Safety Analysis of Parameterised Networks with Non-Blocking Rendez-Vous

Lucie Guillou
IRIF, CNRS, Université Paris Cité, France

Arnaud Sangnier
IRIF, CNRS, Université Paris Cité, France

Nathalie Sznajder
LIP6, CNRS, Sorbonne Université, France

Abstract

We consider networks of processes that all execute the same finite-state protocol and communicate via a rendez-vous mechanism. When a process requests a rendez-vous, another process can respond to it and they both change their control states accordingly. We focus here on a specific semantics, called non-blocking, where the process requesting a rendez-vous can change its state even if no process can respond to it. In this context, we study the parameterised coverability problem of a configuration, which consists in determining whether there is an initial number of processes and an execution allowing to reach a configuration bigger than a given one. We show that this problem is EXPSPACE-complete and can be solved in polynomial time if the protocol is partitioned into two sets of states, the states from which a process can request a rendez-vous and the ones from which it can answer one. We also prove that the problem of the existence of an execution bringing all the processes in a final state is undecidable in our context. These two problems can be solved in polynomial time with the classical rendez-vous semantics.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases Parameterised verification, Coverability, Counter machines

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2023.7

Related Version *Full Version*: `https://arxiv.org/abs/2307.04546` [14]

1 Introduction

Verification of distributed/concurrent systems. Because of their ubiquitous use in applications we rely on constantly, the development of formal methods to guarantee the correct behaviour of distributed/concurrent systems has become one of the most important research directions in the field of computer systems verification in the last two decades. Unfortunately, such systems are difficult to analyse for several reasons. Among others, we can highlight two aspects that make the verification process tedious. First, these systems often generate a large number of different executions due to the various interleavings generated by the concurrent behaviours of the entities involved. Understanding how these interleavings interact is a complex task which can often lead to errors at the design-level or make the model of these systems very complex. Second, in some cases, the number of participants in a distributed system may be unbounded and not known a priori. To fully guarantee the correctness of such systems, the analysis would have to be performed for all possible instances of the system, i.e., an infinite number of times. As a consequence, classical techniques to verify finite state systems, like testing or model-checking, cannot be easily adapted to distributed systems and it is often necessary to develop new techniques.

Parameterised verification. When designing systems with an unbounded number of participants, one often provides a schematic program (or protocol) intended to be implemented by multiple identical processes, parameterised by the number of participants. In general, even if the verification problem is decidable for a given instance of the parameter, verifying all possible instances is undecidable ([3]). However, several settings come into play that can be adjusted to allow automatic verification. One key aspect to obtain decidability is to assume that the processes do not manipulate identities and use simple communication mechanisms like pairwise synchronisation (or rendez-vous) [13], broadcast of a message to all the entities [10] (which can as well be lossy in order to simulate mobility [6]), shared register containing values of a finite set [11], and so on (see [9] for a survey). In every aforementioned case, all the entities execute the same protocol given by a finite state automaton. Note that parameterised verification, when decidable like in the above models, is also sometimes surprisingly easy, compared to the same problem with a fixed number of participants. For instance, liveness verification of parameterised systems with shared memory is PSPACE-complete for a fixed number of processes and in NP when parameterised [7].

Considering rendez-vous communication. In one of the seminal papers for the verification of parameterised networks [13], German and Sistla (and since then [4, 15]) assume that the entities communicate by "rendez-vous", a synchronisation mechanism in which two processes (the *sender* and the *receiver*) agree on a common action by which they jointly change their local state. This mechanism is synchronous and symmetric, meaning that if no process is ready to receive a message, the sender cannot send it. However, in some applications, such as Java Thread programming, this is not exactly the primitive that is implemented. When a Thread is suspended in a waiting state, it is woken up by the reception of a message `notify` sent by another Thread. However, the sender is not blocked if there is no suspended Thread waiting for its message; in this case, the sender sends the `notify` anyway and the message is simply lost. This is the reason why Delzanno et. al. have introduced *non-blocking* rendez-vous in [5] a communication primitive in which the sender of a message is not blocked if no process receives it. One of the problems of interest in parameterised verification is the coverability problem: is it possible that, starting from an initial configuration, (at least) one process reaches a bad state? In [5], and later in [20], the authors introduce variants of Petri nets to handle this type of communication. In particular, the authors investigate in [20] the coverability problem for an extended class of Petri nets with non-blocking arcs, and show that for this model the coverability problem is decidable using the techniques of Well-Structured Transitions Systems [1, 2, 12]. However, since their model is an extension of Petri nets, the latter problem is EXPSPACE-hard [17] (no upper bound is given). Relying on Petri nets to obtain algorithms for parameterised networks is not always a good option. In fact, the coverability problem for parameterised networks with rendez-vous is in P [13], while it is EXPSPACE-complete for Petri nets [19, 17]. Hence, no upper bound or lower bound can be directly deduced for the verification of networks with non-blocking rendez-vous from [20].

Our contributions. We show that the coverability problem for parameterised networks with *non-blocking rendez-vous communication* over a finite alphabet is EXPSPACE-complete. To obtain this result, we consider an extension of counter machines (without zero test) where we add non-blocking decrement actions and edges that can bring back the machine to its initial location at any moment. We show that the coverability problem for these extended counter machines is EXPSPACE-complete (Section 3) and that it is equivalent to our problem over parameterised networks (Section 4). We consider then a subclass of parameterised

networks – *wait-only protocols* – in which no state can allow to both request a rendez-vous and wait for one. This restriction is very natural to model concurrent programs since when a thread is waiting, it cannot perform any other action. We show that coverability problem can then be solved in polynomial time (Section 5). Finally, we show that the synchronization problem, where we look for a reachable configuration with all the processes in a given state, is undecidable in our framework, even for wait-only protocols (Section 6).

Due to lack of space, some proofs are only given in [14].

2 Rendez-vous Networks with Non-Blocking Semantics

For a finite alphabet Σ, we let Σ^* denote the set of finite sequences over Σ (or words). Given $w \in \Sigma^*$, we let $|w|$ denote its length: if $w = w_0 \dots w_{n-1} \in \Sigma^*$, then $|w| = n$. We write $\mathbb{N}$ to denote the set of natural numbers and $[i, j]$ to represent the set $\{k \in \mathbb{N} \mid i \leq k \text{ and } k \leq j\}$ for $i, j \in \mathbb{N}$. For a finite set E, the set $\mathbb{N}^E$ represents the multisets over E. For two elements $m, m' \in \mathbb{N}^E$, we denote $m + m'$ the multiset such that $(m + m')(e) = m(e) + m'(e)$ for all $e \in E$. We say that $m \leq m'$ if and only if $m(e) \leq m'(e)$ for all $e \in E$. If $m \leq m'$, then $m' - m$ is the multiset such that $(m' - m)(e) = m'(e) - m(e)$ for all $e \in E$. Given a subset $E' \subseteq E$ and $m \in \mathbb{N}^E$, we denote by $||m||_{E'}$ the sum $\Sigma_{e \in E'} m(e)$ of elements of E' present in m. The size of a multiset m is given by $||m|| = ||m||_E$. For $e \in E$, we use sometimes the notation $\wr e \wr$ for the multiset m verifying $m(e) = 1$ and $m(e') = 0$ for all $e' \in E \setminus \{e\}$ and, to represent for instance the multiset with four elements a, b, b and c, we will also use the notations $\wr a, b, b, c \wr$ or $\wr a, 2 \cdot b, c \wr$.

2.1 Rendez-Vous Protocols

We can now define our model of networks. We assume that all processes in the network follow the same protocol. Communication in the network is pairwise and is performed by *rendez-vous* through a finite communication alphabet Σ. Each process can either perform an internal action using the primitive τ, or request a rendez-vous by sending the message m using the primitive $!m$ or answer to a rendez-vous by receiving the message m using the primitive $?m$ (for $m \in \Sigma$). Thus, the set of primitives used by our protocols is $RV(\Sigma) = \{\tau\} \cup \{?m, !m \mid m \in \Sigma\}$.

▶ **Definition 2.1** (Rendez-vous protocol). *A* rendez-vous protocol *(shortly protocol) is a tuple* $\mathcal{P} = (Q, \Sigma, q_{in}, q_f, T)$ *where* Q *is a finite set of states,* Σ *is a finite alphabet,* $q_{in} \in Q$ *is the initial state,* $q_f \in Q$ *is the final state and* $T \subseteq Q \times RV(\Sigma) \times Q$ *is the finite set of transitions.*

For a message $m \in \Sigma$, we denote by $R(m)$ the set of states q from which the message m can be received, i.e. states q such that there is a transition $(q, ?m, q') \in T$ for some $q' \in Q$.

A *configuration* associated to the protocol $\mathcal{P}$ is a non-empty multiset C over Q for which $C(q)$ denotes the number of processes in the state q and $||C||$ denotes the total number of processes in the configuration C. A configuration C is said to be *initial* if and only if $C(q) = 0$ for all $q \in Q \setminus \{q_{in}\}$. We denote by $\mathcal{C}(\mathcal{P})$ the set of configurations and by $\mathcal{I}(\mathcal{P})$ the set of initial configurations. Finally for $n \in \mathbb{N} \setminus \{0\}$, we use the notation $\mathcal{C}_n(\mathcal{P})$ to represent the set of configurations of size n, i.e. $\mathcal{C}_n(\mathcal{P}) = \{C \in \mathcal{C}(\mathcal{P}) \mid ||C|| = n\}$. When the protocol is made clear from the context, we shall write $\mathcal{C}$, $\mathcal{I}$ and $\mathcal{C}_n$.

