma$. In all other cases there exists a rule $r = \langle q, !!a, q' \rangle \in R$ such that $\theta_{m-1} \to \theta_m$ thanks to a broadcast by a node $v \in V$. We build the execution $\gamma_k \Rightarrow \gamma_{k+1} \Rightarrow \cdots \Rightarrow \gamma_{k+1+r}$ where $\gamma_k \Rightarrow \gamma_{k+1}$ corresponds to an application of rule r to the same node v that also occurs in γ_k. Furthermore, we apply a lossy step to ensure that message a is always visible in the mailboxes of nodes that are connected to v in θ_{m-1} and to simply remove the message a from the mailboxes of nodes that are not connected to v in θ_{m-1}. As a consequence, $L_b(u)$ contains now the message a for each node $u \sim v$ in γ_{k+1}.

  The remaining r transitions are now reception steps, one for each of the r neighbours $u \in V$ of v that are enabled to react to the message sent by v. Finally, by choosing for the r steps the same exact reception rules that fired during the synchronous broadcast step (they are necessarily enabled because the local state of individual processes by the inductive hypothesis matches the one in θ_{m-1}), we obtain that for $\gamma = \gamma_{k+1+r} = \langle V, E, L \rangle$ we have that $L_s(v) = L'_m(v)$ and $L_b(v)$ has at most one message for all $v \in V$.

The other side of the implication can be proved in exactly the same way as for the case of unordered mailboxes, because when we assume an ABN execution the semantics of the mailbox is already implicitly considered and all preconditions to reception steps are satisfied (it would not be an execution otherwise). □

We can now prove the following result.

Theorem 15. *$COV_{fc}(LFIFO)$ is* PTIME*-complete.*

Proof. Membership to PTIME follows from the reduction to CRP for RBNs. Hardness follows again from a reduction of CVP to COV for ABN lossy FIFO queues. The encoding protocol is the same as for unordered mailboxes. □

5.5 ABN with Emptiness Test

In this section we enrich the ABN model with a new type of transitions in order to enable nodes to test whether their mailbox is empty. We call the resulting model ABN_ϵ. The set Act of action labels is extended to include ϵ, i.e., $Act = \{\tau, \epsilon\} \cup \{!!a, ??a \mid a \in \Sigma\}$. The transition system associated to an ABN_ϵ is extended in order to take ϵ into account. Formally, in addition to the other local, broadcast and reception steps, given two configurations $\gamma = \langle V, E, L\rangle$ and $\gamma' = \langle V, E, L'\rangle$, $\gamma \Rightarrow \gamma'$ also holds if the following condition is met:

Emptiness test There exists a $v \in V$ such that $(L_s(v), \epsilon, L'_s(v)) \in R$, $L_b(v) = L'_b(v) = []$, and $L(u) = L'(u)$ for each $u \in V \setminus \{v\}$.

The only difference w.r.t. the semantics of τ-transitions consists in the $L_b(v) = []$ condition, that ensures that ϵ-transitions only fire when the mailbox is empty.

The introduction of ϵ-transitions affects the different instances of the coverability problem in different ways. The simplest case is for $COV_{fc}(FIFO)$ and $COV(FIFO)$, which of course are still undecidable: the possibility to test the emptiness of the mailbox does not have any effect on the reduction from two-counter machines. The reduction from $COV_{fc}(LFIFO)$ to CRP of Lemma 23 has to be modified in order to consider also ϵ-transitions. Given two configurations $\gamma, \gamma' \in \Gamma$ such that $\gamma \preccurlyeq \gamma'$ (see Section 5.4 for the definition of the $\preccurlyeq$ ordering), if ϵ is enabled in γ then it can be fired starting from γ' too, through a preliminary lossy step that empties the relevant mailbox. This means that ϵ-transitions are almost the same as internal transitions in case of $LFIFO$ mailboxes. Therefore, given a protocol $\mathcal{P} = \langle Q, \Sigma, R, q_0\rangle$ and a target state $q \in Q$, we derive an RBN protocol $\mathcal{P}' = \langle Q, \Sigma, R', \{q_0\}\rangle$ where R' is the set of rules R where all occurrences of ϵ have been replaced by τ, and then we solve CRP for the target state q. Thanks to the previously mentioned property of ϵ-transitions, one could adapt easily enough the proof of Lemma 23 to this case. From these observations we can therefore derive that both $COV(LFIFO)$ and $COV_{fc}(LFIFO)$ are decidable even with ϵ-transitions.

We incur in a completely different case when considering bags: as it can be shown, the extended semantics traces indeed a sharp boundary between decidability and undecidability. Without the emptiness test, both reachability problems $COV(Bag)$ and $COV_{fc}(Bag)$ are decidable; we prove that the operator ϵ introduced with the extended model is sufficient to make them undecidable. The proof proceeds by building a reduction from the control state reachability problem for two-counter machines to $COV(Bag)$. The reduction encodes a counter machine $\mathcal{M}$ with an ABN protocol $\mathcal{P} = \langle Q, \Sigma, R, q_0\rangle$ where, like before, each location $\ell \in Loc$ and each instruction $i \in Inst$ corresponds respectively to a state $\mathcal{P}(\ell) \in Q$ and to a set of intermediate states and rules. The protocol is split in two phases. In the first phase processes follow a distributed election protocol to identify who takes care of which role and who is excluded from the simulation. The second phase is the simulation of $\mathcal{M}$. The alphabet is partitioned in two sets, $\Sigma_e = \{c, s_1, s_2\}$ for the election and $\Sigma_s = \{u_i, tz_i, sub_i \mid i \in \{1, 2\}\}$ for the simulation. Since we do not make

any particular assumption on the connectivity graph, the proof works for both $COV_{fc}(Bag)$ and $COV(Bag)$.

Election A simulation must be carried out by three nodes: a controller and two slaves, one per counter. Figure 5.11 shows the protocol used to choose such roles. We say that a node is *in simulation* if it reaches (at least once) $\mathcal{P}(\ell_0)$, q_{S_1}, or q_{S_2}. The election guarantees minimal

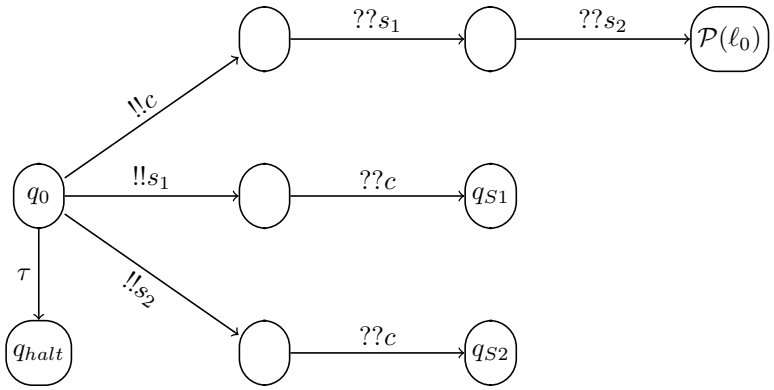


Figure 5.11: $COV(Bag)$: Election protocol

connectivity requirements, as stated in the following Lemma.

Lemma 24. *If a node is in state $\mathcal{P}(\ell_0)$, then at least two of its neighbours are respectively in state q_{S_1} and q_{S_2} or they can possibly move only to those states. If a node is in state q_{S_1} or q_{S_2}, then at least one of its neighbours is already in state $\mathcal{P}(\ell_0)$ or it can possibly move only to $\mathcal{P}(\ell_0)$.*

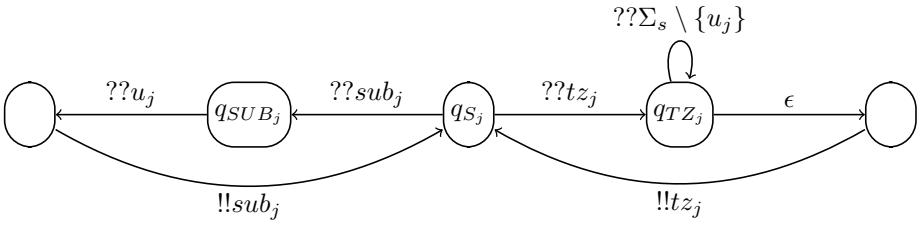


Figure 5.12: $COV(Bag)$: Slave process

Simulation Each slave S_j keeps in its mailbox a number of u_j messages equal to the current value of counter x_j. The controller sends messages sub_j or tz_j to give orders depending on the instruction (ℓ, op, ℓ') that is going to be simulated by the system and waits for the slave which manages the involved counter to react accordingly (see Figure 5.12). Once the slave is done, the same control message is sent back to the controller as acknowledgement and the controller

is able to proceed. When A is a set we write $??A$ to mean that for every $a \in A$ the protocol has a reception rule $??a$ with the same endpoints. Again, the increment can be done directly by the controller with a single broadcast $!!u_j$. In order to be able to prove the correctness of the reduction, we first state some properties of the simulation phase.

Lemma 25. *Any $m \in \Sigma_e$ received by a node in simulation will persist in its mailbox forever. Such a node is said to be* in interference*.*

Proof. By construction, for all $m \in \Sigma_e$, there are no receptions of m starting from any state which may be reached by simulating nodes. □

Lemma 26. *At any time, the value of the counter i is equal to the number of occurrences of units u_i in the mailbox of the corresponding slave, provided that no simulating node is in interference. We say then that the counters are* valid*.*

Proof. As in the FIFO case, every node by construction broadcasts a single message in Σ_e before everything else. Since each node is only able to consume – during election – a number of Σ_e-messages equal to to the number of required neighbours, it follows that we should expect interferences as soon as a simulating node receives a message $m \in \Sigma_e$. Finally, we conclude that invalid counters will never become valid again thanks to Lemma 25. □

We remark that the notion of validity of the counters does not have anything to do with the compliance of their values w.r.t. the ones of the two-counter machine being simulated. Moreover, since the simulation may proceed even with invalid counters, the reduction does not compute reachability of the encoding $\mathcal{P}(\ell_f)$ of the target state ℓ_f, but instead it checks for the reachability of a fresh state q_{target} added according to Figure 5.13. This is needed in order to ensure the correctness of the simulation. It is straightforward to check that the instructions added to $\mathcal{M}$ do

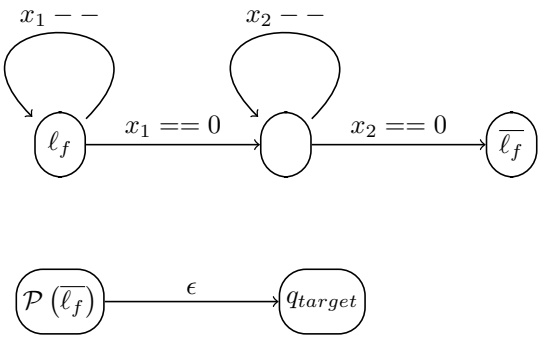


Figure 5.13: $COV(Bag)$: Interference detection

not have any impact on the reachability of the target location, as they just decrement down to zero both counters before reaching the destination. We are now ready to prove that the reduction is

indeed a correct simulation of the given two-counter machine. The rationale is the following. We demonstrate by induction on the number of simulated instructions that for any number of steps, either the counters will be valid and consistent w.r.t. the corresponding state of the two-counter machine or they will be (and remain) invalid. Given this property, we can exploit Lemma 25 in order to show that the added, final transitions from Figure 5.13 ensure that the controller will deadlock before reaching the target state when the counters are invalid.

Theorem 16. *$COV(Bag)$ [$COV_{fc}(Bag)$] is undecidable in ABN$_\epsilon$.*

Proof. Let us consider an execution simulating $k \in \mathbb{N}$ instructions of the two-counter machine. At $k = 0$, either the counters are invalid or they are valid and they are trivially both equal to 0. At the $(k+1)$-th step, we simulated the previous k instructions either by ending up with invalid counters, or with valid counters and c_i units u_i and d_j units u_j. In the first case, after the step $k+1$ the counters will still be invalid (Lemma 26). For the second case, let $(\ell, op, \ell') \in Inst$ be the $(k+1)$-th instruction that is going to be simulated. If interferences occurs during its execution then the counters will be invalid. Otherwise, there are three cases to consider. When op is an increment x_i++, the whole instruction is simulated by a single broadcast that will add a unit u_j to the mailbox of slave S_i, therefore the new values for the counters will be respectively $c_i + 1$ and d_j. When op is a decrement x_i--, the slave S_i will be forced to read a unit u_i from its mailbox (while S_j will not react), in such a way that only the value of counter x_i will change (into $c_i - 1$) after the step, provided that $c_i > 0$. When op is a test $x_i == 0$, the slave S_j will not react and S_i will delete only useless messages in $\Sigma_s \setminus \{u_i\}$ in order to complete the operation, therefore both of the counters after the execution of the step will still have the same values as before. In any case, we can say that after the execution of a new step, either the counters will be valid and consistent w.r.t. the corresponding state of the two-counter machine or they will be invalid.

Because of the instructions and transitions introduced in Figure 5.13, when the controller is in state $\mathcal{P}(\ell_f)$ both slave nodes have to execute an ϵ-transition (during the latest tests for zero) in order for the controller to be able to try to execute his latest ϵ-transition leading to the target state q_{target}. Thanks to the previously proved property on executions we can conclude that either the system will deadlock just before reaching q_{target} when the counters are not valid (thanks to Lemma 25, ϵ-transitions cannot be executed), or will reach q_{target} with valid and consistent counters. □

5.6 Conclusions

In this chapter we have proposed a mathematical model of distributed systems with asynchronous communication. The model is based on an automata-based specification of the behaviour of

	$COV_{fc}(\mathbb{M})$		$COV(\mathbb{M})$	
	ABN	**ABN$_\epsilon$**	**ABN**	**ABN$_\epsilon$**
LFIFO	PTIME	PTIME	PTIME	PTIME
Bag	PTIME	undec.	PTIME	undec.
FIFO	undec.	undec.	undec.	undec.

Table 5.1: Summary of the results for Asynchronous Broadcast Networks

individual nodes, and on an operational semantics based on a global transition systems. For this model, we have studied parameterized verification (i.e. existence of an initial configuration that can lead to a configuration with an error state) varying the policy used to handle the mailbox and the class of topologies considered in the communication layer (fully connected vs arbitrary graphs).

To our knowledge the presented results (summarized in Table 5.5) are original w.r.t. previous results on verification of models with broadcast communication in which communication is typically synchronous as in [EN98, EFM99, DSZ10]. Furthermore, more in general, the resulting expressiveness hierarchy has distinguished features with respect to those associated to other (fragments of) infinite-state models, e.g. extensions of Petri nets [ADB11]. More specifically, differently from models like Broadcast Protocols [EN98], computability and complexity of coverability for ABN presents a huge gap: from PTIME to undecidability. This is due to the fact that, in all positive cases, the model does not provide any means to keep track of occurrences of individual processes. Only reachable control states seem relevant to the analysis. This result was not obvious at all at the beginning of our investigations. Indeed the richness of the communication layer (with data structures like bags and queues local to each node) seemed an obstacle to any feasible analysis (our first procedures have non-elementary complexity [DT12]). The turning point for us was the connection with the reconfiguration semantics of AHN [DSZ10], an ideal representation of dynamic graph reconfigurations that can also be viewed as a low level semantics for different types of communication failures [DSZ12]. In this chapter we have extended the role of reconfiguration and showed that it can be used as a low level model for encoding the behavior of complex communication patterns like those obtained equipping nodes with local mailboxes.

Chapter 6

Broadcast Networks of Register Automata

We introduce a formal model of data-sensitive distributed protocols, called Broadcast Networks of Register Automata (BNRA), aimed at modelling both the local knowledge of distributed nodes as well as their interaction via broadcast communication. A network is modelled 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 [KF94]. Each register assumes values taken from the set of natural numbers. Node interaction is specified via broadcast communication, well-suited to model scenarios in which individual nodes have partial information about the network topology. Messages are allowed to carry data, that can be assigned to or tested against the local registers of receivers. Dynamic updates of the current configuration are modelled via non-deterministic reconfigurations of the underlying connectivity graph. A node may disconnect from its neighbours and connect to other ones at any time of the execution. This behaviour models in a natural way unexpected power-off and dynamic movement of devices. 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.

In the chapter, which is an extended version with proofs of the paper [DST13b], we focus our attention on the decidability and complexity of parameterized verification, 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 (e.g. a loop in the information contained in the routing tables). The considered class of verification problems is parametric in four dimensions, namely, the number of nodes, the topology of the initial configuration to be discovered, and the amount of data contained in local registers and exchanged messages.

Acknowledgements The author would like to thank Prof. Giorgio Delzanno and Prof. Arnaud Sangnier, co-authors of the article [DST13b] from which this chapter is taken.

Related Work

Our formal model of topology-sensitive broadcast communication with data naturally extends those obtained in [DSZ10, DSZ11, DSTZ12b]. Formal models of broadcast networks date back to CBS [Pra95], extended in several ways (time, asynchrony, etc) in successive works. Automated verification methods have been tested on protocols for Ad Hoc Networks with a fixed number of processes in [FvHM07, SRS09, FvGH$^+$12a]. Verification of broadcast protocols in fully connected networks in which nodes and messages range over a finite set of states has been considered, e.g., in [EN98, GS92, APR$^+$01]. Via an adequate counting abstraction, the problem can be reformulated in terms of Petri nets with transfer arcs [EFM99, Del03]. The non-elementary complexity of coverability in this class of nets is proved in [Sch10]. Symbolic backward exploration procedures for network protocols specified in graph rewriting have been presented in [JK08] (termination guaranteed for ring topologies) and [SWJ08] (approximations without termination guarantees). Decidability issues for broadcast communication in fully connected networks have been studied in [EFM99]. Verification of unreliable communicating FIFO systems has been studied in [AJ96]. Coverability problems for broadcast communication in fully connected networks with data is investigated in [ADB11, LNO$^+$08, DRV13].

6.1 Broadcast Networks of Register Automata

6.1.1 Syntax and semantics

We model a distributed network using a graph in which the behaviour 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 register p_i 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:

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

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 behaviour of an individual node.

Definition 17. *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 chapter 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. 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 labelling 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 neighbours. 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 $L_M(v)[i] \neq L_M(v)[j]$. In an initial configuration, all the registers of the nodes

contain different values. 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. Specifically, 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 $Reach^{fc}(\mathcal{P}) = Reach^b(\mathcal{P}) \cap \Gamma^{fc}$.

6.1.2 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 w.r.t. 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 $f : Z \mapsto V$. The satisfaction relation $\models$ is parameterized by a valuation and is defined inductively as follows:

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

We say that a configuration γ satisfies a reachability query φ, denoted by $\gamma \models \varphi$ if and only if there exists a valuation f such that $\gamma \models_f \varphi$. Furthermore we assume that our queries do not contain contradictions w.r.t. $=$ and $\neq$. We now define the parameterized verification problem, i.e., finding an initial configuration that leads to a configuration containing a sub-configuration that matches the query.

Definition 18. *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).

6.1.3 An Example: Route Discovery Protocol

Consider the problem of building a route from nodes of type $sender$ to nodes of type $dest$. We assume that nodes have 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 6.1. Initially nodes have type $sender$, $idle$, and $dest$. Request messages like $rreq$ are used to query adjacent

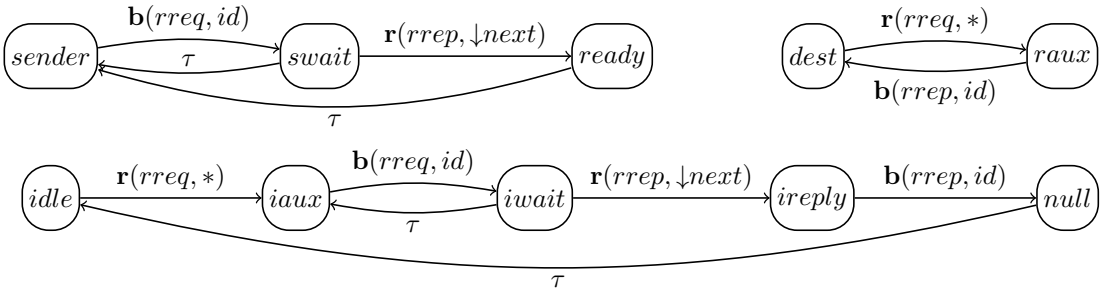


Figure 6.1: Route discovery example

nodes in search for a valid neighbour. Back edges are used to restart the protocol in case of loss of intermediate messages or no reply at all.

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 $ready(z_1) \wedge ready(z_2) \wedge M_{id}(z_1) = M_{next}(z_2) \wedge M_{next}(z_1) = M_{id}(z_2)$.

6.2 Reconfiguration in Arbitrary Graphs

6.2.1 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. 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. 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'. For $q_a, q_b, q_c \in Q$, a list (linked via x) is a set of nodes $\{v_1, \cdots, v_k\}$ such that v_1 has label q_a, v_k has label q_c, v_i has label q_b for $i \in [2..k-1]$, and v_j is the unique node in V that points to v_{j+1} via x and has label in $\{q_a, q_b\}$ for $j \in [0..k-1]$. In other words q_a and q_c are sentinels for a list made of q_b elements. A backward list is defined as before but with reversed pointers, i.e., v_{j+1} points to v_j.

Definition 19. *A protocol $\mathcal{P} = \langle Q, R, q_0 \rangle$ with $\{q_a, q_b, q_c\} \subseteq Q$ is a list-builder for q_a, q_b, and q_c on $x \in [1..r]$ if, for any γ such that $\gamma \in Reach(\mathcal{P})$, if a node v in γ has label q_a, then v is the first node of a list linked via x.*

A backward list-builder is defined in a similar way for backward lists.

Lemma 27. *For $r \geq 2$ and $f \geq 1$, $Cov(r, f)$ is undecidable if there exists a list-builder (r, f)-protocol on $x \in [1..r]$ that can generate lists of any finite length.*

The proof exploits 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. The details of the protocol that extends a list-builder are given in Section 6.4. A similar result can be stated for backward list-builders.

The previous lemma tells us that to prove undecidability of coverability we just have to exhibit a list-builder protocol. In the case of $Cov(2, 2)$, we apply Lemma 27, by showing that protocol $\mathcal{P}_{lb}$ of Figure 6.2 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 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.

Theorem 17. *$Cov(2, 2)$ is undecidable even when restricting one register to be read-only.*

Proof. Let us consider protocol $\mathcal{P}_{lb}$ of Figure 6.2. 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

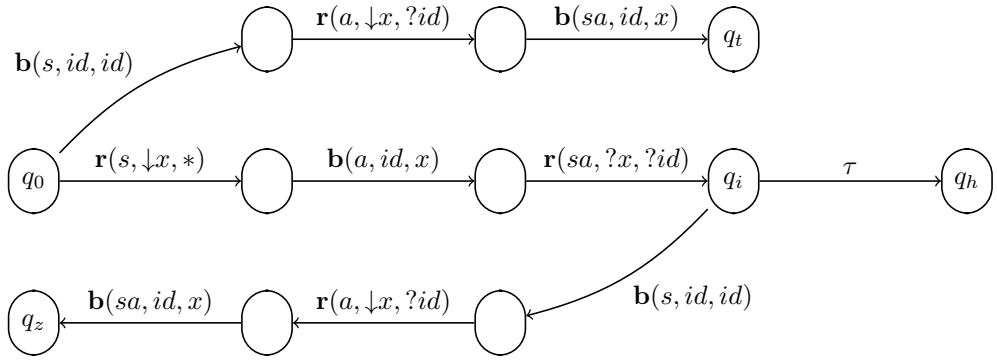


Figure 6.2: $\mathcal{P}_{lb}$: backward list-builder for q_h, q_z, q_t, and Γ_0 on x

current neighbours 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 towards 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 19, is indeed a backward list builder for q_h, q_z, q_t and Γ_0 on $x \in [1..r]$. By applying Lemma 27, we can finally conclude that $Cov(2, 2)$ is undecidable. □

6.2.2 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.

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

Proposition 9. $Cov(*, 1)$ *is* PSPACE*-hard.*

Proof. 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 behaviour 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 6.3. 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 $t^\bullet = \{p_{j_1}, \ldots, p_{j_l}\}$ (with

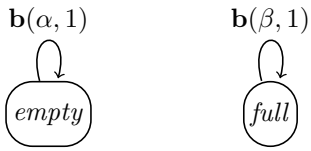


Figure 6.3: Encoding the nodes of type *full* and *empty*

$i_1, \ldots, i_k, j_1, \ldots, j_l \in [1..r]$), we will have in $\mathcal{P}$ the transitions depicted in Figure 6.4. 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 6.5 shows how the simulation begin from the initial state q_0, first nodes can go in states *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 $\gamma \models end$. □

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.

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 labelling function (as for configurations, we will denote λ_Q

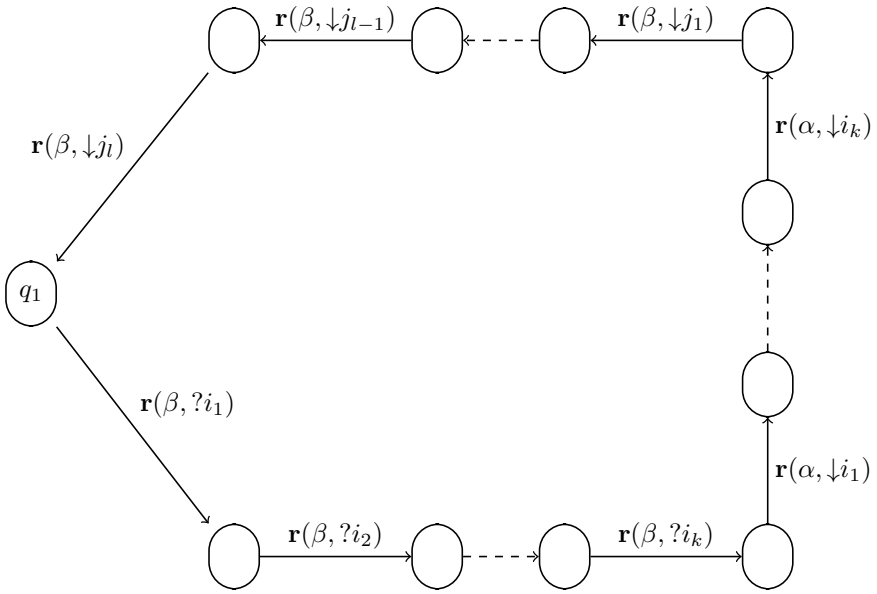


Figure 6.4: Encoding transition t with ${}^\bullet t = \{p_{i_1}, \ldots, p_{i_k}\}$ and $t^\bullet = \{p_{j_1}, \ldots, p_{j_l}\}$

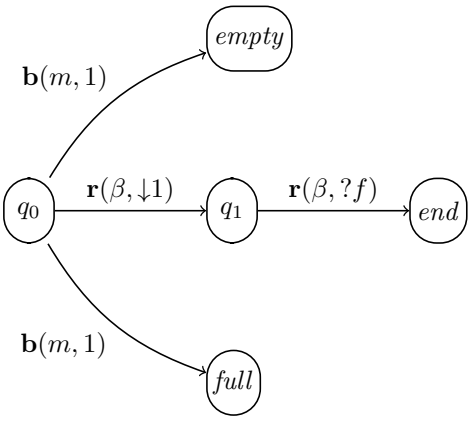


Figure 6.5: Initialization of the simulation and ending of the simulation of the 1-safe net

[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 $\mathcal{P}$. 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. $\lambda_M(v)[j] = \lambda_M(v)[j']$, i.e. a node with the same value in two distinct registers, then there exists a self loop $(w, j, j', w) \in \delta$.

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 $\mathcal{P}$. In order to perform a symbolic reachability on symbolic configurations, we use a operator $\texttt{POST}_{\mathcal{P}}$ that, by working on a graph θ simulates the effect of the application of a broadcast rule on its instances $[\![\theta]\!]$. The formal definition of the $\texttt{POST}_{\mathcal{P}}$ operation, along with missing proofs, is given in [DST13a]. 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 6.6, 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). 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

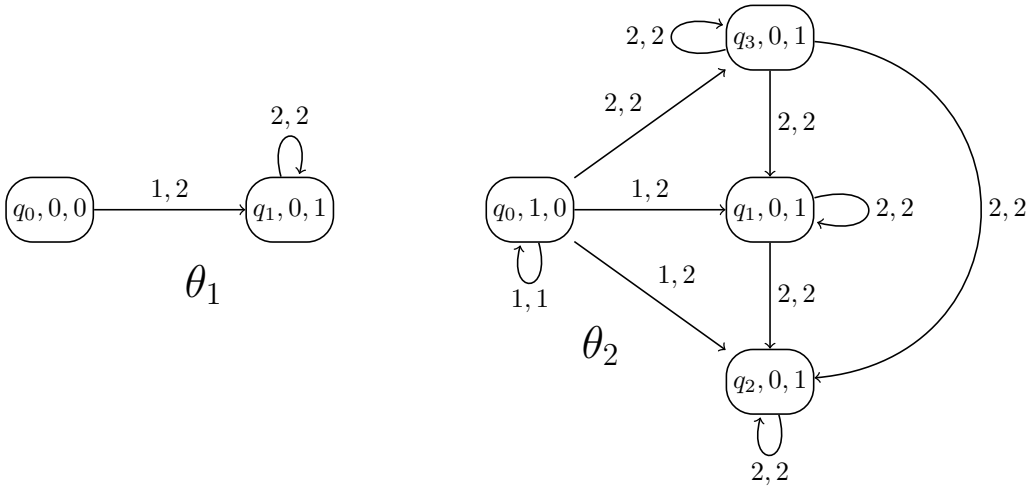


Figure 6.6: Example of computations of symbolic post

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 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 28. *Given a symbolic configuration θ and a reachability query φ, we have $\theta \models \varphi$ if and only there exists $\gamma \in [\![\theta]\!]$ such that $\gamma \models \varphi$.*

Before giving the properties of the $\mathtt{POST}_{\mathcal{P}}$ 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. One can easily prove the following result.

Lemma 29. *(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 $j \geq i$, $\theta_j = \theta_i$.*

Furthermore, 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 $\mathtt{post}_{\mathcal{P}}(S) = \{\gamma' \in \Gamma \mid \exists \gamma \in \Gamma \text{ s.t. } \gamma \Rightarrow \gamma'\}$ and $\mathtt{post}^*_{\mathcal{P}}$ is the reflexive and transitive closure of $\mathtt{post}_{\mathcal{P}}$ (and $\mathtt{POST}^*_{\mathcal{P}}$ the reflexive and transitive closure of $\mathtt{POST}_{\mathcal{P}}$). Note that since symbolic configuration generate a single node for each label, repeated application of $\mathtt{POST}_{\mathcal{P}}$ are ensured to terminate. We can now give the properties of the $\mathtt{POST}_{\mathcal{P}}$ operator.

Lemma 30. *Let θ be a symbolic configuration of the protocol $\mathcal{P}$. Then we have $\theta \sqsubseteq \mathtt{POST}_{\mathcal{P}}(\theta)$ and for all reachability query φ, there exists $\gamma \in \mathtt{post}^*_{\mathcal{P}}([\![\theta]\!])$ such that $\gamma \models \varphi$ iff $\mathtt{POST}^*_{\mathcal{P}}(\theta) \models \varphi$.*

We have consequently an algorithm to solve whether there exists $\gamma \in Reach(\mathcal{P}) = \mathtt{post}_{\mathcal{P}}(\Gamma_0)$. 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 thanks to Lemma 29 and thanks to the first point of the previous lemma. Note that each 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 $\mathtt{POST}^*_{\mathcal{P}}(\theta_0)$. 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 $Vars(\varphi)$ to the nodes of $\mathtt{POST}^*_{\mathcal{P}}(\theta_0)$). Using Lemma 28, 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 9, we can deduce the exact complexity of coverability for protocols using a single field in their messages.

Theorem 18. *$Cov(*, 1)$ is* PSPACE*-complete.*

6.3 Fully Connected Topologies and No Reconfiguration

6.3.1 Undecidability of $Cov^{fc}(2, 1)$

We now move to coverability in fully connected topologies. In contrast with the results obtained without identifiers in [DSZ10] it turns out that, without reconfiguration, coverability is undecidable already in the case of nodes with two registers and one payload field. Following the same line as in Lemma 27, to prove the result it is enough to define a (forward) list-builder protocol. We refer to Lemma 27* as the variation of Lemma 27 obtained considering the relation $\Rightarrow_b$ (see Section 6.4). The protocol 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 27*, the following theorem then holds.

Theorem 19. *$Cov^{fc}(2, 1)$ is undecidable even when one register is read-only.*

Proof. We now show that the protocol $\mathcal{P}^{fc}_{lb}$ in Figure 6.7 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

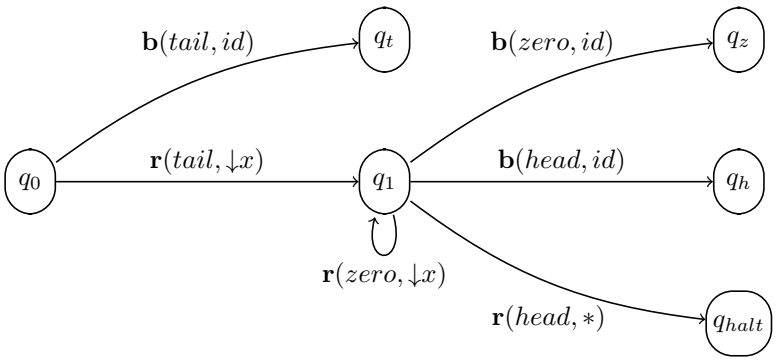


Figure 6.7: $\mathcal{P}^{fc}_{lb}$: list builder for $\langle q_h, q_z, q_t\rangle$ and fully connected configurations on x

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 $\mathbf{b}(tail, id)$. Since the network is fully

connected, every process has to react to the 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 21 (an exteneded version of Definition 19, see Section 6.4), $\mathcal{P}_{lb}^{fc}$ 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 27*.