We explain now the semantics associated with a protocol. For this matter we define the relation $\to_{\mathcal{P}} \subseteq \bigcup_{n \geq 1} \mathcal{C}_n \times \big(\{\tau\} \cup \Sigma \cup \{\mathbf{nb}(m) \mid m \in \Sigma\}\big) \times \mathcal{C}_n$ as follows (here $\mathbf{nb}(\cdot)$ is a special symbol). Given $n \in \mathbb{N} \setminus \{0\}$ and $C, C' \in \mathcal{C}_n$ and $m \in \Sigma$, we have:

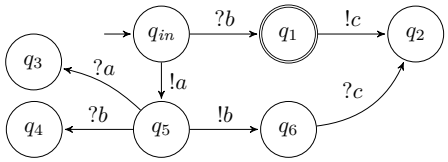

Figure 1 Example of a rendez-vous protocol $\mathcal{P}$.

1. $C \xrightarrow{\tau}_{\mathcal{P}} C'$ iff there exists $(q, \tau, q') \in T$ such that $C(q) > 0$ and $C' = C - \wr q \wr + \wr q' \wr$ **(internal)**;
2. $C \xrightarrow{m}_{\mathcal{P}} C'$ iff there exists $(q_1, !m, q_1') \in T$ and $(q_2, ?m, q_2') \in T$ such that $C(q_1) > 0$ and $C(q_2) > 0$ and $C(q_1) + C(q_2) \geq 2$ (needed when $q_1 = q_2$) and $C' = C - \wr q_1, q_2 \wr + \wr q_1', q_2' \wr$ **(rendez-vous)**;
3. $C \xrightarrow{\mathbf{nb}(m)}_{\mathcal{P}} C'$ iff there exists $(q_1, !m, q_1') \in T$, such that $C(q_1) > 0$ and $(C - \wr q_1 \wr)(q_2) = 0$ for all $(q_2, ?m, q_2') \in T$ and $C' = C - \wr q_1 \wr + \wr q_1' \wr$ **(non-blocking request)**.

Intuitively, from a configuration C, we allow the following behaviours: either a process takes an internal transition (labeled by τ), or two processes synchronize over a rendez-vous m, or a process requests a rendez-vous to which no process can answer (non-blocking sending).

This allows us to define $S_{\mathcal{P}}$ the transition system $(\mathcal{C}(\mathcal{P}), \to_{\mathcal{P}})$ associated to $\mathcal{P}$. We will write $C \to_{\mathcal{P}} C'$ when there exists $a \in \{\tau\} \cup \Sigma \cup \{\mathbf{nb}(m) \mid m \in \Sigma\}$ such that $C \xrightarrow{a}_{\mathcal{P}} C'$ and denote by $\to_{\mathcal{P}}^*$ the reflexive and transitive closure of $\to_{\mathcal{P}}$. Furthermore, when made clear from the context, we might simply write $\to$ instead of $\to_{\mathcal{P}}$. An *execution* is a finite sequence of configurations $\rho = C_0 C_1 \dots$ such that, for all $0 \leq i < |\rho|$, $C_i \to_{\mathcal{P}} C_{i+1}$. The execution is said to be initial if $C_0 \in \mathcal{I}(\mathcal{P})$.

▶ **Example 2.2.** Figure 1 provides an example of a rendez-vous protocol where q_{in} is the initial state and q_1 the final state. A configuration associated to this protocol is for instance the multiset $\wr 2 \cdot q_1, 1 \cdot q_4, 1 \cdot q_5 \wr$ and the following sequence represents an initial execution: $\wr 2 \cdot q_{in} \wr \xrightarrow{\mathbf{nb}(a)} \wr q_{in}, q_5 \wr \xrightarrow{b} \wr q_1, q_6 \wr \xrightarrow{c} \wr 2 \cdot q_2 \wr$.

▶ Remark 2.3. When we only allow behaviours of type **(internal)** and **(rendez-vous)**, this semantics corresponds to the classical rendez-vous semantics ([13, 4, 15]). In opposition, we will refer to the semantics defined here as the *non-blocking semantics* where a process is not *blocked* if it requests a rendez-vous and no process can answer to it. Note that all behaviours possible in the classical rendez-vous semantics are as well possible in the non-blocking semantics but the converse is false.

2.2 Verification Problems

We now present the problems studied in this work. For this matter, given a protocol $\mathcal{P} = (Q, \Sigma, q_{in}, q_f, T)$, we define two sets of final configurations. The first one $\mathcal{F}_\exists(\mathcal{P}) = \{C \in \mathcal{C}(\mathcal{P}) \mid C(q_f) > 0\}$ characterises the configurations where one of the processes is in the final state. The second one $\mathcal{F}_\forall(\mathcal{P}) = \{C \in \mathcal{C}(\mathcal{P}) \mid C(Q \setminus \{q_f\}) = 0\}$ represents the configurations where all the processes are in the final state. Here again, when the protocol is clear from the context, we might use the notations $\mathcal{F}_\exists$ and $\mathcal{F}_\forall$. We study three problems: the *state coverability problem* (SCover), the *configuration coverability* problem (CCover) and the *synchronization problem* (Synchro), which all take as input a protocol $\mathcal{P}$ and can be stated as follows:

Problem name	Question
SCover	Are there $C_0 \in \mathcal{I}$ and $C_f \in \mathcal{F}_\exists$, such that $C_0 \to^* C_f$?
CCover	Given $C \in \mathcal{C}$, are there $C_0 \in \mathcal{I}$ and $C' \geq C$, such that $C_0 \to^* C'$?
Synchro	Are there $C_0 \in \mathcal{I}$ and $C_f \in \mathcal{F}_\forall$, such that $C_0 \to^* C_f$?

▶ Remark 2.4. The difficulty in solving these problems lies in the fact that we are seeking for an initial configuration allowing a specific execution but the set of initial configurations is infinite.

The difference between SCover and Synchro is that in the first one we ask for at least one process to end up in the final state whereas the second one requires all the processes to end in this state. Note that SCover is an instance of CCover but Synchro is not.

Observe that SCover should be seen as a safety property: if q_f is an error state and the answer is negative, then for any number of processes, no process will ever be in that error state.

▶ **Example 2.5.** The rendez-vous protocol of Figure 1 is a positive instance of SCover, as shown in Example 2.2. However, this is not the case for Synchro: if an execution brings a process in q_2, this process cannot be brought afterwards to q_1. If q_2 is the final state, $\mathcal{P}$ is now a positive instance of Synchro (see Example 2.2). Note that if the final state is q_4, $\mathcal{P}$ is not a positive instance of SCover anymore. In fact, the only way to reach a configuration with a process in q_4 is to put (at least) two processes in state q_5 as this is the only state from which one process can send the message b. However, this cannot happen, since from an initial configuration, the only available action consists in sending the message a as a non-blocking request. Once there is one process in state q_5, any other attempt to put another process in this state will induce a reception of message a by the process already in q_5, which will hence leave q_5. Finally, note that for any $n \in \mathbb{N}$, the configuration $\wr n \cdot q_3 \wr$ is coverable, even if $\mathcal{P}$ with q_3 as final state is not a positive instance of Synchro.

3 Coverability for Non-Blocking Counter Machines

We first detour into new classes of counter machines, which we call *non-blocking counter machines* and *non-blocking counter machines with restore*, in which a new way of decrementing the counters is added to the classical one: a non-blocking decrement, which is an action that can always be performed. If the counter is strictly positive, it is decremented; otherwise it is let to 0. We show that the coverability of a control state in this model is Expspace-complete, and use this result to solve coverability problems in rendez-vous protocols.

To define counter machines, given a set of integer variables (also called counters) X, we use the notation $\mathsf{CAct}(X)$ to represent the set of associated actions given by $\{\mathtt{x}+, \mathtt{x}-, \mathtt{x}=0 \mid \mathtt{x} \in X\} \cup \{\bot\}$. Intuitively, $\mathtt{x}+$ increments the value of the counter $\mathtt{x}$, while $\mathtt{x}-$ decrements it and $\mathtt{x}=0$ checks if it is equal to 0. We are now ready to state the syntax of this model.