□

6.3.2 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 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.

Let $\mathcal{P} = \langle Q, R, q_0 \rangle$ be a $(1,1)$-protocol. 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 20. *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 10. *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. □

Proposition 11. *Let $pre_{\mathcal{P}}(S) = \{\gamma \mid \gamma \Rightarrow_b \gamma', \gamma' \in S\}$. There exists an algorithm $\mathtt{PRE}_{\mathcal{P}}$ taking in input $I \subseteq \Xi$ and returning a set $I' \subseteq \Xi$ s.t. $[\![I']\!] = pre_{\mathcal{P}}([\![I]\!])$.*

The formal definition of the predecessor operator is given in [DST13a], together with an example. Following [AJ01], 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 PRE until a fixpoint is reached. The termination test is defined using $\prec$. The wqo $\prec$ ensures termination of the computation [ACJT96]. The following theorem then holds.

Theorem 20. *$Cov^{fc}(1,1)$ is decidable.*

Proof. We show how to apply the symbolic predecessor computation based on PRE. 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 partitionings of Z. Each partitioning $P_i = \{X^i_1, \ldots, X^i_{u_i}\}$ 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 PRE, say $\mathtt{PRE}^*(I)$. To check if the resulting set of symbolic configurations contains an initial state, we need to search for a finite basis $\langle V, L\rangle$ 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 $[[q_0], \ldots, [q_0]] \in \mathtt{PRE}^*(I)$. □

We consider now the complexity. We observe that, without registers and fields our model boils down to the AHNs of [DSZ10]. For fully connected topologies, AHN can simulate reset nets as shown in [DSZ11]. Following from the complexity of coverability in reset nets [Sch10], we have the the following theoretical lower bound.

Corollary 2. *$Cov^{fc}(0,0)$ and $Cov^{fc}(1,1)$ are non elementary.*

6.4 Proof of Lemma 27*

In the following section we prove Lemma 27*, a more general version of Lemma 27. Before introducing its formal statement, we first have to refine and introduce some terminology. We remind that we assume protocols to have at least a read-only register $id \in [1..r]$. A $\langle Q_a, Q_b, Q_c \rangle$-*list* (linked via x), or simply *list*, with pairwise disjoint sets $Q_a, Q_b, Q_c \subset Q$, is the natural extension of the concept of a list for q_a, q_b, and q_c to sets of states. 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 refine the definition of list builder.

Definition 21. *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 $L_Q(v) = q_a$, v is the first [resp. last] node of a $\langle q_a, q_b, q_c \rangle$-list [resp. $\langle q_c, q_b, q_a \rangle$-list] linked via x. Furthermore, if $\rightsquigarrow$ is $\Rightarrow_b$, then $v' \sim v''$ for all successive nodes v' and v'' in the list.*

There are three changes in the definition above. First, the transition relation for which the protocol is guaranteed to work is made explicit. Second, there are two different conditions for forward and backward builders, that is to have v respectively as the first node of a $\langle q_a, q_b, q_c \rangle$-list or as the last node of a $\langle q_c, q_b, q_a \rangle$-list. Third, the added requirement, for $\Rightarrow_b^*$ only, to have $v' \sim v''$ for each pair of successive nodes in the list. This is needed to ensure communication back and forth through the list, because in this case we cannot rely any more on reconfigurations.

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 $\mathbf{r}(m, ?x)$.

We are now ready to state and prove the Lemma.

Lemma 31 (Lemma 27*). *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.

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 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 19, 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.

It's worth to note 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 6.8 shows how increments $(\ell, c{+}{+}, \ell') \in Inst$ are encoded. The head node, in state $\mathcal{P}(\ell)$, sends an increment order inc_c and waits for an acknowledgement reply inc_c^r before moving to the encoding of the next state, $\mathcal{P}(\ell')$. The message is propagated through the list, until 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_c^r can be sent only by the tail node q_t. The nodes in state q_c representing units of the currently tested counter do not propagate the message, therefore the message tz_c travels through the whole list and reaches the tail if and only if there are no q_c nodes, i.e. when $c = 0$.

When considering reconfigurations, i.e. when the transition relation is $\Rightarrow$, all of the previous assumptions still hold, except for the fact that, when sending a message from a node in the list to its successor, we no longer know if the two of them are neighbours. Otherwise said, with $\Rightarrow$

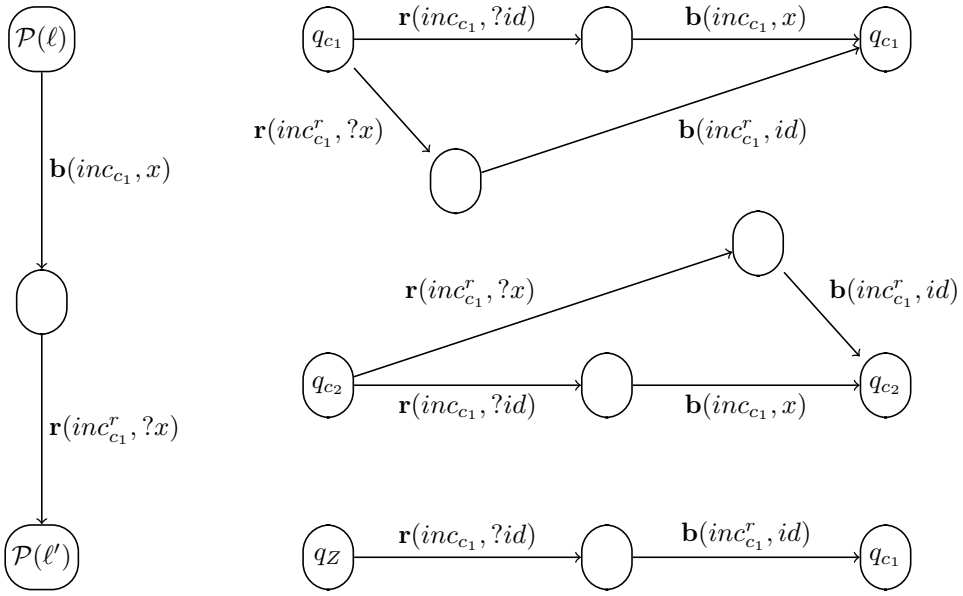


Figure 6.8: Increment of counter c_1

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 halting 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 coverability problem is satisfied by the existence of an execution that leads to the target state. We can therefore conclude that Lemma 27* (Lemma 31) holds. □

6.5 Conclusions

In this chapter we investigated decidability and complexity for verification of a formal model of distributed computation based on register automata communicating via broadcast messages with data. The results are summarized in Table 6.5.

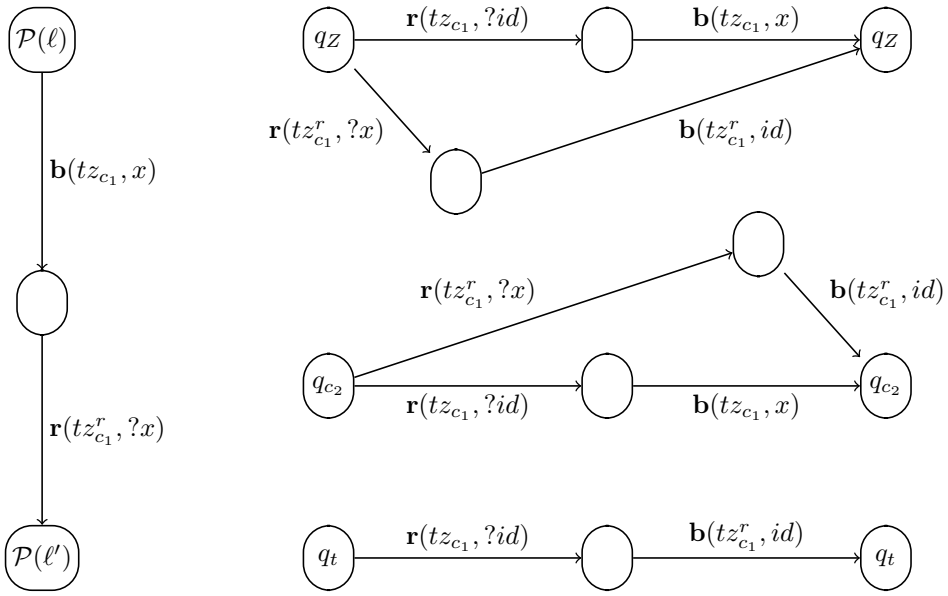


Figure 6.9: Testing for zero counter c_1

	Protocol		
Problem	r	f	**Complexity**
$Cov(r, f)$	0	0	PTIME
	$r \geq 1$	1	PSPACE
	$r \geq 2$	$f \geq 2$	undec.
$Cov^{fc}(r, f)$	0	0	non-elementary
	1	1	non-elementary
	$r \geq 2$	$f \geq 1$	undec.
$Cov^{b}(r, f)$	$r \geq 0$	$f \geq 0$	undec.

Table 6.1: Summary of the results for Broadcast Networks of Register Automata

Chapter 7

Model Checking with Groove

We now move back to models with finitely many processes and consider an orthogonal problem to those studied in the parameterized cases, i.e., the state explosion problem that arises during the exploration of the corresponding reachability graph. Automated validation of distributed algorithms like routing and consensus protocols is a challenging task for state-of-the-art model checkers [FvGH+12a, FvHM07, SRS09, SWJ08, KVW12, DT13a, BSS10]. These protocols indeed depend on the current network topology, and furthermore they operate under asynchronous communication assumptions, features both concurring to state space explosion.

Following a well established connection between graph grammars and verification (see e.g. [JK08, BDK+12, Kön04, KR06, SWJ08]), we propose to attack this problem by applying Graph Transformation Systems (GTS) as an automated validation method for distributed algorithms. Graph grammars provide a declarative language to specify updates (of structure and labels) in a graph-based representation of a dynamic system. Apart from providing an alternative description of the problem that may in itself be interesting, the formalisation of states as graphs rather than the more conventional vectors of data values opens the way towards additional techniques for state space reduction such as graph isomorphism. We consider here two case studies.

The first one is Gafni-Bertsekas algorithm [GB81, DT13b], an instance of the more general class of Link Reversal Routing (LRR) algorithms used for route maintenance in Mobile Ad Hoc Networks [WW11]. In this version of the protocol heights (tuples of integers) are associated to individual nodes. The lexicographic order of heights specifies the direction of an overlay network that points towards a fixed destination node. A virtual link between two nodes is well-defined only if they are connected in the underlying communication topology, made of undirected links. The main feature of the protocol is that link reversal steps maintain the virtual network in form of a DAG.

Our declarative specification is based on graph transformation rules with symbolic conditions on node attributes, negative application conditions and node quantification, as provided in the

GROOVE framework [GdMR+12]. GROOVE has been originally developed as an automated support for object-oriented program transformations, however its specification language can easily be adapted to model dynamically changing networks [KR06]. We exploit this feature to express link updates that describe unexpected network modifications (link deletion and addition) as well as route maintenance phases. Conditions and updates on data fields are combined with graph production rules in order to reason on a height-based version of the protocol. All specifications are given by exploiting symmetries induced by the use of transformations that work modulo graph isomorphism. To validate our model, we use the GROOVE model checker for a CTL temporal logic on transitions. The model checker operates on the labelled transition system (LTS) induced by a set of graph productions. CTL propositions are defined on top of transition names. This way it is possible to state properties on graph patterns that define the enabling of a rule. This is particularly useful when considering enabling conditions defined via regular expressions on edge labels. E.g., they can be used to formally specify reachability of partitioned configurations. Strategies for generating the LTS can be defined by associating priorities to production rules and by using control programs. The combination of priorities, rich patterns expressing bad configurations, and temporal properties allows to specify all the correctness requirements of the Gafni-Bertsekas algorithm.

We report on experimental results obtained with the model checker by varying both the size of the initial configuration and the number of dynamic link modifications. Starting from fully connected topologies and admitting dynamic modifications, we manage to handle networks with up to six nodes.

The second case study we consider is Paxos [Lam98, Lam01], a distributed consensus protocol. The consensus problem requires agreement among a number of agents for a single data value. Some of the agents may fail, so consensus protocols must be fault tolerant. Initially, each agent proposes a value to all other ones. Agents can then exchange information. A correct protocol must ensure that when a node takes the final choice, the chosen value is the same for all correct agents. It is assumed here that messages can be delayed arbitrarily. A subset of processes can crash anytime and restore their state (keeping their local information) after an arbitrary delay. Fisher, Lynch and Patterson have shown that, under the these assumptions, solving consensus is impossible [FLP85]. In [Lam98] Lamport proposed a (possibly non-terminating) algorithm, called Paxos, addressing this problem. Paxos is based on the metaphor of a part-time parliament, in which part-time legislators need to keep consistent records of their passing laws. Because the description proved hard to understand, Lamport later provided a simpler description of the protocol in [Lam01]. This is the version on which we base the models in this chapter. Salient features of our specification are:

- We use node occurrences as abstractions of proposed values and process identifiers. This causes symmetries to show up as graph isomorphism between states.
- We use a Linda-like model for asynchronous communication, in which message broad-

casts are represented by special nodes linked to their senders, without associated buffers or channels. This avoids state differences due to irrelevant message orderings.

- Our rules encapsulate a lot of functionality within a single (atomic) transformation step, and thus avoid intermediate states during evaluation.

We compare the resulting models computed by GROOVE to those generated by SPIN from a specification of the same protocol in PROMELA. The comparison shows that the choices listed above manage to keep the graph-based state space size to a fraction of that of a more traditional vector-based specification, enabling the analysis of larger problem instances despite the inherent complexity of graph transformation. The same state space explosion can also be observed in the same GROOVE model, by disabling graph isomorphism reduction for configurations. As with the vector-based specification, this is enough to significantly reduce the size of configurations we are able to verify.

Our method can be seen as an attempt of combining declarative reasoning and efficient search methods for this class of protocols. Furthermore, it represents an alternative to standard model checking frameworks based on (unstructured) symbolic representations, e.g., BDDs.

Acknowledgements The author would like to thank Prof. Arend Rensink and Prof. Giorgio Delzanno as co-authors of the two papers from which this chapter is taken. The first paper on the topic [DT13b] is written with Prof. Delzanno and improved thanks to suggestions on the use of GROOVE by Prof. Rensink. The second paper [DRT14], written together with both of them, will be presented in April at the GRAPHITE'14 international workshop.

Related Work

Specification and verification of routing protocols dates back to the seminal work on HOL/SPIN for AODV in [BOG02]. More recently, model checking tools (e.g. SPIN [FvHM07] and Uppaal [FvGH$^+$12a]) and constraint-based engines [SRS08, SRS09, SRS10] have been applied to verification of Ad Hoc and wireless protocols. In these approaches executions of a fixed number of agents are explored via enumerative or symbolic methods [FvHM07, FvGH$^+$12a], or by generating positive/negative constraints on links in a lazy manner [SRS08, SRS09, SRS10]. Parameterized verification of models of broadcast communication has been studied from a theoretical point of view in [DSZ10, DSZ11, DSTZ12b], where decidability and complexity frontiers have been given for problems like control state reachability (reachability of a state in which a node has a certain state). Applications of the invisible invariant method to distributed algorithms has been considered in [BPZ06]. Cut-off properties for link reversal routing has been considered in [FW12]. The manifesto on automated verification of distributed algorithms is presented in [KVW12].

The use of graph transformation systems for automated validation of dynamic systems has been proposed in [KR06] using GROOVE. In [SWJ08] symbolic backward exploration with subgraph relation as termination test has been applied to parameterized verification of routing protocols like LUNAR. In the same line of thoughts, in [JK08] the graph minor ordering is used as termination test for symbolic analysis of ring protocols. Decidability of reachability problems for Graph Transformation Systems are studied in [BDK$^+$12]. A comparison of the performance of GROOVE, used as a verification tool for dynamical systems, and SPIN is considered in [KR06].

7.1 Graph Grammars as Executable Specifications

GROOVE is based on a graph representation of system states and on a representation of state updates via graph transformations. Graph production rules specify both matching patterns and negative application conditions (NAC). Graph matching is used to select the pattern in the host graph that has to be rewritten into a new graph (obtained by deleting/adding/merging nodes and edges). A NAC specifies a sub-pattern that must be absent in the host graph in order for the rule to be applicable (e.g. they specify global conditions).

7.1.1 Graph Transformation Systems (GTS)

To define a GTS, we follow the style of [ER97, Hec06]. A graph $G = \langle N, E, L\rangle$ consists of a finite set N of nodes, a finite set $E \subseteq N \times N$ of edges, and a labelling function L of nodes and edges. We use $\mathcal{G}$ to denote the set of all graphs, ranged over by $G, H, \ldots$. A graph matching $m : \mathcal{G} \to \mathcal{G}$ is a graph morphism that preserves node and edge labels, i.e., for $G = \langle N, E, L\rangle$ and $G' = \langle N', E', L'\rangle$, if $e = \langle n, n'\rangle \in E$, then $e' = \langle m(n), m(n')\rangle \in E'$, $L(n) = L'(m(n))$, $L(n') = L'(m(n'))$, and $L(e) = L'(e')$. A graph transformation rule $p \in R$ specifies how the system evolves when going from one state to another: it is identified by its name ($Np \in \mathcal{N}$, where $\mathcal{N}$ is a global set of rule names) and consists of a left-hand side graph (L_p), a right-hand side graph (R_p), and a set of so-called negative application conditions (NAC$_p$, which are supergraphs of L_p).

Definition 22. *A Graph Production System (GPS) $P = \langle I, R\rangle$ consists of a graph I (the initial state), and a set of graph transformation rules R.*

The application of a graph transformation rule p transforms a graph G, the source graph, into a graph H, the target graph, by looking for an occurrence of L_p in G (specified by a graph matching m that cannot be extended to an occurrence of any graph in NAC$_p$) and then by replacing that occurrence with R_p, resulting in H. Such a rule application is denoted as $G \to_{p,m} H$. Each GPS $P = \langle R, I\rangle$ specifies a (possibly infinite) state space which can be generated by repeatedly applying the graph transformation rules on the states, starting from the initial state I.

Definition 23. *A GTS $T = \langle S, \rightarrow, I \rangle$ generated by $P = \langle R, I \rangle$ consists of a set $S \subseteq \mathcal{G}$ of graphs representing states, an initial state $I \in S$, and a transition relation $\rightarrow \in S \times R \times [\mathcal{G} \rightarrow \mathcal{G}] \times S$, such that $\langle G, p, m, H \rangle \in \rightarrow$ iff there is a rule application $G \rightarrow_{p,m} H'$ with H' isomorphic to H.*

7.1.2 The GROOVE Simulator and Model Checker

GROOVE [GdMR+12] consists of a GUI that allows editing of rules and graphs and animated simulations of a specification. The state space is stored as an LTS. The strategy according to which the state space is explored can be set as a parameter. In the latest versions of GROOVE there are several specification facilities. We will hereafter refer to the latest version to date, version 4.8.6. Node labels can either be node types or flags. A flag is used to model a boolean condition which is true for a node if and only if the flag is present. To specify data fields ranging over basic types like booleans, integers, and strings, we can use node attributes. Attributes are treated as special edges that do not point to a standard node, but to a node that corresponds to a data value. The type of a node specifies the set of allowed flags and edge labels and and the type of its data fields. Operations over data fields are specified as node relations (evaluated automatically in data fields corresponding to the results) or as typed expressions with constructs for both updates (e.g. let) and nested expressions.

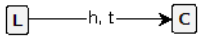
Universal quantification is another interesting feature of the input language. A universally quantified (sub)rule is a rule that is applied to all subgraphs that satisfy the relevant application conditions, rather than just a single one as in the standard case. The use of universally quantified rules allows to naturally define parametric transformations (thus saving space in both the input model and in the state space). Universal quantification can be nested within existential quantifiers to define shared patterns between multiple applications of the same production (e.g. to rewrite all nodes that have a field/data in common with a specific node). The documentation of the feature is not very detailed, so we will use it in a restricted way.

GROOVE provides different means for controlling the rewriting process. The first method is based on priorities. Low-priority rules may only be applied if no higher-priority rule is applicable. Another method is via control programs, which can be used to restrict the system to specific sequences of rule applications. A control program can be viewed as an automata whose language specifies admissible sequences of transition applications.

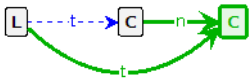
Finally, another interesting feature is the use of regular expressions to define nested conditions on configurations. They extend graph matching with parametric pattern conditions such as the existence of a path between two nodes. For instance, an edge with label $a+$ between two nodes matches any (non null) path in which edges have only labels in a. Such a pattern can then be used to update the host graph.

7.1.3 An Example: Linked List

To illustrate how graph productions are specified in the GROOVE visual language we consider an example in which graphs represent dynamically created linked lists with tail insertion (put) and head removal (get). In the initial configuration we use two nodes as sentinels to denote the empty list. The first node has two forward pointers (h=head, t=tail) both pointing to the second node.

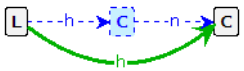


The put operation inserts a new node pointed by the tail pointer. The GROOVE visual language adopts coloured nodes and edges to denote deletion and addition of edges, nodes, and label updates. Indeed, the rule is specified as follows.

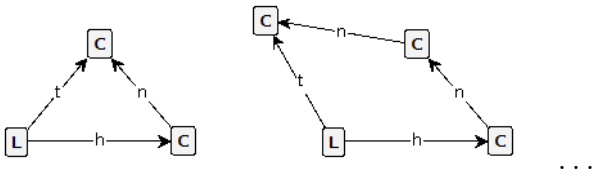


The dashed line denotes the deletion of the old t-edge. A deletion acts both as a guard (the edge being removed has to exist) and as a postcondition (the edge is removed from the graph). The thick lines denote the addition of two new edges. This notation can be expanded into a graph production containing a graph L_p with two nodes L and C connected via a t-edge in the left-hand side, and a graph R_p with three nodes L, C and C, with an n-edge connecting the last two nodes and a t-edge connecting the first and last node. The L_p graph is removed and the nodes and edges of R_p are created at its place. The nodes of R_p are linked to the nodes of L_p via a further graph morphism (usually denoted by using extra numerical labels to put in relation nodes in the left- and right-hand side).

Deletion of a cell is specified via the following rule.



Dashed lines denote removal of old h- and n-edges. The thick line denotes the addition of a new h-edge. Clearly, the two productions assume that in the transformed graph there exists only one h-edge and only one t-edge (this property is invariant under applications of the productions). Starting from the initial configuration, the firing of put produces the following sequence of configurations



The transformation can continue either back to the previous states, or to a list with an additional cell. Therefore, the considered graph production system generates lists of arbitrary length.

GROOVE provides a GUI with a Simulator for the step-by-step visualization of the behaviour of a system, in which it is possible to highlight the matching pattern for a specific rule. Rules are partitioned in accord to the associated priorities. The simulator guides the execution via the corresponding rule ordering.

The built-in verifier generates the LTS of a given graph production system in form of a reachability graph. Based on this representation of the state-space, the model checker supports verification of CTL/LTL specifications that can be defined over graph patterns. This is achieved by using rule names as propositions. Specifically, the left-hand side of a formula (and the corresponding NAC) is used to check for the presence of a given sub-pattern in the current configuration. Consider a rule with name bad, whose left-hand side denotes a bad pattern (e.g. a cycle in a graph). Then, the firing of bad denotes the occurrence of the bad pattern in the reachability graph. Formulas are built over predicates defined over rule names, temporal operators like A, E (for CTL only), F, G, X (for CTL/LTL), and of their Boolean combinations (and/or/negation). A CTL formula like $AG\ !bad$ can then be used to specify the safety property "the bad pattern can never be reached". In our linked list example we could specify bad patterns like self-loops with regular expressions on h- and t-edges (unreachable in our model). If a property does not hold, the model checker returns a counter-example.

7.2 The Gafni-Bertsekas algorithm

We consider here the partial reversal version of the Gafni-Bertsekas algorithm [GB81]. In this setting a node that becomes a sink tries to minimize the number of links to be reversed. The partial reversal method can be implemented using heights. More in detail, every node u has a tuple of values $\langle \alpha_u, \beta_u, id_u \rangle$, where α_u is a non negative integer, β_u is an integer, and id_u is an integer that denotes a unique identifier for the node. Initially, α_u is 0 for every node u. Furthermore, heights are totally ordered using the lexicographic ordering, namely

$$\langle \alpha_u, \beta_u, u \rangle < \langle \alpha_v, \beta_v, v \rangle$$

if and only if $\alpha_u < \alpha_v$ or ($\alpha_u = \alpha_v$ and $\beta_u < \beta_v$) or ($\alpha_u = \alpha_v$ and $\beta_u = \beta_v$ and $id_u < id_v$). The latter condition is used to break the tie whenever the other values cannot be used to order a

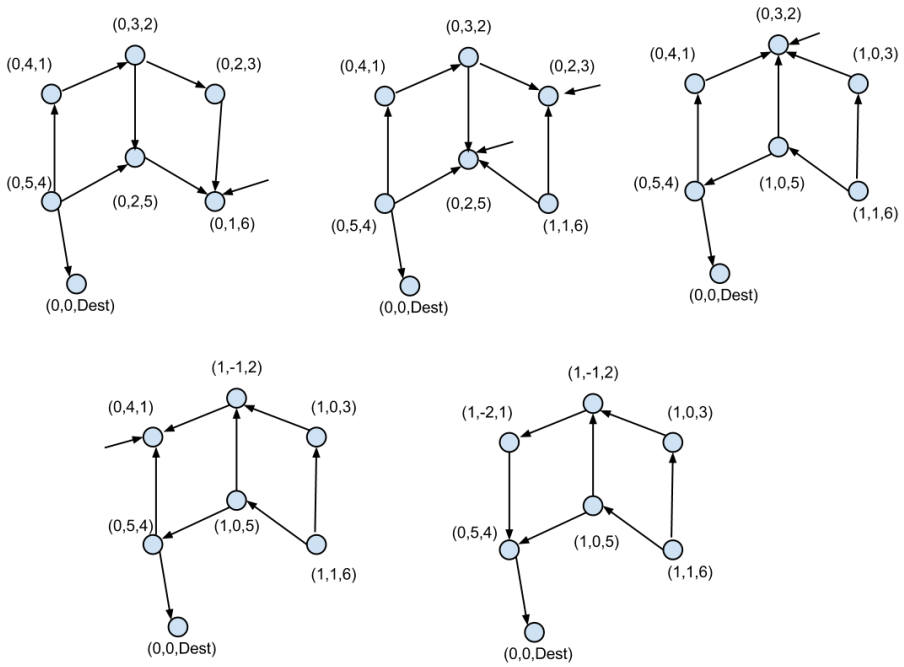


Figure 7.1: Execution of Height-based Reversal.

pair of nodes. For every pair u and v of adjacent nodes, v points to u in the overlay network via a virtual edge if and only if $\langle \alpha_u, \beta_u, u \rangle < \langle \alpha_v, \beta_v, v \rangle$. The destination node is always considered as a global minimum, i.e., its height is $\langle 0, 0, 0 \rangle$ and it has only incoming virtual edges that must never be reversed. Route maintenance is triggered when a node u has no more incoming edges, i.e., the node is a local minimum w.r.t. $<$. Let N_u be the set of neighbours of node u. The node tries to repair the configuration by updating the value of α_u with a value that is larger than the minimum value of the α's for nodes in N_u,

$$\alpha'_u = (min_{v \in N_u} \alpha_v) + 1$$

After the update, all edges directed to nodes with smaller α will be reversed. To minimize the number of reversals for nodes with the same value for α we operate on β. Namely, we set the new value of β_u to be strictly less than the minimum value of the β's for those nodes in N_u with a value for α equal to α'_u, i.e.,

$$\beta'_u = (min_{v \in \{v' \in N_u | \alpha_{v'} = \alpha'_u\}} \beta_v) - 1$$

If the graph is connected, the algorithm is guaranteed to terminate and to produce a new DAG pointing to the destination node (Propositions 1 and 2 in [GB81]). As in the scenario considered in the original algorithm, we assume here that route maintenance is performed after the failure of a single link and terminated before the subsequent link failure (the algorithm is designed for

networks with such a relation between the frequency of the two types of events). We show an example of reversal steps in Figure 7.1.

We remark that the full reversal algorithm can be obtained by ignoring β and changing the updates of α as follows:

$$\alpha'_u = (max_{v \in N_u} \alpha_v) + 1$$

This way all incoming edges of a sink node are reversed into outgoing edges. Only when passing from informal specifications to formal ones, we can uncover details that must be taken into account in a real implementation of the protocol. Since the informal specification of the LRR algorithm is based on graph transformations, it seems a natural case-study for a tool like GROOVE in order to fully exploit symmetries and compactness of graph production rules.

7.3 Gafni-Bertsekas: Formal Specification using GROOVE

In this section we describe a formal specification of the Gafni-Bertsekas algorithm described in Section 7.2 using the GROOVE input language. Only when passing from informal specifications to formal ones, we can uncover details that must be taken into account in a real implementation of the protocol. Since the informal specification of the LRR algorithm is based on graph transformations, it seems a natural case-study for a tool like GROOVE in order to fully exploit symmetries and compactness of graph production rules.

The model has a type system which defines three types of nodes, together with the flags and edges which they support: **Node** is the type for nodes which execute the LRR protocol, **Counter** is used for the unique node keeping a global counter for generating fresh node identifiers, and, finally, **Lock** is the type of a control node used to serialize application of a specific set of rules.

7.3.1 Network Initialization

An initial configuration for the protocol consists of a graph where adj edges represent communication links between adjacent nodes. All nodes are initially labelled with the $init$ flag. In Figure 7.2 we show a initial configuration with four nodes. To model undirected edges we use pairs of directed edges connecting the same nodes. We will discuss in Section 7.4 how to specify rules that can dynamically rearrange the connection topology of the network (add and delete edges). On top of the communication topology, the protocol builds a virtual DAG, that we will represent via $next$ edges, with a single destination node. One of the properties that we will have to ensure is that, after dynamic modifications that do not partition the network and after route maintenance, every node maintains a path of $next$-edges leading to the destination node.

To define an initial consistent DAG w.r.t. $next$-edges, we first select a destination node and then

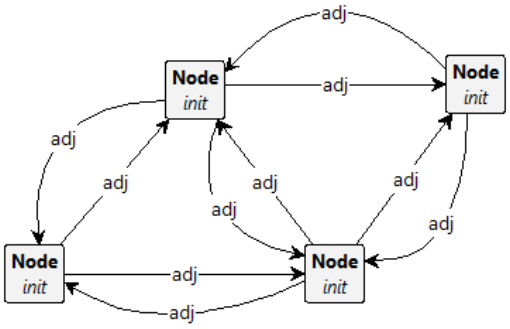


Figure 7.2: Initial configuration with four nodes.

initialize the heights of all other nodes by choosing increasing identifiers. For this purpose, we use the rules in Figure 7.3, 7.4, and 7.5. They are all given the maximum priority level, in order to ensure no other rule will be fired before the initialization of the system is complete. For clarity, we remark that flags and attributes added or removed as a postcondition of a rule are respectively preceded by a plus or a minus sign, whereas nodes [resp. edges] with thick green borders [resp. green lines] are created as a side effect of the rule. Rule INIT-DEST of Figure 7.3 introduces a

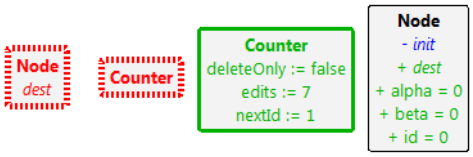


Figure 7.3: INIT-DEST: Non-deterministic choice of destination node and counter initialization.

new **Counter** node with fields $deleteOnly$ and $edits$ used in the validation phase (see Section 7.4), and $nextId$ used for the generation of fresh identifiers. In addition it non-deterministically selects a destination node. INIT-DEST may fire only if the system is still uninitialized, i.e., both the **Counter** and a destination node are missing. This negative conditions (NAC) are expressed through nodes with thick red dashed borders. The rule INIT-LRR (Figure 7.4) fires once for each

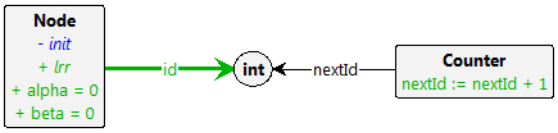


Figure 7.4: INIT-LRR: Initialization of active LRR nodes.

Node still labelled by $init$. It introduces the integer fields $alpha$ and $beta$ initialized to 0, and it assigns a unique identifier id. The flag of the node is changed to lrr to mark it as ready, and the

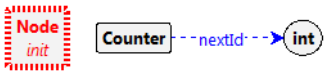


Figure 7.5: INIT-DONE: Removal of $nextId$ field.

$nextId$ field of **Counter** is increased by one. When INIT-LRR is done (i.e. there are no more $init$ nodes), INIT-DONE (Figure 7.5) marks the end of the initialization phase and the beginning of the simulation of the Gafni-Bertsekas LRR protocol.