▶ **Definition 3.1.** *A* counter machine *(shortly CM) is a tuple $M = (Loc, X, \Delta, \ell_{in})$ such that Loc is a finite set of locations, $\ell_{in} \in Loc$ is an initial location, X is a finite set of counters, and $\Delta \subseteq Loc \times \mathsf{CAct}(X) \times Loc$ is finite set of transitions.*

We will say that a CM is test-free (shortly test-free CM) whenever $\Delta \cap \mathrm{Loc} \times \{\mathtt{x}=0 \mid \mathtt{x} \in X\} \times \mathrm{Loc} = \emptyset$. A configuration of a CM $M = (\mathrm{Loc}, X, \Delta, \ell_{in})$ is a pair (ℓ, v) where $\ell \in \mathrm{Loc}$ specifies the current location of the CM and $v \in \mathbb{N}^X$ associates to each counter a natural value. The size of a CM M is given by $|M| = |\mathrm{Loc}| + |X| + |\Delta|$. Given two configurations (ℓ, v) and (ℓ', v') and a transition $\delta \in \Delta$, we define $(\ell, v) \overset{\delta}{\rightsquigarrow}_M (\ell', v')$ if and only if $\delta = (\ell, op, \ell')$ and one of the following holds:

- $op = \bot$ and $v = v'$;
- $op = \mathtt{x}+$ and $v'(\mathtt{x}) = v(\mathtt{x}) + 1$ and $v'(\mathtt{x}') = v(\mathtt{x}')$ for all $\mathtt{x}' \in X \setminus \{\mathtt{x}\}$;
- $op = \mathtt{x}-$ and $v'(\mathtt{x}) = v(\mathtt{x}) - 1$ and $v'(\mathtt{x}') = v(\mathtt{x}')$ for all $\mathtt{x}' \in X \setminus \{\mathtt{x}\}$;
- $op = \mathtt{x}{=}0$ and $v(\mathtt{x}) = 0$ and $v' = v$.

In order to simulate the non-blocking semantics of our rendez-vous protocols with counter machines, we extend the class of test-free CM with non-blocking decrement actions.

▶ **Definition 3.2.** *A* non-blocking test-free counter machine *(shortly* NB-CM*) is a tuple* $M = (Loc, X, \Delta_b, \Delta_{nb}, \ell_{in})$ *such that* $(Loc, X, \Delta_b, \ell_{in})$ *is a test-free CM and* $\Delta_{nb} \subseteq Loc \times \{nb(\mathtt{x}-) \mid \mathtt{x} \in X\} \times Loc$ *is a finite set of* non-blocking *transitions.*

Observe that in a NB-CM, both blocking and non-blocking decrements are possible, depending on the type of transition taken. Again, a configuration is given by a pair $(\ell, v) \in \mathrm{Loc} \times \mathbb{N}^X$. Given two configurations (ℓ, v) and (ℓ, v') and $\delta \in \Delta_b \cup \Delta_{nb}$, we extend the transition relation $(\ell, v) \overset{\delta}{\rightsquigarrow}_M (\ell, v')$ over the set Δ_{nb} in the following way: for $\delta = (\ell, nb(\mathtt{x}-), \ell') \in \Delta_{nb}$, we have $(\ell, v) \overset{\delta}{\rightsquigarrow}_M (\ell', v')$ if and only if $v'(\mathtt{x}) = \max(0, v(\mathtt{x}) - 1)$, and $v'(\mathtt{x}') = v(\mathtt{x}')$ for all $\mathtt{x}' \in X \setminus \{\mathtt{x}\}$.

We say that M is an NB-CM *with restore* (shortly NB-R-CM) when $(\ell, \bot, \ell_{in}) \in \Delta$ for all $\ell \in \mathrm{Loc}$, i.e. from each location, there is a transition leading to the initial location with no effect on the counters values.

For a CM M with set of transitions Δ (resp. an NB-CM with sets of transitions Δ_b and Δ_{nb}), we will write $(\ell, v) \rightsquigarrow_M (\ell', v')$ whenever there exists $\delta \in \Delta$ (resp. $\delta \in \Delta_b \cup \Delta_{nb}$) such that $(\ell, v) \overset{\delta}{\rightsquigarrow}_M (\ell', v')$ and use $\rightsquigarrow_M^*$ to represent the reflexive and transitive closure of $\rightsquigarrow_M$. When the context is clear we shall write $\rightsquigarrow$ instead of $\rightsquigarrow_M$. We let $\mathbf{0}_X$ be the valuation such that $\mathbf{0}_X(\mathtt{x}) = 0$ for all $\mathtt{x} \in X$. An execution is a finite sequence of configurations $(\ell_0, v_0) \rightsquigarrow (\ell_1, v_1) \rightsquigarrow \ldots \rightsquigarrow (\ell_k, v_k)$. It is said to be initial if $(\ell_0, v_0) = (\ell_{in}, \mathbf{0}_X)$. A configuration (ℓ, v) is called reachable if $(\ell_{in}, \mathbf{0}_X) \rightsquigarrow^* (\ell, v)$.

We shall now define the coverability problem for (non-blocking test-free) counter machines, which asks whether a given location can be reached from the initial configuration. We denote this problem Cover$[\mathcal{M}]$, for $\mathcal{M} \in \{\text{CM}, \text{test-free CM}, \text{NB-CM}, \text{NB-R-CM}\}$. It takes as input a machine M in $\mathcal{M}$ (with initial location ℓ_{in} and working over a set X of counters) and a location ℓ_f and it checks whether there is a valuation $v \in \mathbb{N}^X$ such that $(\ell_{in}, \mathbf{0}_X) \rightsquigarrow^* (\ell_f, v)$.

In the rest of this section, we will prove that Cover[NB-R-CM] is Expspace-complete. To this end, we first establish that Cover[NB-CM] is in Expspace, by an adaptation of Rackoff's proof which shows that coverability in Vector Addition Systems is in Expspace [19]. This gives also the upper bound for NB-R-CM, since any NB-R-CM is a NB-CM. This result is established by the following theorem, whose proof is omitted due to lack of space.

▶ **Theorem 3.3.** *Cover*[NB-CM] *and Cover*[NB-R-CM] *are in Expspace.*

To obtain the lower bound, inspired by Lipton's proof showing that coverability in Vector Addition Systems is Expspace-hard [8, 17], we rely on 2Exp-bounded test-free CM. We say that a CM $M = (\mathrm{Loc}, X, \Delta, \ell_{in})$ is 2Exp-*bounded* if there exists $n \in O(|M|)$ such that any reachable configuration (ℓ, v) satisfies $v(\mathtt{x}) \leq 2^{2^n}$ for all $\mathtt{x} \in X$. We use then the following result.

▶ **Theorem 3.4** ([8, 17])**.** *Cover[2Exp-bounded test-free CM] is Expspace-hard.*

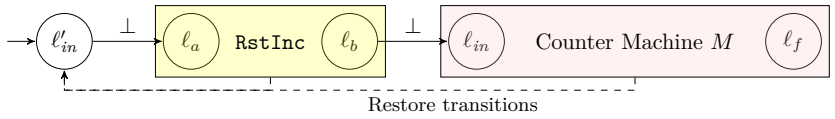

Figure 2 The NB-R-CM N.

We now show how to simulate a 2Exp-bounded test-free CM by a NB-R-CM, by carefully handling restore transitions that may occur at any point in the execution. We will ensure that each restore transition is followed by a reset of the counters, so that we can always extract from an execution of the NB-R-CM a correct initial execution of the original test-free CM. The way we enforce resetting of the counters is inspired by the way Lipton simulates 0-tests of a CM in a test-free CM. As in [17, 8], we will describe the final NB-R-CM by means of several submachines. To this end, we define *procedural non-blocking counter machines* that are NB-CM with several identified *output states*: formally, a procedural-NB-CM is a tuple $N = (\mathrm{Loc}, X, \Delta_b, \Delta_{nb}, \ell_{in}, L_{out})$ such that $(\mathrm{Loc}, X, \Delta_b, \Delta_{nb}, \ell_{in})$ is a NB-CM, $L_{out} \subseteq \mathrm{Loc}$, and there is no outgoing transition from states in L_{out}.

Now fix a 2Exp-bounded test-free CM $M = (\mathrm{Loc}, X, \Delta, \ell_{in})$, $\ell_f \in \mathrm{Loc}$ the location to be covered. There is some c, such that, any reachable configuration (ℓ, v) satisfies $v(\mathtt{x}) < 2^{2^{c|M|}}$ for all $\mathtt{x} \in X$, fix $n = c|M|$. We build a NB-R-CM N as pictured in Figure 2. The goal of the procedural NB-CM `RstInc` is to ensure that all counters in X are reset. Hence, after each restore transition, we are sure that we start over a fresh execution of the test-free CM M. We will need the mechanism designed by Lipton to test whether a counter is equal to 0. So, we define two families of sets of counters $(Y_i)_{0\leq i\leq n}$ and $(\overline{Y_i})_{0\leq i\leq n}$ as follows. Let $Y_i = \{\mathtt{y}_i, \mathtt{z}_i, \mathtt{s}_i\}$ and $\overline{Y}_i = \{\overline{\mathtt{y}}_i, \overline{\mathtt{z}}_i, \overline{\mathtt{s}}_i\}$ for all $0 \leq i < n$ and $Y_n = X$ and $\overline{Y}_n = \emptyset$ and $X' = \bigcup_{0\leq i\leq n} Y_i \cup \overline{Y}_i$. All the machines we will describe from now on will work over the set of counters X'.