7.3.2 Virtual DAG of $next$-edges

At first no virtual edges (labelled by $next$) towards the destination exist, as the nodes did not interact with each other, yet. In such a case the preconditions to fire NEW-LINKS (Figure 7.6)

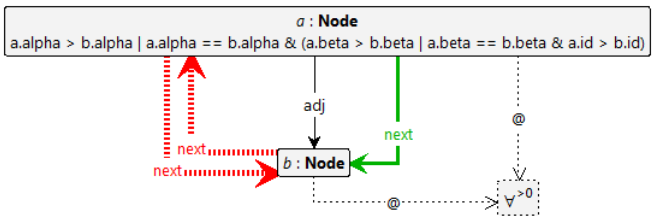


Figure 7.6: NEW-LINKS: Creation of $next$-edges.

are satisfied. NEW-LINKS creates $next$-edges between pairs of adjacent nodes in accord to their relative heights (a $next$-edge goes from higher to lower heights). The special syntax a : **Node** fixes an identifier a for the node of type **Node**. Such identifier can be used in expressions, e.g., $a.alpha$, to concisely access fields of node a. Thick, dashed edges are treated as negative preconditions. The rule selects each pair of adjacent nodes a and b without any $next$ edge connecting them (negative condition specified by a dashed edges with label $next$) and creates an $next$ edge from a to b iff a has an height lexicographically greater than b. The comparison of triples is specified as a $test$ label inside node a. Since nodes a and b are universally quantified, the considered pattern is applied to all matching subgraphs of the host graph at the same time. In other words, with just one firing of the rule every missing next edge is added. When the network is initialized and all of the $next$ edges are ready, a configuration with four nodes may look like that of Figure 7.7. This configuration contains an example of a sink node (with $id = 1$), i.e., a node other than the destination without outgoing $next$ edges. The considered graph is a DAG w.r.t. the $next$ relation. We remark that even though $next$ edges are redundant w.r.t. the information on heights, they are useful for several reasons. Indeed, they represent an abstraction of data maintained in local tables as specified by the protocol, with the purpose of minimizing the number of height comparisons. This is useful also in our model, because by looking at

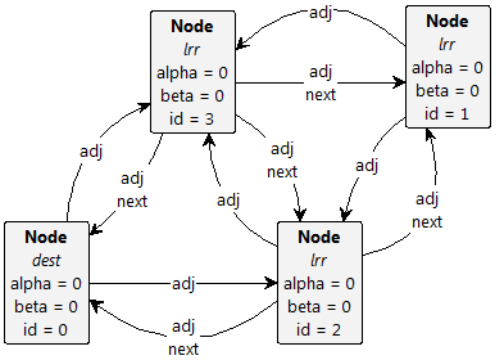


Figure 7.7: Fully-initialized network with four nodes and $next$ edges.

$next$ edges rather than at the heights many rules can be significantly simplified. Furthermore, the heights cannot be exploited when checking properties like, e.g., the existence of a route of arbitrary length to the destination. Instead, the presence of edges encoding the same information enables us to write such properties as regular expressions on paths, like $next+$ (we will discuss this point in Section 7.4).

It is also important to remark that, since the initialization of the heights is non-deterministic, even for a fixed initial topology, the creation of $next$-edges can generate several different DAGs that depend on the order in which identifiers are assigned to nodes. During state-exploration GROOVE applies symmetry reduction to avoid generation of isomorphic (w.r.t. both adj- and $next$-links) graphs.

7.3.3 Sink Detection

Sink nodes (e.g. the node with $id = 1$ in Figure 7.7) trigger the route maintenance phase. The rules in Figures 7.8 and 7.9 have a decreasing priority in order to execute them in the

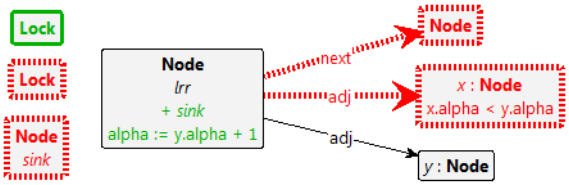


Figure 7.8: SINK-ALPHA: Detection of a sink node and update of $alpha$.

correct order. We also use a **Lock** node in order to ensure that each rule is applied at most once, depending on the cases.

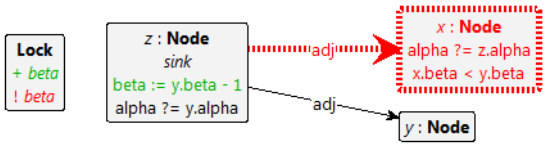


Figure 7.9: SINK-BETA: Update of $beta$.

In a situation such as in Figure 7.7, a new $sink$ can be detected through rule SINK-ALPHA. The preconditions require that **Lock** is not present, there are no $sink$ nodes, and that the node has no outgoing $next$ edges. All these negative conditions are marked with dashed lines. The chosen lrr node n is marked with the $sink$ label, and a fresh **Lock** node is generated to enable SINK-BETA (and forbid other applications of SINK-ALPHA). To update the value of the $alpha$ attribute of n, say $n.alpha$, we first select a neighbour node y with minimum value v for $alpha$ and then we assign $v + 1$ to $n.alpha$. Note that to select the node y with minimum $alpha$, we reason by contraposition and require that no other neighbour x has a value strictly smaller than $y.alpha$ (NAC that combines edges and conditions on fields).

Once $alpha$ is updated, the LRR protocol can proceed with the update of $beta$, which is performed via the rule SINK-BETA of Figure 7.9. SINK-BETA requires the presence of a **Lock** node without the $beta$ flag, which is added as a post-condition in order to let the rule fire at most once. Differently from $alpha$, $beta$ has only to be compared w.r.t. the neighbours sharing the same $alpha$ as the $sink$. Since it is not always the case that there are such neighbours, this rule may be skipped. The rule exploits again negative conditions and $test$ labels to select the neighbour node with adequate values for $alpha$ and $beta$.

7.3.4 Link Reversal

At this point both $alpha$ and $beta$ in the $sink$ have been updated, so we can proceed with the reversal of all incoming $next$ edges in the $sink$ which originate from neighbours with a smaller height. Rule REVERSAL of Figure 7.10 works lexicographically w.r.t. the heights of the $sink$'s

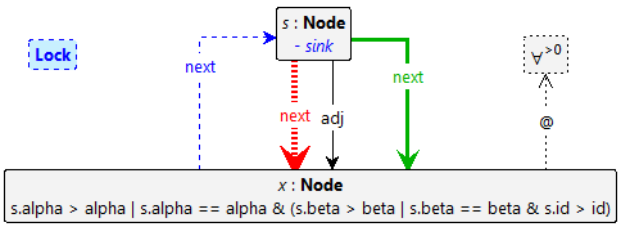


Figure 7.10: REVERSAL: Reversal of virtual edges according to the new height.

neighbours, exactly as NEW-LINK, except that it changes the orientation of all incoming $next$ edges instead of adding new ones. The dashed blue edge $next$ is used to remove an existing $next$ edge. The dashed thick red edge $next$ is used to test the absence of a reversed $next$ edge, while the green edge specifies the addition of the reversed $next$ edge. Via universal quantification, we specify that the same updates must be applied to all subgraphs of the host graph that match the specified pattern. The rule must be read as existentially quantified on s (to fix a $sink$ node) and universally quantified on x (to specify reversal of edges from x to s for every neighbour of x of s). The rule also deletes the $sink$ label from the selected node s, and removes the **Lock** node to terminate the reversal phase of the selected sink. New sink nodes may appear as a result of the

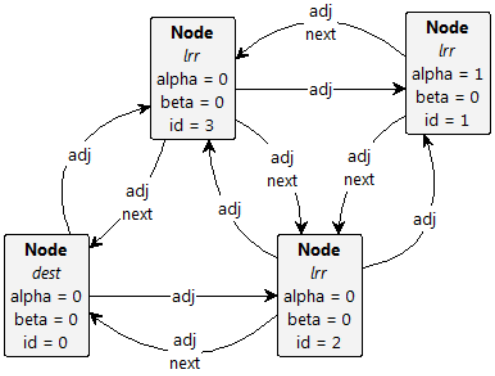


Figure 7.11: Example network without sinks.

first series of reversals. This would trigger another sequence of rule applications to propagate the height update, starting again from SINK-ALPHA. In the case of the example configuration in Figure 7.7, a complete run of link reversal in the sink node with $id = 1$ would result in the configuration of Figure 7.11. Its $alpha$ has been updated from 0 to 1, $beta$ is still 0 (since there is no neighbour with $alpha = 1$), and all of its $next$ edges have been reversed.

In the next section we will discuss how to apply GROOVE to validate our model.

7.4 Bounded Model Checking

Based on our model, we have performed different types of analysis using the GROOVE model checker for configurations of fixed size.

The analysis starts from a fixed initial topology, i.e., we fix the number of nodes and the adj-edges as in the four nodes example in Figure 7.2. Since the protocol addresses route maintenance in wireless networks, in order to test our implementation we have to introduce rules to model dynamic changes to the underlying connectivity network. Such changes will trigger the link

reversal phases. Since updates non-deterministic, the initial topology can change in arbitrary ways.

The rules LINK-ADD (Figure 7.12) and LINK-DEL (Figure 7.13) have the lowest possible priority. In this way changes to the network will occur only when the protocol is stable, i.e., when there are no more sinks.

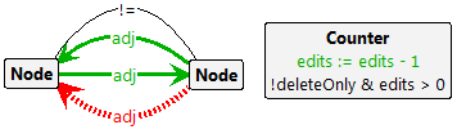


Figure 7.12: LINK-ADD: Addition of a new link between two nodes.

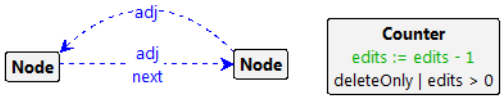


Figure 7.13: LINK-DEL: Deletion of a link between two adjacent nodes.

We put an upper bound to the number of dynamic changes to the network in the following way. The boolean field $deleteOnly$ of the **Counter** node is used to distinguish between two variants of bounded model checking. When it is set to $true$, rule LINK-ADD is disabled while LINK-DEL may fire without restrictions. When $deleteOnly$ is set to $false$, both rules are enabled, but just for a limited number of times decided by the $edits$ field of the **Counter**.

Link deletions may lead to partitionings, i.e., configurations where subsets of lrr nodes do not have a path of adj edges to the destination. In such cases the Gafni-Bertsekas protocol is known to diverge, as there is no mechanism to detect partitions. With rule BOUND-PARTITION (Figure 7.14) we filter out every such divergent execution. This means also that we do not need a

Figure 7.14: BOUND-PARTITION: Detection of a partitioning.

bound for the heights of the nodes, because only terminating executions are left. The rule has a rather high priority and no side effects because we want to block the computation as soon as a partitioning arises (matching configurations become final states). Thanks to a NAC expressed as a regular expression, the rule matches as soon as the current configuration contains an lrr node without a path to the destination.

The objective of our analysis is to check loop-freedom and ensure that, when the protocol is stable, every lrr node has at least a route to the destination. The rule LOOP (Figure 7.15) has high priority and, provided the network is sink-free, matches as soon as a node exposes a loop of any length of $next$ edges. The rule DISCONNECTED (Figure 7.16) has the lowest possible

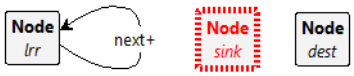


Figure 7.15: LOOP: Loop detection in routes.

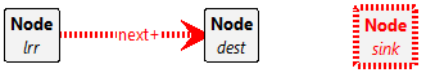


Figure 7.16: DISCONNECTED: Bad routes detection.

priority and it is very similar to BOUND-PARTITION, except that the regular expression on paths to the destination uses $next$ instead of adj.

7.4.1 Correctness Requirements

We formally specify the correctness of the algorithm via the following CTL formula.

$$AG\ !(\text{DISCONNECTED} \mid\mid \text{LOOP})$$

For a fixed initial configuration, the formula requires that in every derivation π of the LTS associated to our model (temporal connective A), and in every configurations G occurring in π (temporal connective G), it is never possible to fire the transitions DISCONNECTED and LOOP (! [resp. $||$] denotes negation [resp. disjunction]).

Following from the order induced by the priorities associated to the rules, the formula is true only if after each route maintenance phase that does not partition the topology, the protocol repairs the routes for every sink node. Thus the combination of execution strategies and CTL property can be used to formalize the correctness of the Gafni-Bertsekas algorithm.

7.4.2 Experiments and Evaluation

We checked the CTL property for a number of different bounds on the number of nodes and edits (link modifications). The tests were conducted on a common laptop with an Intel i5 CPU @ 2.53GHz and 4GB of RAM. The results of state-space exploration are listed in Table 7.1. For

#nodes	$deleteOnly$	$edits$	#states	time (s)
4	$true$	-	143	1
	$false$	7	4503	4
		8	7642	5
		9	13040	6
		10	20963	8
		11	34028	11
		12	52120	16
		13	80690	22
		14	118060	32
		15	174644	48
		16	244769	78
5	$true$	-	3556	4
	$false$	3	4416	4
		4	10732	7
		5	24541	10
		6	55487	19
		7	124001	35
		8	279591	109
6	$true$	-	172166	67
	$false$	2	65936	22
		3	271035	99

Table 7.1: Experimental results.

each number n of nodes considered, with $deleteOnly$ equal to $true$ we started the exploration from the fully connected configuration with n nodes (dynamic modifications are used then to generate arbitrary topologies). On the other hand, when $deleteOnly$ is equal to $false$, we fix an initial configuration with n nodes and some missing edges. This is to ensure both LINK-ADD and LINK-DEL have different options to edit the connectivity graph right from the start.

The CTL correctness specification holds in every considered test-case. As expected, by removing the detection of partitionings the algorithm does not terminate (heights grows unboundedly).

The analysis scales up to 6 nodes with only deletions or at most 3 edits. With 4 [resp. 5] nodes we managed to consider at most 16 [resp. 8] edits. The low number of nodes considered in initial configurations is due to the fast growing of the state space (e.g. more than 750000 graphs with 7 nodes). The high number of combinations is also due to the presence of heights and, in particular, identifiers. On one hand they are a basic feature of the height-based version of the protocol. The consistency between these information and the virtual DAG is part of the correctness requirements of the algorithm. On the other hand they reduce the application of symmetry reduction (graph isomorphism) during state exploration. The use of symmetry reductions guided by user-defined observations (e.g. restricting isomorphisms to adj- and $next$-edges) could be useful here as an heuristics to reduce state-space.

7.5 The Paxos Consensus Algorithm

The description of Paxos in [Lam01] distinguishes three separate agent roles: *proposers* that can propose values for consensus, *acceptors* that accept a value among those proposed, and *learners* that learn the accepted values and eventually choose one of them. We present the protocol on the basis of a pseudo-code description from the lecture nodes [MMM13].

In a first step, the proposer selects a fresh round identifier and broadcasts it to all live acceptors, in a message called $Prepare$. It then collects votes for that round number from acceptors. Acceptors' replies, called $Promise$s, contain the round number and a pair consisting of the last round and value that they promised in previous rounds (with the same or a different proposer). Rounds and values are initialized to the default of -1, and only change upon $Accept$ messages. When the proposer checks that a majority is reached, it selects a value to submit again to the acceptors. For the selection of this value, the proposer inspects every $Promise$ received in the current round and selects the value with the highest non-default round; if it did not receive a non-default value, it uses its own initial proposal ($myval$). It then submits the current round and the chosen value to the acceptors, in a message called $Accept$.

Acceptors wait for proposals (i.e., $Prepare$ messages) of round identifiers but consider only those that are *fresh*, in the sense of being higher than the last one they have seen so far. If the received round is fresh, acceptors answer with a $Promise$ not to accept proposals with smaller round

Paxos — Proposer p

constants
$A =$ set of live acceptors
$maj = \lceil (\#A + 1)/2 \rceil$
init $crnd \leftarrow -1$ */* current round */*

on $\langle Propose, val \rangle$
/ pick fresh round */*
$crnd \leftarrow pickNextRound(crnd)$
$myval \leftarrow val$
$P \leftarrow \emptyset$
send $\langle Prepare, crnd \rangle$ **to** A

on $\langle Promise, rnd, prnd, pval \rangle$ with $rnd = crnd$
from acceptor a
$P \leftarrow P \oplus (prnd, pval)$

on event $\#P \geq maj$
$j = \max\{prnd \mid (prnd, pval) \in P\}$
if $j \geq 0$ **then**
$V = \{pval \mid (j, pval) \in P\}$
/ pick value with largest* $prnd$ **/*
$myval \leftarrow pick(V)$
send $\langle Accept, crnd, myval \rangle$ **to** A

Paxos — Acceptor a

constants $L =$ set of learners
init $crnd \leftarrow -1$ */* current round */*
$prnd \leftarrow -1$ */* previous round */*
$pval \leftarrow -1$ */* previous value */*

on $\langle Prepare, rnd \rangle$ with $rnd > crnd$
from proposer p
$crnd \leftarrow rnd$
send $\langle Promise, crnd, prnd, pval \rangle$ **to** p

on $\langle Accept, rnd, aval \rangle$ with $rnd \geq crnd$
from proposer p
$crnd \leftarrow rnd$
$prnd \leftarrow rnd$
$pval \leftarrow aval$
send $\langle Learn, crnd, aval \rangle$ **to** L

Paxos — Learner l

constants
$A =$ set of live acceptors
$maj = \lceil (\#A + 1)/2 \rceil$
init $V \leftarrow \emptyset$

on $\langle Learn, rnd, lval \rangle$ **from** acceptor a
$V \leftarrow V \oplus (rnd, lval)$

on event $\exists m = (rnd, lval) : \#\{m \mid m \in V\} \geq maj$
choose $lval$

Figure 7.17: Pseudo-code of the Paxos protocol

numbers. Since messages might arrive out-of-order, even different $Prepare$s with increasing rounds of the same proposer might arrive in arbitrary order (this justifies the need of the $Promise$ message). Acceptors also wait for $Accept$ messages: in that case local information about the current round is updated and, if the round is fresh, the accepted pair $(rnd, aval)$ is forwarded to the learner, in a message called $Learn$.