Procedural-NB-CM $\mathtt{TestSwap}_i(\mathtt{x})$. We use a family of procedural-NB-CM defined in [17, 8]: for all $0 \leq i < n$, for all $\overline{\mathtt{x}} \in \overline{Y}_i$, $\mathtt{TestSwap}_i(\overline{\mathtt{x}})$ is a procedural-NB-CM with an initial location $\ell_{in}^{\mathtt{TS},i,\mathtt{x}}$, and two output locations $\ell_z^{\mathtt{TS},i,\mathtt{x}}$ and $\ell_{nz}^{\mathtt{TS},i,\mathtt{x}}$. It tests if the value of $\overline{\mathtt{x}}$ is equal to 0, using the fact that the sum of the values of $\mathtt{x}$ and $\overline{\mathtt{x}}$ is equal to 2^{2^i}. If $\overline{\mathtt{x}} = 0$, it swaps the values of $\mathtt{x}$ and $\overline{\mathtt{x}}$, and the execution ends in the output location $\ell_z^{\mathtt{TS},i,\mathtt{x}}$. Otherwise, counters values are left unchanged and the execution ends in $\ell_{nz}^{\mathtt{TS},i,\mathtt{x}}$. In any case, other counters are not modified by the execution. Note that $\mathtt{TestSwap}_i(\mathtt{x})$ makes use of variables in $\bigcup_{1\leq j<i} Y_i \cup \overline{Y}_i$.

Procedural NB-CM $\mathtt{Rst}_i$. We use these machines to define a family of procedural-NB-CM called $(\mathtt{Rst}_i)_{0\leq i\leq n}$ that reset the counters in $Y_i \cup \overline{Y_i}$, assuming that their values are less than or equal to 2^{2^i}. Let $0 \leq i \leq n$, we let $\mathtt{Rst}_i = (\mathrm{Loc}^{\mathtt{R},i}, X', \Delta_b^{\mathtt{R},i}, \Delta_{nb}^{\mathtt{R},i}, \ell_{in}^{\mathtt{R},i}, \{\ell_{out}^{\mathtt{R},i}\})$. The machine $\mathtt{Rst}_0$ is pictured Figure 3. For all $0 \leq i < n$, the machine $\mathtt{Rst}_{i+1}$ uses counters from $Y_i \cup \overline{Y_i}$ and procedural-NB-CM $\mathtt{Testswap}_i(\overline{\mathtt{z}}_i)$ and $\mathtt{Testswap}_i(\overline{\mathtt{y}}_i)$ to control the number of times variables from Y_{i+1} and $\overline{Y}_{i+1}$ are decremented. It is pictured Figure 4. Observe that since $Y_n = X$, and $\overline{Y_n} = \emptyset$, the machine $\mathtt{Rst}_n$ will be a bit different from the picture: there will only be non-blocking decrements over counters from Y_n, that is over counters X from the initial test-free CM M. If $\overline{\mathtt{y}}_i$, $\overline{\mathtt{z}}_i$ (and $\overline{\mathtt{s}}_i$) are set to 2^{2^i} and $\mathtt{y}_i$, $\mathtt{z}_i$ (and $\mathtt{s}_i$) are set to 0, then each time this procedural-NB-CM takes an outer loop, the variables of $Y_{i+1} \cup \overline{Y}_{i+1}$ are decremented (in a non-blocking fashion) 2^{2^i} times. This is ensured by the properties of $\mathtt{TestSwap}_i(\mathtt{x})$. Moreover, the location $\ell_z^{\mathtt{TS},i,\mathtt{y}}$ will only be reached when the counter $\overline{\mathtt{y}}_i$

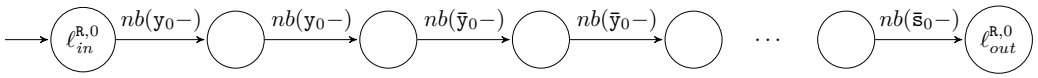

Figure 3 Description of $\mathtt{Rst}_0$.

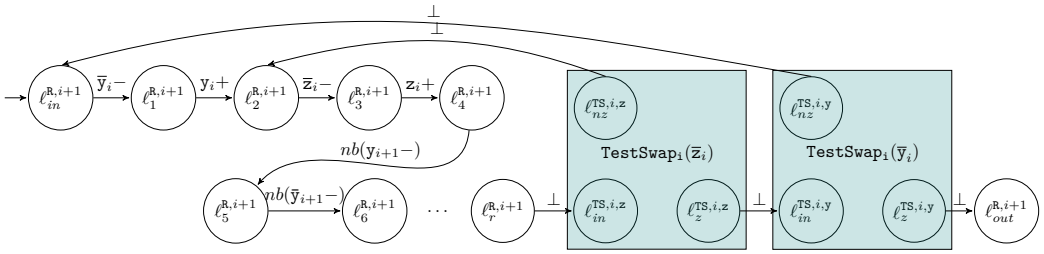

Figure 4 Description of $\mathtt{Rst}_{i+1}$.

is set to 0, and this will happen after 2^{2^i} iterations of the outer loop, again thanks to the properties of $\mathtt{TestSwap}_i(\mathtt{x})$. So, all in all, variables from Y_i and $\overline{Y}_{i+1}$ will take a non-blocking decrement $2^{2^i}.2^{2^i}$ times, that is $2^{2^{i+1}}$.

For all $\mathtt{x} \in X'$, we say that $\mathtt{x}$ is *initialized* in a valuation v if $\mathtt{x} \in Y_i$ for some $0 \leq i \leq n$ and $v(\mathtt{x}) = 0$, or $\mathtt{x} \in \overline{Y}_i$ for some $0 \leq i \leq n$ and $v(\mathtt{x}) = 2^{2^i}$. For $0 \leq i \leq n$, we say that a valuation $v \in \mathbb{N}^{X'}$ is *i-bounded* if for all $\mathtt{x} \in Y_i \cup \overline{Y}_i$, $v(\mathtt{x}) \leq 2^{2^i}$.

The construction ensures that when one enters $\mathtt{Rst}_i$ with a valuation v that is i-bounded, and in which all variables in $\bigcup_{0 \leq j < i} Y_j \cup \overline{Y}_j$ are initialized, the location $\ell_{out}^{\mathtt{R},i}$ is reached with a valuation v' such that: $v'(\mathtt{x}) = 0$ for all $\mathtt{x} \in Y_i \cup \overline{Y}_i$ and $v'(\mathtt{x}) = v(\mathtt{x})$ for all $\mathtt{x} \notin Y_i \cup \overline{Y}_i$. Moreover, if v is j-bounded for all $0 \leq j \leq n$, then any valuation reached during the execution remains j-bounded for all $0 \leq j \leq n$.

Procedural NB-CM $\mathtt{Inc}_i$**.** The properties we seek for $\mathtt{Rst}_i$ are ensured whenever the variables in $\bigcup_{0 \leq j < i} Y_j \cup \overline{Y}_j$ are initialized. This is taken care of by a family of procedural-NB-CM introduced in [17, 8]. For all $0 \leq i < n$, $\mathtt{Inc}_i$ is a procedural-NB-CM with initial location $\ell_{in}^{\mathtt{Inc},i}$, and unique output location $\ell_{out}^{\mathtt{Inc},i}$. They enjoy the following property: for $0 \leq i < n$, when one enters $\mathtt{Inc}_i$ with a valuation v in which all the variables in $\bigcup_{0 \leq j < i} Y_j \cup \overline{Y}_j$ are initialized and $v(\mathtt{x}) = 0$ for all $\mathtt{x} \in \overline{Y}_i$, then the location $\ell_{out}^{\mathtt{Inc}_i}$ is reached with a valuation v' such that $v'(\mathtt{x}) = 2^{2^i}$ for all $\mathtt{x} \in \overline{Y}_i$, and $v'(\mathtt{x}) = v(\mathtt{x})$ for all other $\mathtt{x} \in X'$. Moreover, if v is j-bounded for all $0 \leq j \leq n$, then any valuation reached during the execution remains j-bounded for all $0 \leq j \leq n$.

Procedural NB-CM $\mathtt{RstInc}$**.** Finally, let $\mathtt{RstInc}$ be a procedural-NB-CM with initial location ℓ_a and output location ℓ_b, over the set of counters X' and built as an alternation of $\mathtt{Rst}_i$ and $\mathtt{Inc}_i$ for $0 \leq i < n$, finished by $\mathtt{Rst}_n$. It is depicted in Figure 5. Thanks to the properties of the machines $\mathtt{Rst}_i$ and $\mathtt{Inc}_i$, in the output location of each $\mathtt{Inc}_i$ machine, the counters in $\overline{Y}_i$ are set to 2^{2^i}, which allow counters in $Y_{i+1} \cup \overline{Y}_{i+1}$ to be set to 0 in the output location of $\mathtt{Rst}_{i+1}$. Hence, in location $\ell_{out}^{\mathtt{Inc},n}$, counters in $Y_n = X$ are set to 0.

From [17, 8], each procedural machine $\mathtt{TestSwap}_i(\mathtt{x})$ and $\mathtt{Inc}_i$ has size at most $C \times n^2$ for some constant C. Hence, observe that N is of size at most B for some $B \in O(|M|^3)$. One can show that $(\ell_{in}, \mathbf{0}_X) \rightsquigarrow_M^* (\ell_f, v)$ for some $v \in \mathbb{N}^X$, if and only if $(\ell'_{in}, \mathbf{0}_{X'}) \rightsquigarrow_N^* (\ell_f, v')$ for some $v' \in \mathbb{N}^{X'}$. Using Theorem 3.4, we obtain:

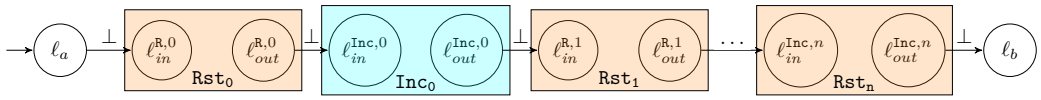

Figure 5 `RstInc`.

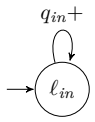

Figure 6 Incrementing q_{in}.

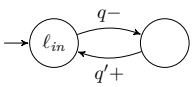

Figure 7 Transitions for $(q, \tau, q') \in T$.

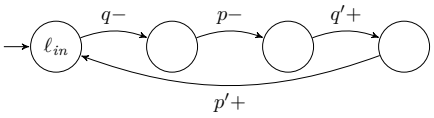

Figure 8 Transitions for a rendez-vous $(q, !a, q'), (p, ?a, p') \in T$.

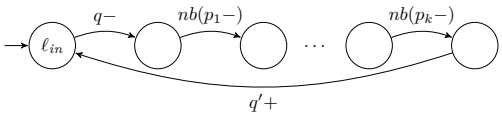

Figure 9 Transitions for a non-blocking sending $(q, !a, q') \in T$ and $R(a) = \{p_1 \dots p_k\}$.

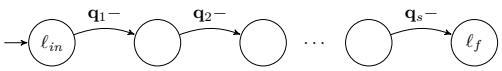

Figure 10 Verification for the coverability of $C_F = \wr \mathbf{q}_1 \wr + \wr \mathbf{q}_2 \wr + \cdots + \wr \mathbf{q}_s \wr$.

▶ **Theorem 3.5.** *Cover[NB-R-CM] is Expspace-hard.*

4 Coverability for Rendez-Vous Protocols

In this section we prove that SCover and CCover problems are both Expspace-complete for rendez-vous protocols. To this end, we present the following reductions: CCover reduces to Cover[NB-CM] and Cover[NB-R-CM] reduces to SCover. This will prove that CCover is in Expspace and SCover is Expspace-hard (from Theorem 3.3 and Theorem 3.5). As SCover is an instance of CCover, the two reductions suffice to prove Expspace-completeness for both problems.

4.1 From Rendez-vous Protocols to NB-CM

Let $\mathcal{P} = (Q, \Sigma, q_{in}, q_f, T)$ a rendez-vous protocol and C_F a configuration of $\mathcal{P}$ to be covered. We shall also decompose C_F as a sum of multisets $\wr \mathbf{q}_1 \wr + \wr \mathbf{q}_2 \wr + \cdots + \wr \mathbf{q}_s \wr$. Observe that there might be $\mathbf{q}_i = \mathbf{q}_j$ for $i \neq j$. We build the NB-CM $M = (\mathrm{Loc}, X, \Delta_b, \Delta_{nb}, \ell_{in})$ with $X = Q$. A configuration C of $\mathcal{P}$ is meant to be represented in M by (ℓ_{in}, v), with $v(q) = C(q)$ for all $q \in Q$. The only meaningful location of M is then ℓ_{in}. The other ones are here to ensure correct updates of the counters when simulating a transition. We let $\mathrm{Loc} = \{\ell_{in}\} \cup \{\ell^1_{(t,t')}, \ell^2_{(t,t')}, \ell^3_{(t,t')} \mid t = (q, !a, q'), t' = (p, ?a, p') \in T\} \cup \{\ell_t, \ell^a_{t,p_1}, \cdots, \ell^a_{t,p_k} \mid t = (q, !a, q') \in T, R(a) = \{p_1, \dots, p_k\}\} \cup \{\ell_q \mid t = (q, \tau, q') \in T\} \cup \{\ell_1 \dots \ell_s\}$, with final location $\ell_f = \ell_s$, where $R(m)$ for a message $m \in \Sigma$ has been defined in Section 2. The sets Δ_b and Δ_{nb} are shown Figures 6–10. Transitions pictured Figures 6–8 and 10 show how to simulate a rendez-vous protocol with the classical rendez-vous mechanism. The non-blocking rendez-vous are handled by the transitions pictured Figure 9. If the NB-CM M faithfully simulates $\mathcal{P}$, then this loop of non-blocking decrements is taken when the values of the counters in $R(a)$ are equal to 0, and the configuration reached still corresponds to a configuration in $\mathcal{P}$. However, it could be that this loop is taken in M while some counters in $R(a)$ are strictly positive. In this case, a blocking rendez-vous has to be taken in $\mathcal{P}$, e.g. $(q, !a, q')$ and $(p, ?a, p')$ if the counter p in M is strictly positive. Therefore, the value of the

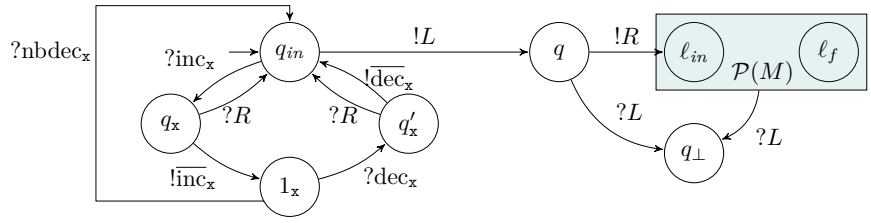

Figure 11 The rendez-vous protocol $\mathcal{P}$ built from the NB-R-CM M. Note that there is one gadget with states $\{q_x, q'_x, 1_x\}$ for each counter $x \in X$.

reached configuration (ℓ_{in}, v) and the corresponding configuration C in $\mathcal{P}$ will be different: first, $C(p') > v(p')$, since the process in p has moved in the state p' in $\mathcal{P}$ when there has been no increment of p' in M. Furthermore, all other non-blocking decrements of counters in $R(a)$ in M may have effectively decremented the counters, when in $\mathcal{P}$ no other process has left a state of $R(a)$. However, this ensures that $C \geq v$. The reduction then guarantees that if (ℓ_{in}, v) is reachable in M, then a configuration $C \geq v$ is reachable in $\mathcal{P}$. Then, if it is possible to reach a configuration (ℓ_{in}, v) in M whose counters are high enough to cover ℓ_F, then the corresponding initial execution in $\mathcal{P}$ will reach a configuration $C \geq v$, which hence covers C_F.

▶ **Theorem 4.1.** *CCover over rendez-vous protocols is in Expspace.*

4.2 From NB-R-CM to Rendez-Vous Protocols

The reduction from Cover[NB-R-CM] to SCover in rendez-vous protocols mainly relies on the mechanism that can ensure that at most one process evolves in some given set of states, as explained in Example 2.5. This will allow to somehow select a "leader" among the processes that will simulate the behaviour of the NB-R-CM whereas other processes will simulate the values of the counters. Let $M = (\text{Loc}, X, \Delta_b, \Delta_{nb}, \ell_{in})$ a NB-R-CM and $\ell_f \in \text{Loc}$ a final target location. We build the rendez-vous protocol $\mathcal{P}$ pictured in Figure 11, where $\mathcal{P}(M)$ is the part that will simulate the NB-R-CM M. The locations $\{1_x \mid x \in X\}$ will allow to encode the values of the different counters during the execution: for a configuration C, $C(1_x)$ will represent the value of the counter x. We give then $\mathcal{P}(M) = (Q_M, \Sigma_M, \ell_{in}, \ell_f, T_M)$ with $Q_M = \text{Loc} \cup \{\ell_\delta \mid \delta \in \Delta_b\}$, $\Sigma_M = \{\text{inc}_x, \overline{\text{inc}}_x, \text{dec}_x, \overline{\text{dec}}_x, \text{nbdec}_x \mid x \in X\}$, and $T_M = \{(\ell_i, !\text{inc}_x, \ell_\delta), (\ell_\delta, ?\overline{\text{inc}}_x, \ell_j) \mid \delta = (\ell_i, x+, \ell_j) \in \Delta_b\} \cup \{(\ell_i, !\text{dec}_x, \ell_\delta), (\ell_\delta, ?\overline{\text{dec}}_x, \ell_j) \mid \delta = (\ell_i, x-, \ell_j) \in \Delta_b\} \cup \{(\ell_i, !\text{nbdec}_x, \ell_j) \mid (\ell_i, nb(x-), \ell_j) \in \Delta_{nb}\} \cup \{(\ell_i, \tau, \ell_j) \mid (\ell_i, \perp, \ell_j) \in \Delta_b\}$. Here, the reception of a message $\overline{\text{inc}}_x$ (respectively $\overline{\text{dec}}_x$) works as an acknowledgement, ensuring that a process has indeed received the message inc_x (respectively dec_x), and that the corresponding counter has been incremented (resp. decremented). For non-blocking decrement, obviously no acknowledgement is required. The protocol $\mathcal{P} = (Q, \Sigma, q_{in}, \ell_f, T)$ is then defined with $Q = Q_M \cup \{1_x, q_x, q'_x \mid x \in X\} \cup \{q_{in}, q, q_\perp\}$, $\Sigma = \Sigma_M \cup \{L, R\}$ and T is the set of transitions T_M along with the transitions pictured in Figure 11. Note that there is a transition $(\ell, ?L, q_\perp)$ for all $\ell \in Q_M$.