A learner collects votes ($Accept$ messages) on pairs $(rnd, lval)$ sent by acceptors and waits to detect a majority for one of them. When a majority is detected, the $lval$ component is chosen.

The pseudo-code of the algorithm, based on [MMM13], is given in Figure 7.17. Majority is defined as maj. The pseudo-code includes a special $Propose$ message that corresponds to an external command sent to the node in order to inject a new proposal (a value) into the whole system. In the proposer code, $pickNextRound$ must return a fresh value (w.r.t. all processes) for the next round, and $pick$ must return the value associated to a tuple with highest round number. We use $\oplus$ to denote multiset union (which is required to count multiple occurrences of the same pair).

The protocol is guaranteed to reach consensus for $maj \geq \lceil (\#A + 1)/2 \rceil$, where $\#A$ denotes the size of the set A of correct acceptors, and only if acceptors and learners have enough time to take a decision (i.e., to detect a majority). If proposers indefinitely inject new proposals, the protocol may diverge. Here we will concentrate on the following limited notion of correctness:

Definition 24 (safety)**.** *The protocol is correct if, when a value is chosen by a learner, it has been proposed by a proposer, and no other value has been chosen by any learner in previous rounds of the protocol.*

This means that, whenever a value is chosen by a learner, any successive choices always select the same value (possibly with larger round identifiers); i.e., the algorithms stabilizes w.r.t. the value components of tuples sent to the learners.

Simplifying assumptions In both the GROOVE and the SPIN model we present, we have made the following simplifying assumptions about the protocol:

- Proposers never send more than one $Prepare$ message. This does not restrict the protocol for the purpose of the correctness criterion in Def. 24 (the effect of multiple $Prepare$ messages may be mimicked by increasing the number of proposers) but it causes the protocol to always terminate.
- There is only a single learner. This cannot affect the correctness of the protocol either, as all learners have access to exactly the same information and hence are bound to have the same behaviour. In other words, any error in a scenario with multiple learners must necessarily occur already with a single learner.

7.6 Paxos: Formal Specification using GROOVE

In the graph-based model, the global states of the protocol are captured by single graphs. Each such graph is typed according to the type graph in Fig 7.18.

As the type graph shows, there are two abstract types, **Process** and **Message**: each **Message** has a round number rnd during which it was sent, and a sender (which is a **Process**). In addition there is a type **Counters**, which will always have a singular instance that serves as a container for the global variables maj (the bound considered to be a majority) and nextRnd (an auxiliary variable used to dispense initial round numbers). There are three types of **Process** and four types of **Message**, corresponding to the roles and messages of the protocol. The arrows and attributes correspond to the local fields and variables discussed in Sect. 7.5. In addition, the following may be noted:

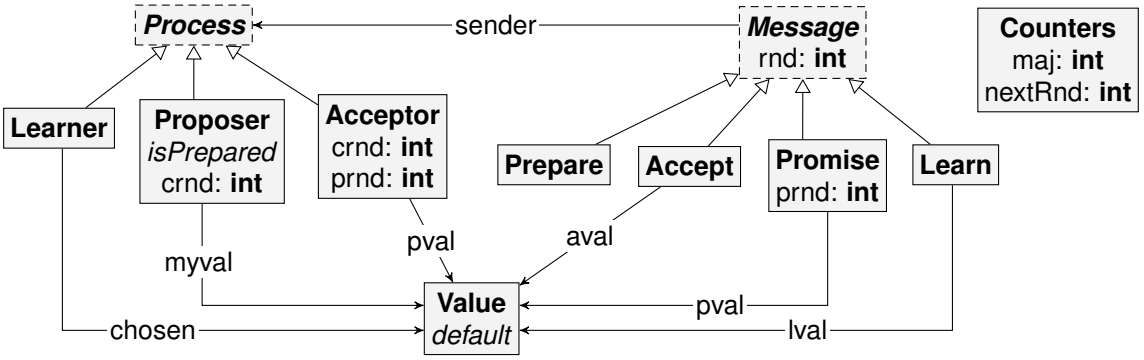


Figure 7.18: Type graph of the Paxos protocol

- Processes and messages have no explicit identities. This is important in order to ensure that symmetrical states give rise to isomorphic graphs.
- **Proposer** instances have a flag isPrepared, which will be set when a proposer has sent a **Prepare** message and is ready to receive **Promise**s.
- The values proposed and chosen by the protocol are not represented as integers but as nodes of type **Value**. The default flag on **Value** will be used to distinguished the default value with which all acceptors are initialised (-1 in the pseudocode of Figure 7.17).

Figure 7.19 shows an initial configuration with three proposers, four acceptors and a majority bound of 2. The protocol is expected to be incorrect in this case, as the majority does not exceed half of the acceptors.

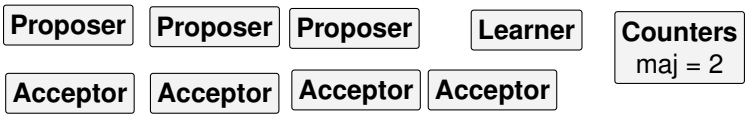


Figure 7.19: Example initial configuration of the Paxos protocol

Initialization The dynamics of the protocol are captured by transformation rules. In addition, the model is equipped with a control program to schedule the rules. Figure 7.20 shows the main control loop as well as the initialization rule.

The control loop specifies that the rule initValue is to be invoked, followed by a perpetual choice between proposer, acceptor and learner actions, as specified by the functions proposer() etc. (see below).

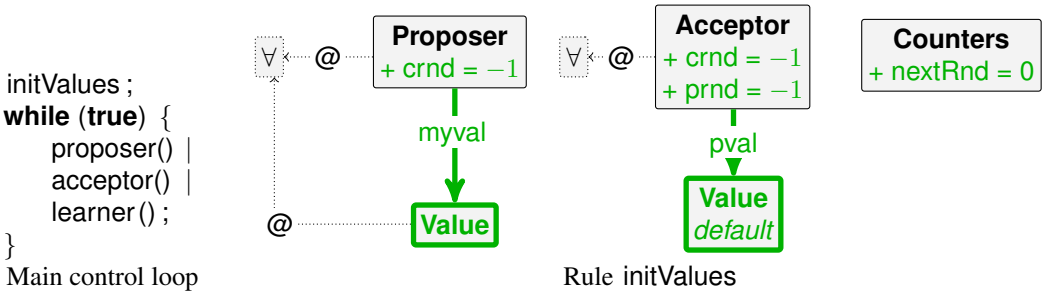


Figure 7.20: Top-level control and initialization rule

The rule deserves clarification. The ∀-quantifiers cause all nodes connected with dashed @-labelled arrows to be matched as often as possible. The fat, gray **Value** nodes (green in a coloured view) are created as a result of the rule, as are the +-prefixed attributes in the **Proposer**, **Acceptor** and **Counters** nodes. For instance, applying the rule changes the graph of Figure 7.19 into Figure 7.21.

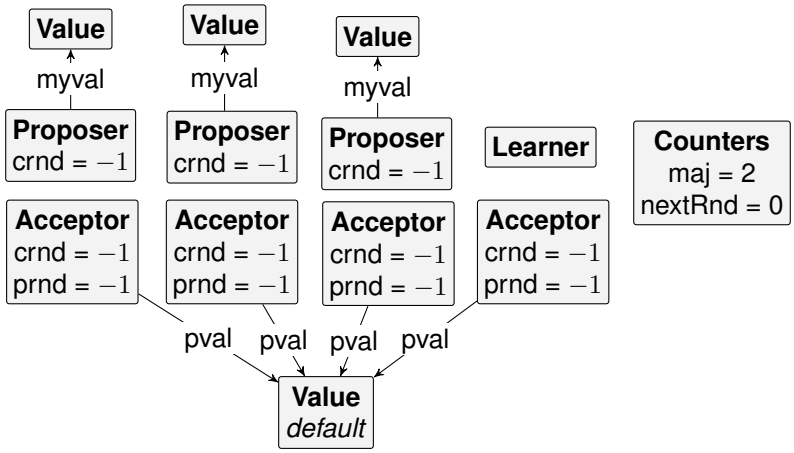


Figure 7.21: Initial configuration of Figure 7.19 after application of initValues

Proposers Figure 7.22 shows the control function and the rules that encompass the functionality of proposers, as specified by the pseudocode in Figure 7.17. The control function proposer() specifies a non-deterministic choice between the rule onPropose on the one hand, corresponding to the "**on** *Propose*" clause of Figure 7.17, and a sequence of onPromise, changeMyval and sendAccept on the other (where **try** causes changeMyval to be applied only if possible), corresponding to the "**on** *Promise*" and "**on event**" clauses. The parameter prop ensures that the rules are applied to the same proposer instance.

- Rule onPropose specifies the update of the proposer's crnd attribute and the creation of a

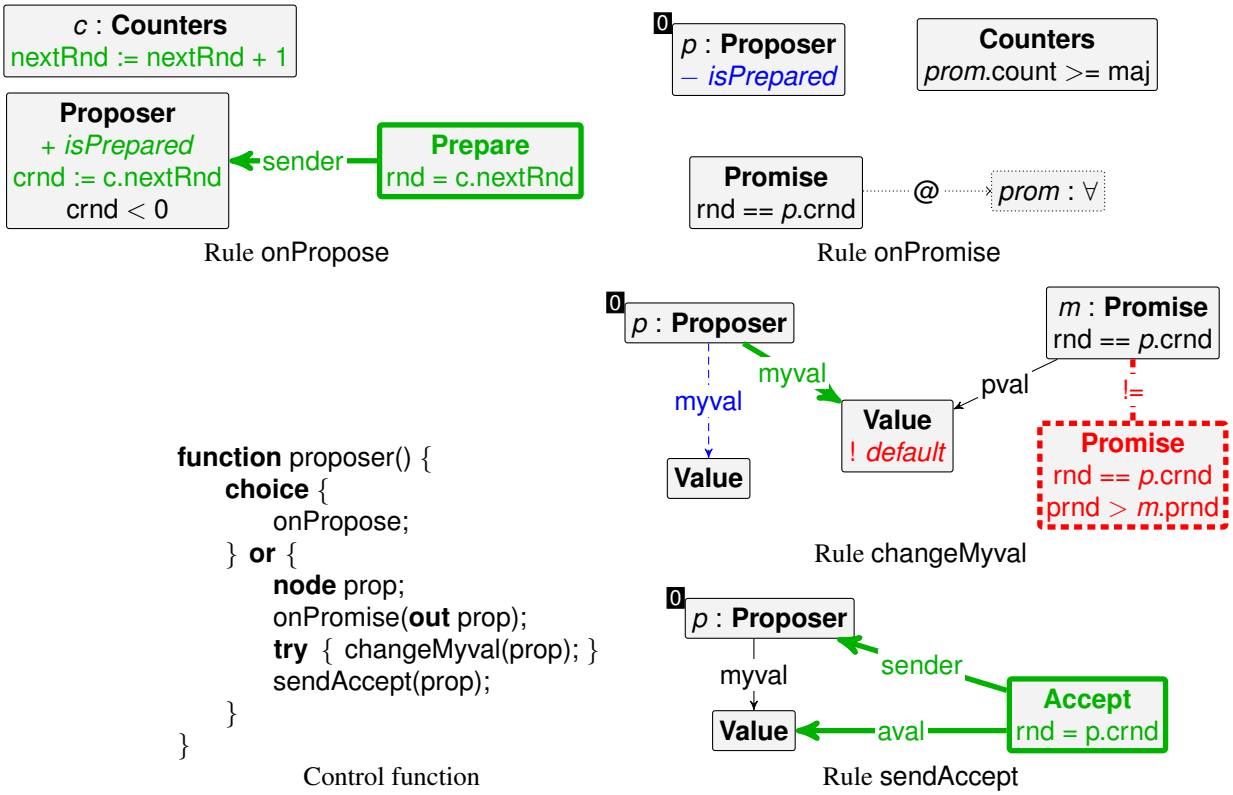


Figure 7.22: Proposer behaviour

Prepare message, under the condition that crnd ¡ 0. Moreover, the isPrepared flag is set.

- Rule onPromise tests if the number of **Promise** messages with this proposer's round number exceeds the majority bound. Note that prom.count, where prom refers to the ∀-quantifier, stands for the number of matches of the @-connected subgraph — in this case, just the **Promise** node. The isPrepared flag is a precondition for this rule and is at the same time deleted, making sure that each proposer can execute this event only once.

  The adornment in the top left of the **Proposer** indicates that this node is a rule parameter. When the rule is applied, the value of this parameter is bound to the prop-variable in the control program.

- Rule changeMyval adjusts the myval field to the pval of the promise with the highest prnd value, but only if that is not the default value. (The dashed **Promise**-node — red in a coloured view— with the !=-labelled edge to the top right **Promise** specifies that there is *no* promise with a *higher* prnd.)

- Finally, rule sendAccept specifies that an **Accept** message is sent. Due to the scheduling in the control program, this only occurs after changeMyval has had a chance to be applied.

Acceptors and learners Figure 7.22 shows the control function and rules for the acceptor and learner roles. The control functions merely specify a choice between rules, which in turn capture

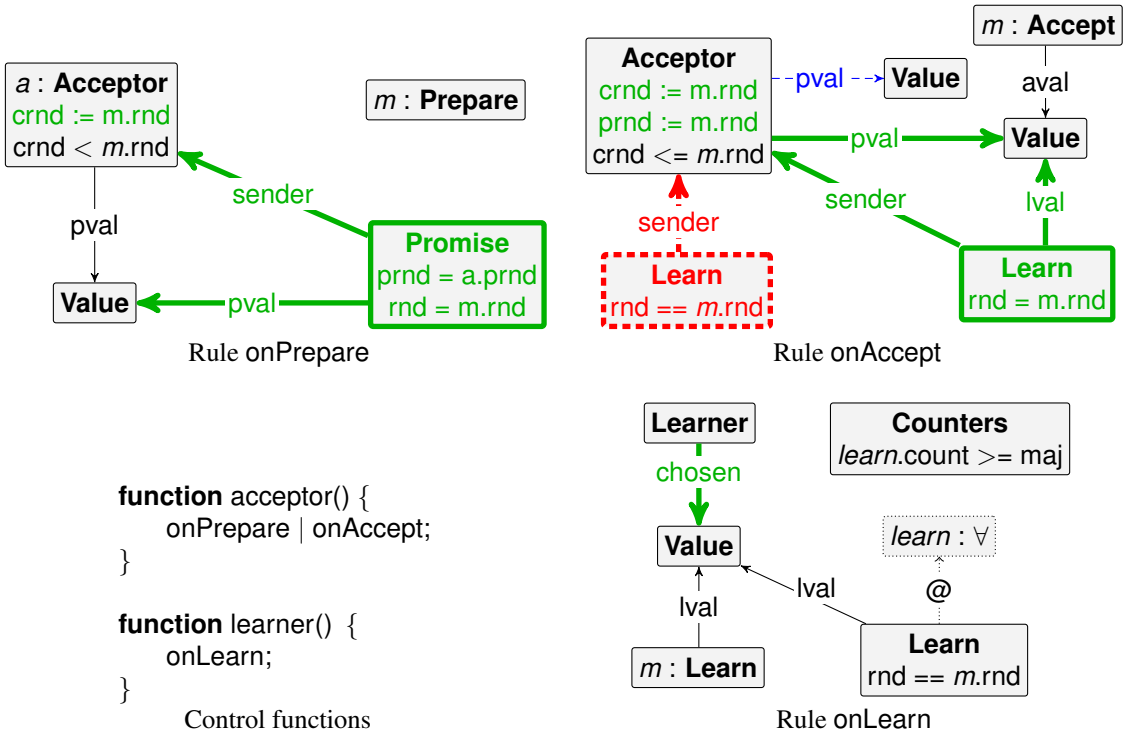


Figure 7.23: Acceptor and learner behaviour

the corresponding part of the pseudocode of Figure 7.17. We will discuss the rules.

- Rule onPrepare creates a **Promise** message upon discovery of a **Prepare** with the right rnd.
- Rule onAccept creates a **Learn** message upon discovery of an **Accept** with the right rnd. The dashed (red) **Learn**-node is a negative condition ensuring that the rule is applicable at most once for any given **Acceptor** and **Accept**.
- Rule onLearn counts the number of **Learn** messages with identical rnd and lval fields, in the same way as onPromise of Figure 7.22, and chooses the corresponding **Value** if the count has reached the majority bound. Note that there is nothing to prevent this rule from being applied more than once; however, the same value will be chosen every time.

Correctness Based on the combination of control and rules presented above, GROOVE can generate (and optionally visualise) the state space. Moreover, for the purpose of actually validating a model against a set of requirements, GROOVE has built-in LTL and CTL model checkers.

However, for the problem at hand, model checking is overkill, as we just want to check safety as in Definition 24. To achieve this, it suffices to try and find graphs that are *unsafe*. If that attempt fails, the protocol is correct for the initial configuration. The negated safety property is captured by the rules in Figure 7.24. Note that neither of these rules actually modifies the graph.

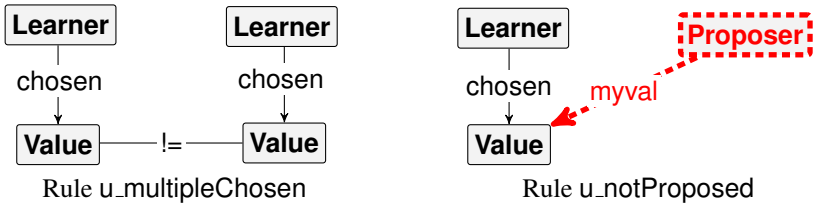


Figure 7.24: Safety rules encoding the negation of the property in Def. 24

- Rule u_multipleChosen tests whether two **Learner**s have chosen distinct **Value**s. The distinctness is explicitly required by the !=-edge. Due to the fact that rules may be matched non-injectively, this rule is also applicable to a graph in which a *single* **Learner** has chosen two distinct **Value**s.
- Rule u_notProposed tests whether a **Learner** has chosen a **Value** that has not been proposed by any **Proposer**.

GROOVE supports a range of different exploration strategies. For this particular case it can do a depth-first search that halts as soon as a graph is found that satisfies the propositional formula u_multipleChosen || u_notProposed. If no such state exists, this strategy will cause the entire state space to be searched, potentially encompassing (many) millions of states.

7.7 Paxos: Formal Specification using SPIN

We now present a formal specification of Paxos in PROMELA, the input language of the model checker SPIN. A PROMELA specification consists of a number of processes communicating through shared channels. The processes are obtained by instantiating so-called **proctype** templates. We call this type of model *vector-based* because of the way the states are encoded during state space exploration, namely as vectors of values (representing local variables and channel contents).

Initialisation. For Paxos, we define three **proctype**s, corresponding to the roles of the protocol. For instance, the initial configuration shown for GROOVE in Fig 7.19 is specified in PROMELA as

```
init { atomic {
   run proposer(1,1); run proposer(2,2); run proposer(3,3);
   run acceptor(0); run acceptor(1); run acceptor(2); run acceptor(3);
   run learner();
} }
```

We will discuss the template definitions and their parameters below. The channels and majority bound are defined as follows:

```
#define MAJ 3    /* majority bound */
#define MAX 10   /* maximum channel capacity */
chan prepare = [MAX] of { byte, byte };          /* prop to acc:  id, crnd          */
chan accept  = [MAX] of { byte, byte, short };   /* prop to acc:  id, crnd, myval   */
chan promise = [MAX] of { byte, short, short };  /* acc to prop:  crnd, prnd, pval  */
chan learn   = [MAX] of { short, short, short }; /* acc to learn: id, crnd, aval    */
```

Every channel conveys messages of one of the four types in the protocol. Any process can access any of the channels for sending or receiving. For the purpose of this protocol we always use !! to insert messages in channels in lexicographic order and ?? to select the first matching message from them; this effectively turns the channels into multisets of messages while keeping channels in canonical form (ordered lists).

Templates. Figure 7.25 shows the proposer and acceptor templates. Proposers are identified by a unique value passed in as the first parameter, crnd. The other parameter, myval, is the proposed value. The template starts with a bprepare invocation, which causes a broadcast of prepare messages. This is followed by a non-deterministic **do**–**od** loop which is exited once an accept message has been broadcast. In this loop, in an atomic step, first the messages in the promise buffer with the right rnd parameter are counted by iterating over all the messages (using a temporary artificial message with non-existent round number as marker); then, if the count exceeds the majority, the accept is broadcast.

The acceptor template of Figure 7.25 is very similar to the pseudocode of Figure 7.17. The id parameter is required to address individual acceptors in prepare and accept messages. Finally, the learner template is defined in Figure 7.26. The counters mcount[rnd] keep track of the number of received learn messages associated to round rnd.

Modelling choices. In order to keep the state space size as small as possible, we have taken the following measures in the PROMELA model:

- All communication uses the ordered send and unordered receive operators !! and ??.
- Sequences of statements are wrapped in **d_step** and **atomic** blocks wherever possible. Their execution is thus represented by a single transition, not interrupted by other processes

```
proctype proposer(int crnd; short myval) {
  short rnd, prnd, pval, hr=−1, hv=−1;
  byte count;
  bprepare(crnd);
  do :: atomic {
        d_step {
          promise!9,0,0;
          do :: promise?rnd,prnd,pval −>
                if :: rnd == 9 −> break;
                   :: rnd == crnd −>
                      count++;
                      if :: prnd > hr −>
                            hr = prnd; hv = pval
                         :: else fi ;
                   :: else fi ;
                promise!rnd,prnd,pval
          od }
        if :: count >= MAJ −>
              baccept(crnd, hv<0 −> myval : hv);
              break
           :: else fi ;
        prnd=0; pval=0; hv=−1; hr=−1; count=0;
  } od
}
```

```
proctype acceptor(int id) {
  short crnd=−1, prnd=−1, pval=−1;
  short aval, rnd;
  do
  :: atomic {
       prepare??eval(id),rnd −>
       if :: rnd > crnd −>
             crnd = rnd;
             promise!!crnd,prnd,pval;
          :: else fi ;
       rnd = 0 /* reset */
     }
  :: atomic {
       accept??eval(id),rnd,aval −>
       if :: rnd >= crnd −>
             crnd = rnd;
             prnd = rnd;
             pval = aval;
             learn !! id,crnd,aval
          :: else fi ;
       rnd = 0; aval = 0 /* reset */
     }
  od
}
```

```
inline baccept(rnd,val) {
  accept!!0,rnd,val; accept!!1,rnd,val; accept!!2,rnd,val; accept!!3,rnd,val
}
inline bprepare(rnd) {
  prepare!!0,rnd; prepare!!1,rnd; prepare!!2,rnd; prepare!!3,rnd
}
```

Figure 7.25: Proposer and acceptor templates, with inlined broadcast primitives.

(unless a statement blocks inside an **atomic** statement).

- The proposer uses a "quorum transition" scheme for atomically counting the relevant messages, while avoiding a state change in the promise buffer.
- Local variables are reset to their initial values when they are no longer needed.
- Message count variables are no longer increased after they reach the majority bound.

```
active proctype learner() {
  short lastval = −1, id, rnd, lval;
  byte mcount[MAX];
  do :: d_step {
          learn??id,rnd,lval −>
          if :: mcount[rnd] < MAJ −> mcount[rnd]++ :: else fi;
          if :: mcount[rnd] >= MAJ −>
                if :: lastval >= 0 && lastval != lval −> assert(false)
                   :: lastval == −1 −> lastval = lval
                   :: else fi
             :: else fi;
          id = 0; rnd = 0; lval = 0 /* reset */
  } od
}
```

Figure 7.26: Learner process template.

7.8 Analyzing Paxos: GROOVE versus SPIN

GROOVE experimental results. Table 7.2 shows the outcome of running GROOVE on the graph-based model with different start states. These results were obtained using an Intel i7–2600 (64 bits) CPU at 3.4 GHz under Windows 7, running Java 6 with 8 GB of memory.

Each row reports the state count and time used for state space exploration with a given number of proposers (first column) and acceptors (second column), for a majority bound that is too low (columns 3–5), respectively just high enough (columns 6–8) for the protocol to be correct. In all cases where the bound is too low, a violation of the safety properties (Def. 24) was found well before the full state space was explored; in contrast, with the bound high enough, no violations were found and hence all reachable states were exhaustively enumerated. The number of states in the second case is typically several orders of magnitude higher than in the first case. Every next step not reported in the table (a higher number of proposers for the same number of acceptors, or vice versa) causes the full state space exploration to run out of memory.

Figure 7.27 shows two graphs derived from the results in Table 7.2. The first graph shows how the state space size (for full exploration) scales with the number of acceptors, for 2, 3 and 4 proposers. We call attention to the following phenomena:

- The vertical axis has a logarithmic scale; clearly the state space grows exponentially with the number of acceptors. Moreover, the growth much is accelerated for larger numbers of proposers, from less then 1 order of magnitude for each next acceptor (2 proposers) to around 2 orders of magnitude (4 proposers)
- The increase has a slight sawtooth shape, due to the fact that the majority bound does not increase with every next acceptor, but with every *second* additional acceptor.

		Low majority (w. iso)			High majority (with iso)			High majority (no iso)		
Proc	**Acc**	**Maj**	**States**	**Time (ms)**	**Maj**	**States**	**Time (ms)**	**Maj**	**States**	**Reduct**
2	2	1	45	110	2	78	160	2	224	65.18%
2	3	1	55	150	2	757	790	2	6,882	89.00%
2	4	2	174	360	3	1,279	1,460	3	31,256	95.91%
2	5	2	222	550	3	9,729	6,970	3	1,178,114	99.17%
2	6	3	723	1,820	4	15,783	11,521	4	–	–
2	7	3	911	3,100	4	92,289	69,316	4	–	–
2	8	4	2,574	4,960	5	143,376	131,028	5	–	–
2	9	4	3,154	4,982	5	665,564	844,890	5	–	–
2	10	5	7,729	10,581	6	992,044	1,529,461	6	–	–
2	11	5	9,233	29,963	6	3,820,671	7,698,559	6	–	–
2	12	6	20,192	40,804	7	5,491,406	14,794,452	7	–	–
3	2	1	51	120	2	677	581	2	6,674	89.86%
3	3	1	62	160	2	32,899	7,032	2	1,079,582	96.95%
3	4	2	610	842	3	98,330	25,983	3	–	–
3	5	2	407	900	3	3,880,277	6,681,550	3	–	–
3	6	3	1,481	3,411	4	12,247,549	8,771,175	4	–	–
4	2	1	58	140	2	6,082	2,020	2	260,910	97.67%
4	3	1	246	350	2	1,523,338	940,095	2	–	–
4	4	2	1,258	1,710	3	9,337,923	4,523,411	3	–	–
5	2	1	66	150	2	55,420	9,131	2	12,743,315	99.57%
6	2	1	75	170	2	506,370	77,888	2	–	–
7	2	1	85	190	2	4,607,455	1,154,561	2	–	–

Table 7.2: GROOVE model checking results, with and without isomorphism reduction

The second graph shows the state space size and running time for an increasing number of proposers and a fixed number of 2 acceptors. The following can be remarked:

- The increase is again exponential; in fact, the exponent is larger than for the case of 2 proposers and increasing number of acceptors. This can be explained by the fact that, in contrast to the acceptors, the proposers essentially do *not* engender symmetry, since they are assigned identities through the $crnd$ attribute.
- The time required by the analysis essentially keeps step with the state space size, being slightly worse for smaller problem sizes due to initialization effects and for larger problem sizes due to garbage collection.

Finally, we want to draw attention to the tool performance, in terms of number of states generated per second. This fluctuates for different problem sizes, from under 400 to over 6000. The symmetries in the pool of acceptors cause the analysis to be slower for p proposers and a acceptors than for a proposers and p acceptors, as the isomorphism checking necessary to detect these symmetries is computationally expensive (see [Ren07] for an extensive discussion of the principles behind the symmetry reduction of GROOVE). In fact, profiling shows that for $(p, a) = (2, 11)$,

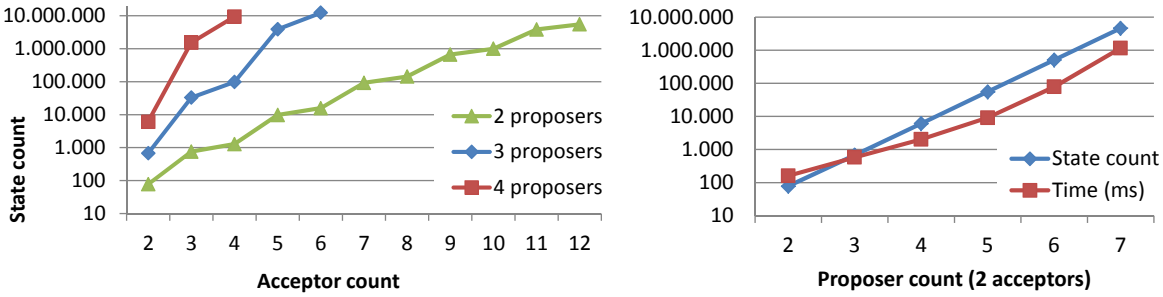


Figure 7.27: Graphs for the results of Table 7.2

which has the lowest rate of generated states/s, over 75% of the total computation time is spent in isomorphism checking.

Isomorphism reduction. The last two columns of Table 7.2 also show the reduction due to isomorphism checking, for those problem instances for which GROOVE was able to compute the state space without reduction. Clearly, the gain is enormous, and easily justifies the time spent in computing isomorphism. The same effect was reported, more extensively, in [CvR08].

SPIN experimental results. To compare graph- and vector-based representations, we have applied the Spin model checker to the Paxos Promela model presented above. The experiments in Table 7.3 are obtained on an i7 processor with 6GB of RAM running jSpin version 7 under Windows 7 with default option for memory management. Without use of underapproximated

		Low majority					High majority				
P	**A**	**Maj**	**States**	**States***	**Time**	**Time***	**Maj**	**States**	**States***	**Time**	**Time***
2	2	1	437	437	5	4	2	620	620	5	2
2	3	1	2,292	2,292	47	15	2	38,562	38,562	343	206
2	4	2	66,748	66,748	2,500	559	3	343,844	343,844	4,400	2,540
2	5	2	387,982	387,982	7,910	4,290	3	–	32,983,781	–	351,000
2	6	3	–	8,893,356	–	135,000	4	–	63,305,564	–	847,000
2	7	3	–	44,537,332	–	802,000	4	–	67,887,523	–	1,220,000
3	2	1	4,622	4,622	31	31	2	19,536	19,536	195	114
3	3	1	52,151	52,151	874	496	2	–	7,234,348	–	56,900
3	4	2	–	4,532,701	–	60,600	3	–	63,845,934	–	85,156
4	2	1	50,607	50,607	809	415	2	617,129	617,129	8,500	5,600
4	3	1	1,198,754	1,198,749	30,200	15,900	2	–	66,611,998	–	789,000
4	4	2	–	59,715,947	–	1,140,000	3	–	70,322,030	–	1,330,000

Table 7.3: SPIN model checking results (time is in ms): – indicates an out of memory error; * indicates results obtained via bit-hashing.

search, with 2 proposers Spin is capable of finding violations up to 5 acceptors, whereas it can prove safety up to 4 acceptors. With 3 proposers, violations cannot be detected with more than

3 acceptors, and safety cannot be proved for more than 2 acceptors. A similar result is obtained for 4 proposers. By applying underapproximation heuristics like bit-hashing, Spin can deal with larger configurations while still detecting violations with low majorities, as shown by the $*$-marked columns in Table 7.3.