With two non-blocking transitions on L and R at the beginning, protocol $\mathcal{P}$ can faithfully simulate the NB-R-CM M without further ado, provided that the initial configuration contains enough processes to simulate all the counters values during the execution: after having sent a process in state ℓ_{in}, any transition of M can be simulated in $\mathcal{P}$. Conversely, an initial execution of $\mathcal{P}$ can send multiple processes into the $\mathcal{P}(M)$ zone, which can mess up the simulation. However, each new process entering $\mathcal{P}(M)$ will first send the message

L, and as a consequence the process already in $\{q\} \cup Q_M$, if any, will move to the deadlock state $q_\perp$ receiving this message, and then the new process will send the message R, which will be received by some process in $\{q_{\mathtt{x}}, q'_{\mathtt{x}} \mid \mathtt{x} \in X\}$, if any. Moreover, the construction of the protocol ensures that there can only be one process in the set of states $\{q_{\mathtt{x}}, q'_{\mathtt{x}} \mid \mathtt{x} \in X\}$. Then, if we have reached a configuration simulating the configuration (ℓ, v) of M, sending a new process in the $\mathcal{P}(M)$ zone will lead to a configuration (ℓ_{in}, v), and hence simply mimicks a restore transition of M. So every initial execution of $\mathcal{P}$ corresponds to an initial execution of M.

▶ **Theorem 4.2.** *SCover and CCover over rendez-vous protocols are Expspace complete.*

5 Coverability for Wait-Only Protocols

In this section, we study a restriction on rendez-vous protocols in which we assume that a process waiting to answer a rendez-vous cannot perform another action by itself. This allows for a polynomial time algorithm for solving CCover.

5.1 Wait–Only Protocols

We say that a protocol $\mathcal{P} = (Q, \Sigma, q_{in}, q_f, T)$ is *wait-only* if the set of states Q can be partitioned into Q_A — the *active states* — and Q_W — the *waiting* states — with $q_{in} \in Q_A$ and:

- for all $q \in Q_A$, for all $(q', ?m, q'') \in T$, we have $q' \neq q$;
- for all $q \in Q_W$, for all $(q', !m, q'') \in T$, we have $q' \neq q$ and for all $(q', \tau, q'') \in T$, we have $q' \neq q$.

From a waiting state, a process can only perform receptions (if it can perform anything), whereas in an active state, a process can only perform internal actions or send messages. Examples of wait-only protocols are given by Figures 12 and 13.

In the sequel, we will often refer to the paths of the underlying graph of the protocol. Formally, a *path* in a protocol $\mathcal{P} = (Q, \Sigma, q_{in}, q_f, T)$ is either a control state $q \in Q$ or a finite sequence of transitions in T of the form $(q_0, a_0, q_1)(q_1, a_1, q_2) \dots (q_k, a_k, q_{k+1})$, the first case representing a path from q to q and the second one from q_0 to q_{k+1}.

5.2 Abstract Sets of Configurations

To solve the coverability problem for wait-only protocols in polynomial time, we rely on a sound and complete abstraction of the set of reachable configurations. In the sequel, we consider a wait-only protocol $\mathcal{P} = (Q, \Sigma, q_{in}, q_f, T)$ whose set of states is partitioned into a set of active states Q_A and a set of waiting states Q_W. An *abstract set of configurations* γ is a pair (S, Toks) such that:

- $S \subseteq Q$ is a subset of states, and,
- $\mathit{Toks} \subseteq Q_W \times \Sigma$ is a subset of pairs composed of a waiting state and a message, and,
- $q \notin S$ for all $(q, m) \in \mathit{Toks}$.

We then abstract the set of reachable configurations as a set of states of the underlying protocol. However, as we have seen, some states, like states in Q_A, can host an unbounded number of processes together (this will be the states in S), while some states can only host a bounded number (in fact, 1) of processes together (this will be the states stored in Toks). This happens when a waiting state q answers a rendez-vous m, that has necessarily been requested for a process to be in q. Hence, in Toks, along with a state q, we remember the

last message m having been sent in the path leading from q_{in} to q, which is necessarily in Q_W. Observe that, since several paths can lead to q, there can be $(q, m_1), (q, m_2) \in \mathit{Toks}$ with $m_1 \neq m_2$. We denote by Γ the set of abstract sets of configurations.

Let $\gamma = (S, \mathit{Toks})$ be an abstract set of configurations. Before we go into the configurations represented by γ, we need some preliminary definitions. We note $\mathsf{st}(\mathit{Toks})$ the set $\{q \in Q_W \mid$ there exists $m \in \Sigma$ such that $(q, m) \in \mathit{Toks}\}$ of control states appearing in *Toks*. Given a state $q \in Q$, we let $\mathrm{Rec}(q)$ be the set $\{m \in \Sigma \mid$ there exists $q' \in Q$ such that $(q, ?m, q') \in T\}$ of messages that can be received in state q (if q is not a waiting state, this set is empty). Given two different waiting states q_1 and q_2 in $\mathsf{st}(\mathit{Toks})$, we say q_1 and q_2 are *conflict-free* in γ if there exist $m_1, m_2 \in \Sigma$ such that $m_1 \neq m_2$, $(q_1, m_1), (q_2, m_2) \in \mathit{Toks}$ and $m_1 \notin \mathrm{Rec}(q_2)$ and $m_2 \notin \mathrm{Rec}(q_1)$. We now say that a configuration $C \in \mathcal{C}(\mathcal{P})$ *respects* γ if and only if for all $q \in Q$ such that $C(q) > 0$ one of the following two conditions holds:

1. $q \in S$, or,
2. $q \in \mathsf{st}(\mathit{Toks})$ and $C(q) = 1$ and for all $q' \in \mathsf{st}(\mathit{Toks}) \setminus \{q\}$ such that $C(q') = 1$, we have that q and q' are conflict-free.

Note that these conditions only speak about states q such that $C(q) > 0$ as we are only interested in characterising the reachable states (and unreachable states should not appear in S or $\mathsf{st}(\mathit{Toks})$). Let $[\![\gamma]\!]$ be the set of configurations respecting γ. Note that in $[\![\gamma]\!]$, for q in S there is no restriction on the number of processes that can be put in q and if q in $\mathsf{st}(\mathit{Toks})$, it can host at most one process. Two states from $\mathsf{st}(\mathit{Toks})$ can both host a process if they are conflict-free.

Finally, we will only consider abstract sets of configurations that are *consistent*. This property aims to ensure that concrete configurations that respect it are indeed reachable from states of S. Formally, we say that an abstract set of configurations $\gamma = (S, \mathit{Toks})$ is *consistent* if (i) for all $(q, m) \in \mathit{Toks}$, there exists a path $(q_0, a_0, q_1)(q_1, a_1, q_2) \dots (q_k, a_k, q)$ in $\mathcal{P}$ such that $q_0 \in S$ and $a_0 = !m$ and for all $1 \leq i \leq k$, we have that $a_i = ?m_i$ and that there exists $(q'_i, !m_i, q''_i) \in T$ with $q'_i \in S$, and (ii) for two tokens $(q, m), (q', m') \in \mathit{Toks}$ either $m \in \mathrm{Rec}(q')$ and $m' \in \mathrm{Rec}(q)$, or, $m \notin \mathrm{Rec}(q')$ and $m' \notin \mathrm{Rec}(q)$. Condition (i) ensures that processes in S can indeed lead to a process in the states from $\mathsf{st}(\mathit{Toks})$. Condition (ii) ensures that if in a configuration C, some states in $\mathsf{st}(\mathit{Toks})$ are pairwise conflict-free, then they can all host a process together.

▶ **Lemma 5.1.** *Given $\gamma \in \Gamma$ and a configuration C, there exists $C' \in [\![\gamma]\!]$ such that $C' \geq C$ if and only if $C \in [\![\gamma]\!]$. Checking that $C \in [\![\gamma]\!]$ can be done in polynomial time.*

5.3 Computing Abstract Sets of Configurations

Our polynomial time algorithm is based on the computation of a polynomial length sequence of consistent abstract sets of configurations leading to a final abstract set characterising in a sound and complete manner (with respect to the coverability problem), an abstraction for the set of reachable configurations. This will be achieved by a function $F : \Gamma \to \Gamma$, that inductively computes this final abstract set starting from $\gamma_0 = (\{q_{in}\}, \emptyset)$. Formal definition of the function F relies on intermediate sets $S'' \subseteq Q$ and $\mathit{Toks}'' \subseteq Q_W \times \Sigma$, which are the smallest sets satisfying the conditions described in Table 1.

From S and *Toks*, rules described in Table 1 add states and tokens to S'' and Toks'' from the outgoing transitions from states in S and $\mathsf{st}(\mathit{Toks})$. It must be that every state added to S'' can host an unbounded number of processes, and every state added to Toks'' can host at least one process, furthermore, two conflict-free states in Toks'' should be able to host at least one process at the same time.

Table 1 Definition of S'', $Toks''$ for $\gamma = (S, Toks)$.

Construction of intermediate states S'' and $Toks''$
1. $S \subseteq S''$ and $Toks \subseteq Toks''$
2. for all $(p, \tau, p') \in T$ with $p \in S$, we have $p' \in S''$
3. for all $(p, !a, p') \in T$ with $p \in S$, we have:
a. $p' \in S''$ if $a \notin \mathrm{Rec}(p')$ or if there exists $(q, ?a, q') \in T$ with $q \in S$;
b. $(p', a) \in Toks''$ otherwise (i.e. when $a \in \mathrm{Rec}(p')$ and for all $(q, ?a, q') \in T$, $q \notin S$);
4. for all $(q, ?a, q') \in T$ with $q \in S$ or $(q, a) \in Toks$, we have $q' \in S''$ if there exists $(p, !a, p') \in T$ with $p \in S$;
5. for all $(q, ?a, q') \in T$ with $(q, m) \in Toks$ with $m \neq a$, if there exists $(p, !a, p') \in T$ with $p \in S$, we have:
a. $q' \in S''$ if $m \notin \mathrm{Rec}(q')$;
b. $(q', m) \in Toks''$ if $m \in \mathrm{Rec}(q')$.

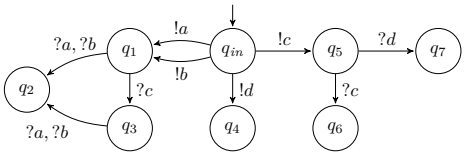

Figure 12 Wait-only protocol $\mathcal{P}_1$.

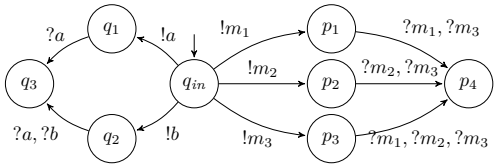

Figure 13 Wait-only protocol $\mathcal{P}_2$.

▶ **Example 5.2.** Consider the wait-only protocol $\mathcal{P}_1$ depicted on Figure 12. From $(\{q_{in}\}, \emptyset)$, rules described in Table 1 construct the following pair $(S_1'', Toks_1'') = (\{q_{in}, q_4\}, \{(q_1, a), (q_1, b), (q_5, c)\})$. In $\mathcal{P}_1$, it is indeed possible to reach a configuration with as many processes as one wishes in the state q_4 by repeating the transition $(q_{in}, !d, q_4)$ (rule 3a). On the other hand, it is possible to put *at most* one process in the waiting state q_1 (rule 3b), because any other attempt from a process in q_{in} will yield a reception of the message a (resp. b) by the process already in q_1. Similarly, we can put at most one process in q_5. Note that in $Toks_1''$, the states q_1 and q_5 are conflict-free and it is hence possible to have simultaneously one process in both of them.

If we apply rules of Table 1 one more time to $(S_1'', Toks_1'')$, we get $S_2'' = \{q_{in}, q_2, q_4, q_6, q_7\}$ and $Toks_2'' = \{(q_1, a), (q_1, b), (q_3, a), (q_3, b), (q_5, c)\}$. We can put at most one process in q_3: to add one, a process will take the transition $(q_1, ?c, q_3)$. Since (q_1, a), $(q_1, b) \in Toks_1''$, there can be at most one process in state q_1, and this process arrived by a path in which the last request of rendez-vous was $!a$ or $!b$. Since $\{a, b\} \subseteq \mathrm{Rec}(q_3)$, by rule 5b, $(q_3, a), (q_3, b)$ are added. On the other hand we can put as many processes as we want in the state q_7 (rule 5a): from a configuration with one process on state q_5, successive non-blocking request on letter c, and rendez-vous on letter d will allow to increase the number of processes in state q_7.

However, one can observe that q_5 can in fact host an unbounded number of processes: once two processes have been put on states q_1 and q_5 respectively (remember that q_1 and q_5 are conflict-free in $(S_1'', Toks_1'')$), iterating rendez-vous on letter c (with transition $(q_1, ?c, q_3)$) and rendez-vous on letter a put as many processes in state q_5.

As a consequence we need to apply another transformation to $(S_2'', Toks_2'')$ to obtain $F(S_1'', Toks_1'')$. We shall see that this second step has no impact when computing $F((\{q_{in}\}, \emptyset))$ hence we have that $F((\{q_{in}\}, \emptyset)) = (S_1'', Toks_1'')$.

We shall finally get that $F(\gamma)$ is equal to (S', Toks'), where the construction of S' from (S'', Toks''), is given by Table 2 and $\mathit{Toks}' = \mathit{Toks}'' \setminus (S \times \Sigma)$, i.e. all states added to S' are removed from Toks' so a state belongs either to S' or to $\mathsf{st}(\mathit{Toks}')$.

Table 2 Definition of S' where $F(\gamma) = (S', \mathit{Toks}')$ for (S'', Toks'').

Construction of state S', the smallest set including S'' and such that:
6. for all $(q_1, m_1), (q_2, m_2) \in \mathit{Toks}''$ such that $m_1 \neq m_2$ and $m_2 \notin \mathrm{Rec}(q_1)$ and $m_1 \in \mathrm{Rec}(q_2)$, we have $q_1 \in S'$;
7. for all $(q_1, m_1), (q_2, m_2), (q_3, m_2) \in \mathit{Toks}''$ s.t $m_1 \neq m_2$ and $(q_2, ?m_1, q_3) \in T$, we have $q_1 \in S'$;
8. for all $(q_1, m_1), (q_2, m_2), (q_3, m_3) \in \mathit{Toks}''$ such that $m_1 \neq m_2$ and $m_1 \neq m_3$ and $m_2 \neq m_3$ and $m_1 \notin \mathrm{Rec}(q_2)$, $m_1 \in \mathrm{Rec}(q_3)$ and $m_2 \notin \mathrm{Rec}(q_1)$, $m_2 \in \mathrm{Rec}(q_3)$, and $m_3 \in \mathrm{Rec}(q_2)$ and $m_3 \in \mathrm{Rec}(q_1)$, we have $q_1 \in S'$.

▶ **Example 5.3.** Now the case of state q_5 evoked in the previous example leads to application of rule 7, since (q_5, c), $(q_1, a) \in \mathit{Toks}_2''$, and (q_3, a) $(q_1, ?c, q_3) \in T$. Finally, we get that $F(S_1'', \mathit{Toks}_1'') = F(F(\{q_{in}\}, \emptyset)) = (\{q_{in}, q_2, q_4, q_5, q_6, q_7\}, \{(q_1, a), (q_1, b), (q_3, a), (q_3, b)\})$. Since q_1 and q_3 are not conflict-free, they won't be reachable together in a configuration.

We consider now the wait-only protocol $\mathcal{P}_2$ depicted on Figure 13. In that case, to compute $F((\{q_{in}\}, \emptyset))$ we will first have $S'' = \{q_{in}\}$ and $\mathit{Toks}'' = \{(q_1, a), (q_2, b), (p_1, m_1), (p_2, m_2), (p_3, m_3)\}$ (using rule 3b), to finally get $F((\{q_{in}\}, \emptyset)) = (\{q_{in}, q_1, p_1\}, \{(q_2, b), (p_2, m_2), (p_3, m_3)\}))$. Applying rule 6 to tokens (q_1, a) and (q_2, b) from Toks'', we obtain that $q_1 \in S'$: whenever one manages to obtain one process in state q_2, this process can answer the requests on message a instead of processes in state q_1, allowing one to obtain as many processes as desired in state q_1. Now since (p_1, m_1), (p_2, m_2) and (p_3, m_3) are in Toks'' and respect the conditions of rule 8, p_1 is added to the set S' of unbounded states. This case is a generalisation of the previous one, with 3 processes. Once one process has been put on state p_2 from q_{in}, iterating the following actions: rendez-vous over m_3, rendez-vous over m_1, non-blocking request of m_2, will ensure as many processes as one wants on state p_1. Finally applying successively F, we get in this case the abstract set $(\{q_{in}, q_1, q_3, p_1, p_2, p_3, p_4\}, \{(q_2, b)\})$.

We show that F satisfies the following properties.