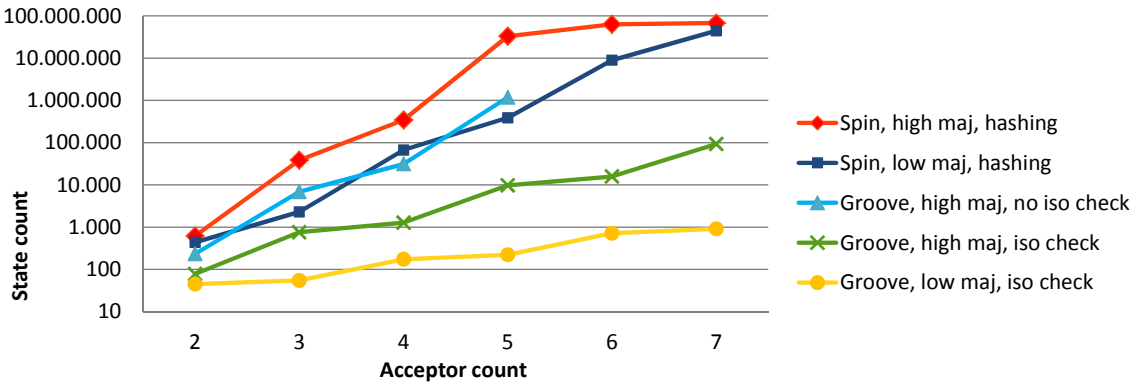


Figure 7.28: GROOVE versus SPIN state space sizes, 2 proposers

7.9 Conclusions

In this chapter we have presented declarative models of the Gafni-Bertsekas LRR protocol and of the Paxos consensus algorithm, obtained via graph transformation rules as those used as input language for the GROOVE simulator and model checker. The use of extended graph transformation rules, e.g., with negative conditions and nested quantification, allows to naturally compile pseudo-code in a declarative specification in GROOVE. Furthermore, when compared to analysis with more traditional verification tools like SPIN, experimental results with the GROOVE model checker show an impressive reduction of the state-space obtained via symmetry reductions based on graph isomorphism. These reductions fully exploit the underlying graph-representation of the configurations in which a key point is to use anonymous nodes to denote values and identifiers (without need of introducing integers or other enumerative types). For two proposers, Figure 7.28 shows the difference, in logarithmic scale, between GROOVE with iso-check and SPIN with bitstate hashing.

The case studies we considered and the experimental comparison show that well engineered graph-based search engines can compete with vector-based enumerative engines, even on non trivial protocol case-studies like Paxos. Comparisons with symbolic model checkers on this kind of protocols and other examples of distributed algorithms seems an interesting research direction to understand the limit of declarative model checker tools like GROOVE.

Bibliography

[ACJT96] P.A. Abdulla, K. Cerans, B. Jonsson, and Y.K. Tsay. General decidability theorems for infinite-state systems. In *Logic in Computer Science, 1996. LICS'96. Proceedings., Eleventh Annual IEEE Symposium on*, pages 313–321. IEEE, 1996.

[ADB11] P. A. Abdulla, G. Delzanno, and L. Van Begin. A classification of the expressive power of well-structured transition systems. *Inf. Comput.*, 209(3):248–279, 2011.

[ADM04] P.A Abdulla, J. Deneux, and P. Mahata. Multi-clock timed networks. In *Logic in Computer Science, 2004. Proceedings of the 19th Annual IEEE Symposium on*, pages 345–354. IEEE, 2004.

[ADR+11] Parosh Abdulla, Giorgio Delzanno, Othmane Rezine, Arnaud Sangnier, and Riccardo Traverso. On the verification of timed ad hoc networks. In *Proceedings of the 9th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'11)*, volume 6919 of *Lecture Notes in Computer Science*, pages 256–270. Springer, 2011.

[AJ93] Parosh Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. In *Logic in Computer Science, 1993. LICS'93., Proceedings of Eighth Annual IEEE Symposium on*, pages 160–170. IEEE, 1993.

[AJ96] P. A. Abdulla and B. Jonsson. Undecidable verification problems for programs with unreliable channels. *Inf. Comput.*, 130(1):71–90, 1996.

[AJ01] P.A. Abdulla and B. Jonsson. Ensuring completeness of symbolic verification methods for infinite-state systems* 1. *Theoretical Computer Science*, 256(1-2):145–167, 2001.

[AJ03] Parosh Aziz Abdulla and Bengt Jonsson. Model checking of systems with many identical timed processes. *TCS*, 290(1):241–264, 2003.

[Alu91] Rajeev Alur. *Techniques for automatic verification of real-time systems*. PhD thesis, Stanford University, 1991.

[APR+01] T. Arons, A. Pnueli, S. Ruah, J. Xu, and L. D. Zuck. Parameterized verification with automatically computed inductive assertions. In *CAV'01*, volume 2102 of *LNCS*, pages 221–234. Springer, 2001.

[BDK+12] N. Bertrand, G. Delzanno, B. König, A. Sangnier, and J. Stückrath. On the decidability status of reachability and coverability in graph transformation systems. In *RTA*, pages 101–116, 2012.

[BDL04] G. Behrmann, A. David, and K.G. Larsen. A tutorial on uppaal. *Formal methods for the design of real-time systems*, pages 33–35, 2004.

[BLL+95] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL — a Tool Suite for Automatic Verification of Real–Time Systems. In *Proc. of Workshop on Verification and Control of Hybrid Systems III*, number 1066 in Lecture Notes in Computer Science, pages 232–243. Springer–Verlag, October 1995.

[BOG02] K. Bhargavan, D. Obradovic, and C. A. Gunter. Formal verification of standards for distance vector routing protocols. *J. ACM*, 49(4):538–576, 2002.

[BPZ06] I. Balaban, A. Pnueli, and L. D. Zuck. Invisible safety of distributed protocols. In *ICALP (2)*, pages 528–539, 2006.

[BQV05] Roberto Baldoni, Leonardo Querzoni, and Antonino Virgillito. Distributed event routing in publish/subscribe communication systems: a survey. Technical report, DIS, Universita di Roma "La Sapienza", 2005.

[BSS10] P. Bokor, M. Serafini, and N. Suri. On efficient models for model checking message-passing distributed protocols. In *Formal Techniques for Distributed Systems*, pages 216–223. Springer, 2010.

[BVEG+87] H. Barendregt, M. Van Eekelen, J. Glauert, J. Kennaway, M. Plasmeijer, and M. Sleep. Term graph rewriting. In *PARLE Parallel Architectures and Languages Europe*, pages 141–158. Springer, 1987.

[CEP95] Allan Cheng, Javier Esparza, and Jens Palsberg. Complexity results for 1-safe nets. *Theoretical Computer Science*, 147(1):117–136, 1995.

[CFI96] G. Cécé, A. Finkel, and S. Purushothaman Iyer. Unreliable channels are easier to verify than perfect channels. *Inf. Comput.*, 124(1):20–31, 1996.

[CS08] P. Chambart and Ph. Schnoebelen. Mixing lossy and perfect fifo channels. In *CONCUR*, pages 340–355, 2008.

[CvR08] P. Crouzen, J. C. van de Pol, and A. Rensink. Applying formal methods to gossiping networks with MCRL and groove. *ACM SIGMETRICS performance evaluation review*, 36(3):7–16, December 2008.

[Del03] G. Delzanno. Constraint-based verification of parameterized cache coherence protocols. *FMSD*, 23(3):257–301, 2003.

[DEP99] G. Delzanno, J. Esparza, and A. Podelski. Constraint-based analysis of broadcast protocols. In *CSL*, pages 50–66, 1999.

[Din92] G. Ding. Subgraphs and well quasi ordering. *J. of Graph Theory*, 16(5):489–502, 1992.

[DRT14] Giorgio Delzanno, Arend Rensink, and Riccardo Traverso. Graph- versus vector-based analysis of a consensus protocol. In *3rd Workshop On GRAPH Inspection and Traversal Engineering (GRAPHITE), to appear*, 2014.

[DRV13] G. Delzanno and F. Rosa-Velardo. On the coverability and reachability languages of monotonic extensions of petri nets. *Theor. Comput. Sci.*, 467:12–29, 2013.

[DST13a] G. Delzanno, A. Sangnier, and R. Traverso. Parameterized verification of broadcat networks of register automata (technical report), 2013. Available at the URL *http://verify.disi.unige.it/publications*.

[DST13b] Giorgio Delzanno, Arnaud Sangnier, and Riccardo Traverso. Parameterized verification of broadcast networks of register automata. In *RP*, pages 109–121, 2013.

[DSTZ12a] G. Delzanno, A. Sangnier, R. Traverso, and G. Zavattaro. On the complexity of parameterized reachability in reconfigurable broadcast networks, 2012. Research Report hal-00740518, HAL, CNRS, France, 2012.

[DSTZ12b] Giorgio Delzanno, Arnaud Sangnier, Riccardo Traverso, and Gianluigi 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)*, volume 18 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 289–300, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[DSZ10] G. Delzanno, A. Sangnier, and G. Zavattaro. Parameterized verification of ad hoc networks. In *CONCUR*, pages 313–327, 2010.

[DSZ11] G. Delzanno, A. Sangnier, and G. Zavattaro. On the power of cliques in the parameterized verification of ad hoc networks. In *FOSSACS*, pages 441–455, 2011.

[DSZ12] Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. Verification of ad hoc networks with node and communication failures. In *Formal Techniques for Distributed Systems*, pages 235–250. Springer, 2012.

[DT12] Giorgio Delzanno and Riccardo Traverso. A Formal Model of Asynchronous Broadcast Communication. In *Italian Conference on Theoretical Computer Science (ICTCS)*, 2012.

[DT13a] Giorgio Delzanno and Riccardo Traverso. Decidability and complexity results for verification of asynchronous broadcast networks. In *LATA*, pages 238–249, 2013.

[DT13b] Giorgio Delzanno and Riccardo Traverso. Specification and validation of link reversal routing via graph transformations. In *SPIN*, pages 160–177, 2013.

[EFM99] Javier Esparza, Alain Finkel, and Richard Mayr. On the verification of broadcast protocols. In *Logic in Computer Science, 1999. Proceedings. 14th Symposium on*, pages 352–359. IEEE, 1999.

[EM01] Cristian Ene and Traian Muntean. A broadcast-based calculus for communicating systems. In *IPDPS*, volume 1, page 149, 2001.

[EN98] E. A. Emerson and K. S. Namjoshi. On model checking for non-deterministic infinite-state systems. In *LICS*, pages 70–80, 1998.

[ER97] H. Ehrig and G. Rozenberg. *Handbook of Graph Grammars and Computing by Graph Transformations (Vol 1–3)*. World Scientific Publishing, 1997.

[FLP85] M. J. Fischer, N. A. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[FRSVB02] Alain Finkel, Jean-François Raskin, Mathias Samuelides, and Laurent Van Begin. Monotonic extensions of petri nets: Forward and backward search revisited. *Electr. Notes Theor. Comput. Sci.*, 68(6):85–106, 2002.

[FS01] A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1-2):63–92, 2001.

[FvGH+12a] A. Fehnker, R. J. van Glabbeek, P. Höfner, A. McIver, M. Portmann, and W. L. Tan. Automated Analysis of AODV Using UPPAAL. In *TACAS'12*, volume 7214 of *LNCS*, pages 173–187. Springer, 2012.

[FvGH+12b] A. Fehnker, R. J. van Glabbeek, P. Höfner, A. McIver, M. Portmann, and W. L. Tan. A process algebra for wireless mesh networks. In *ESOP*, pages 295–315, 2012.

[FvHM07] A. Fehnker, L. van Hoesel, and A. Mader. Modelling and verification of the lmac protocol for wireless sensor networks. In *Integrated Formal Methods*, pages 253–272. Springer, 2007.

[FW12] M. Függer and J. Widder. Efficient checking of link-reversal-based concurrent systems. In *CONCUR*, pages 486–499, 2012.

[GB81] E. Gafni and D. Bertsekas. Distributed algorithms for generating loop-free routes in networks with frequently changing topology. *IEEE Transactions on Communications*, 29:11–18, 1981.

[GdMR+12] A. H. Ghamarian, M. de Mol, A. Rensink, E. Zambon, and M. Zimakova. Modelling and analysis using groove. *STTT*, 14(1):15–40, 2012.

[God07] Jens Chr Godskesen. A calculus for mobile ad hoc networks. In *Coordination Models and Languages*, pages 132–150. Springer, 2007.

[GS92] S. M. German and A. P. Sistla. Reasoning about systems with many processes. *J. ACM*, 39(3):675–735, 1992.

[Hec06] R. Heckel. Graph transformation in a nutshell. *Electronic Notes in Theoretical Computer Science*, 148(1):187–198, 2006.

[HNSY94] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. In *Logic in Computer Science, 1992. LICS'92., Proceedings of the Seventh Annual IEEE Symposium on*, pages 394–406. IEEE, 1994.

[Hoa78] C.A.R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[JK08] S. Joshi and B. König. Applying the graph minor theorem to the verification of graph transformation systems. In *CAV*, pages 214–226, 2008.

[Kat99] J.P. Katoen. Concepts, algorithms, and tools for model checking. *Erlangen: Institut f. Mathematische Maschinen und Datenverarbeitung*, 1999.

[KF94] M. Kaminski and N. Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.

[Kön04] B. König. *Analysis and verification of systems with dynamically evolving structure*. PhD thesis, Universität Stuttgart, 2004.

[KR06] H. Kastenberg and A. Rensink. Model checking dynamic states in groove. In *SPIN*, pages 299–305, 2006.

[KVW12] I. Konnov, H. Veith, and J. Widder. Who is afraid of model checking distributed algorithms? Unpublished contribution to: CAV Workshop $(EC)^2$, 2012.

[Lad75] Richard E Ladner. The circuit value problem is log space complete for p. *ACM Sigact News*, 7(1):18–20, 1975.

[Lam98] L. Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(3):133–169, 1998.

[Lam01] Leslie Lamport. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.

[Lip76] Richard Lipton. The reachability problem requires exponential space. *Research Report 62, Department of Computer Science, Yale University, New Haven, Connecticut*, 1976.

[LNO+08] R. Lazic, T. Newcomb, J. Ouaknine, A. W. Roscoe, and J. Worrell. Nets with tokens which carry data. *Fundam. Inform.*, 88(3):251–274, 2008.

[Mer09] Massimo Merro. An observational theory for mobile ad hoc networks (full version). *Information and Computation*, 207(2):194–208, 2009.

[MMM13] K. Marzullo, A. Mei, and H. Meling. A simpler proof for paxos and fast paxos. Course notes, 2013.

[MOM02] N. Martí-Oliet and J. Meseguer. Rewriting logic: roadmap and bibliography. *Theoretical Computer Science*, 285(2):121–154, 2002.

[MS92] R. Milner and D. Sangiorgi. Barbed bisimulation. *Automata, Languages and Programming*, pages 685–695, 1992.

[MS10] Massimo Merro and Eleonora Sibilio. A timed calculus for wireless systems. In *Fundamentals of Software Engineering*, volume 5961 of *Lecture Notes in Computer Science*, pages 228–243. Springer, 2010.

[Mur89] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[NH06] Sebastian Nanz and Chris Hankin. A framework for security analysis of mobile wireless networks. *Theoretical Computer Science*, 367(1):203–227, 2006.

[Pra95] K.V.S. Prasad. A calculus of broadcasting systems. *Science of Computer Programming*, 25(2-3):285–327, 1995.

[Rac78] Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6(2):223–231, 1978.

[Ren07] A. Rensink. Isomorphism checking in groove. In A. Zündorf and D. Varró, editors, *Graph-Based Tools (GraBaTs)*, volume 1 of *Electronic Communications of the EASST*. European Association of Software Science and Technology, 2007.

[Sch10] Ph. Schnoebelen. Revisiting ackermann-hardness for lossy counter machines and reset petri nets. In *MFCS*, pages 616–628, 2010.

[SRS08] A. Singh, C. R. Ramakrishnan, and S. A. Smolka. A process calculus for mobile ad hoc networks. In *COORDINATION*, pages 296–314, 2008.

[SRS09] A. Singh, C. R. Ramakrishnan, and S. A. Smolka. Query-based model checking of ad hoc network protocols. In *CONCUR*, pages 603–619, 2009.

[SRS10] A. Singh, C. R. Ramakrishnan, and S. A. Smolka. A process calculus for mobile ad hoc networks. *Sci. Comput. Program.*, 75(6):440–469, 2010.

[Ste90] P. Stenstrom. A survey of cache coherence schemes for multiprocessors. *Computer*, 23(6):12–24, 1990.

[SWJ08] Mayank Saksena, Oskar Wibling, and Bengt Jonsson. Graph grammar modeling and verification of ad hoc routing protocols. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 18–32. Springer, 2008.

[upp] Official UPPAAL website. http://www.uppaal.org.

[vHH04] L.F.W. van Hoesel and P.J.M. Havinga. A lightweight medium access protocol (lmac) for wireless sensor networks: Reducing preamble transmissions and transceiver state switches. In *1st International Workshop on Networked Sensing Systems, INSS 2004*, pages 205–208, Tokio, Japan, 2004. Society of Instrument and Control Engineers (SICE).

[WW11] J. L. Welch and J. E. Walter. *Link Reversal Algorithms*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2011.

[YPD94] W. Yi, P. Pettersson, and M. Daniels. Automatic verification of real-time communicating systems by constraint-solving. In *Proc. of the 7th International Conference on Formal Description Techniques*, pages 223–238. Citeseer, 1994.