▶ **Lemma 5.4.**
1. *$F(\gamma)$ is consistent and can be computed in polynomial time for all consistent $\gamma \in \Gamma$.*
2. *If $(S', \mathit{Toks}') = F(S, \mathit{Toks})$ then $S \subseteq S'$ (with $S \neq S'$) or $\mathit{Toks} \subseteq \mathit{Toks}'$.*
3. *For all consistent $\gamma \in \Gamma$, if $C \in [\![\gamma]\!]$ and $C \to C'$ then $C' \in [\![F(\gamma)]\!]$.*
4. *For all consistent $\gamma \in \Gamma$, if $C' \in [\![F(\gamma)]\!]$, then there exists $C'' \in \mathcal{C}$ and $C \in [\![\gamma]\!]$ such that $C'' \geq C'$ and $C \to^* C''$.*

5.4 Polynomial Time Algorithm

We now present our polynomial time algorithm to solve CCover for wait-only protocols. We define the sequence $(\gamma_n)_{n \in \mathbb{N}}$ as follows: $\gamma_0 = (\{q_{in}\}, \emptyset)$ and $\gamma_{i+1} = F(\gamma_i)$ for all $i \in \mathbb{N}$. First note that γ_0 is consistent and that $[\![\gamma_0]\!] = \mathcal{I}$ is the set of initial configurations. Using Lemma 5.4, we deduce that γ_i is consistent for all $i \in \mathbb{N}$. Furthermore, each time we apply F to an abstract set of configurations (S, Toks) either S or Toks increases, or (S, Toks) stabilises. Hence for all $n \geq |Q|^2 * |\Sigma|$, we have $\gamma_{n+1} = F(\gamma_n) = \gamma_n$. Let $\gamma_f = \gamma_{|Q|^2 * |\Sigma|}$. Using Lemma 5.4, we get:

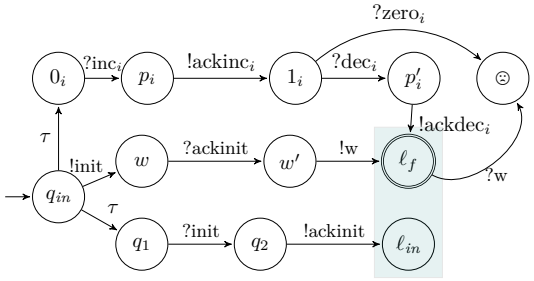

Figure 14 The protocol $\mathcal{P}$ – The coloured zone contains transitions pictured in Figures 15–17.

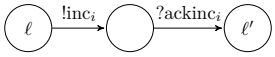

Figure 15 Translation of $(\ell, \mathtt{x}_i+, \ell')$.

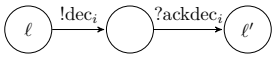

Figure 16 Translation of $(\ell, \mathtt{x}_i-, \ell')$.

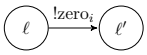

Figure 17 Translation of $(\ell, \mathtt{x}_i{=}0, \ell')$.

▶ **Lemma 5.5.** *Given $C \in \mathcal{C}$, there exists $C_0 \in \mathcal{I}$ and $C' \geq C$ such that $C_0 \to^* C'$ if and only if there exists $C'' \in [\![\gamma_f]\!]$ such that $C'' \geq C$.*

We need to iterate $|Q|^2 * |\Sigma|$ times the function F to compute γ_f and each computation of F can be done in polynomial time. Furthermore checking whether there exists $C'' \in [\![\gamma_f]\!]$ such that $C'' \geq C$ for a configuration $C \in \mathcal{C}$ can be done in polynomial time by Lemma 5.1, hence using the previous lemma we obtain the desired result.

▶ **Theorem 5.6.** *CCover and SCover restricted to wait-only protocols are in Ptime.*

6 Undecidability of Synchro

It is known that Cover[CM] is undecidable in its full generality [18]. This result holds for a very restricted class of counter machines, namely Minsky machines (Minsky-CM for short), which are CM over 2 counters, $\mathtt{x}_1$ and $\mathtt{x}_2$. Actually, it is already undecidable whether there is an execution $(\ell_{in}, \mathbf{0}_{\{\mathtt{x}_1,\mathtt{x}_2\}}) \rightsquigarrow^* (\ell_f, \mathbf{0}_{\{\mathtt{x}_1,\mathtt{x}_2\}})$. Reduction from this last problem gives the following result.

▶ **Theorem 6.1.** *Synchro is undecidable, even for wait-only protocols.*

Fix $M = (\mathrm{Loc}, \ell_0, \{\mathtt{x}_1, \mathtt{x}_2\}, \Delta)$ with $\ell_f \in \mathrm{Loc}$ the final state. W.l.o.g., we assume that there is no outgoing transition from state ℓ_f in the machine. The protocol $\mathcal{P}$ is described in Figures 14–16. The states $\{0_i, p_i, 1_i, p'_i \mid i = 1, 2\}$ will be visited by processes simulating values of counters, while the states in Loc will be visited by a process simulating the different locations in the Minsky-CM. If at the end of the computation, the counters are equal to 0, it means that each counter has been incremented and decremented the same number of times, so that all processes simulating the counters end up in the state ℓ_f. The first challenge is to appropriately check when a counter equals 0. This is achieved thanks to the non-blocking semantics: the process sends a message !zero$_i$ to check if the counter i equals 0. If it is does not, the message will be received by a process that will end up in the deadlock state ☹. The second challenge is to ensure that only one process simulates the Minsky-CM in the states in Loc. This is ensured by the states $\{w, w'\}$. Each time a process arrives in the ℓ_{in} state, another must arrive in the w' state, as a witness that the simulation has begun. This witness must reach ℓ_f for the computation to be a testifier of a positive instance of Synchro, but it should be the first to do so, otherwise a process already in ℓ_f will receive the message "w" and reach the deadlock state ☹. Thus, if two processes simulate the Minsky-CM, there will be two witnesses, and they won't be able to reach ℓ_f together.

7 Conclusion

We have introduced the model of parameterised networks communicating by non-blocking rendez-vous, and showed that safety analysis of such networks becomes much harder than in the framework of classical rendez-vous. Indeed, CCover and SCover become Expspace-complete and Synchro undecidable in our framework, while these problems are solvable in polynomial time in the framework of [13]. We have introduced a natural restriction of protocols, in which control states are partitioned between *active* states (that allow requesting of rendez-vous) and *waiting* states (that can only answer to rendez-vous) and showed that CCover can then be solved in polynomial time. Future work includes finding further restrictions that would yield decidability of Synchro. A candidate would be protocols in which waiting states can only receive *one* message. Observe that in that case, the reduction of Section 6 can be adapted to simulate a test-free CM, hence Synchro for this subclass of protocols is as hard as reachability in Vector Addition Systems with States, i.e. non-primitive recursive [16]. Decidability remains open though.

References

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

2 P. A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. Algorithmic analysis of programs with well quasi-ordered domains. *Information and Computation*, 160(1-2):109–127, 2000.

3 K. R. Apt and D. C. Kozen. Limits for automatic verification of finite-state concurrent systems. *Inf. Process. Lett.*, 22(6):307–309, 1986.

4 A. R. Balasubramanian, J. Esparza, and M. A. Raskin. Finding cut-offs in leaderless rendez-vous protocols is easy. In *FOSSACS'21*, volume 12650 of *LNCS*, pages 42–61. Springer, 2021.

5 G. Delzanno, J. F. Raskin, and L. Van Begin. Towards the automated verification of multithreaded java programs. In *TACAS'02*, volume 2280 of *LNCS*, pages 173–187. Springer, 2002.

6 G. Delzanno, A. Sangnier, R. Traverso, and G. Zavattaro. On the complexity of parameterized reachability in reconfigurable broadcast networks. In *FSTTCS'12*, volume 18 of *LIPIcs*, pages 289–300. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012.

7 A. Durand-Gasselin, J. Esparza, P. Ganty, and R. Majumdar. Model checking parameterized asynchronous shared-memory systems. *Formal Methods in System Design*, 50(2-3):140–167, 2017.

8 J. Esparza. Decidability and complexity of petri net problems – An introduction. In *Advanced Course on Petri Nets*, pages 374–428. Springer, 1998.

9 J. Esparza. Keeping a crowd safe: On the complexity of parameterized verification (invited talk). In Ernst W. Mayr and Natacha Portier, editors, *Proceedings of 31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014)*, volume 25 of *LIPIcs*, pages 1–10. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014.

10 J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *LICS'99*, pages 352–359. IEEE Comp. Soc. Press, July 1999.

11 J. Esparza, P. Ganty, and R. Majumdar. Parameterized verification of asynchronous shared-memory systems. In *CAV'13*, volume 8044 of *LNCS*, pages 124–140. Springer-Verlag, 2013.

12 A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1-2):63–92, 2001.

13 S. M. German and A. P. Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 39(3):675–735, 1992.

14 L. Guillou, A. Sangnier, and N. Sznajder. Safety analysis of parameterised networks with non-blocking rendez-vous, 2023. `arXiv:2307.04546`.

15 F. Horn and A. Sangnier. Deciding the existence of cut-off in parameterized rendez-vous networks. In *CONCUR'20*, volume 171 of *LIPIcs*, pages 46:1–46:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

16 Jérôme Leroux. The reachability problem for petri nets is not primitive recursive. In *FOCS'21*, pages 1241–1252. IEEE, 2021.

17 R.J. Lipton. *The reachability problem requires exponential space*. Research report (Yale University. Department of Computer Science). Department of Computer Science, Yale University, 1976.

18 Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., 1967.

19 C. Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6:223–231, 1978.

20 J. F. Raskin and L. Van Begin. Petri nets with non-blocking arcs are difficult to analyze. In *INFINITY'03*, volume 98 of *Electronic Notes in Theoretical Computer Science*, pages 35–55. Elsevier, 2003.