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


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 \textit{Pspace}\textsuperscript{-}\textit{complete} for a fixed number of processes and in \textit{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 \textit{sender} and the \textit{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 \texttt{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 \texttt{notify} anyway and the message is simply lost. This is the reason why Delzanno et. al. have introduced \textit{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 \textit{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 \textit{P} [13], while it is \textit{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 \textit{non-blocking rendez-vous communication} over a finite alphabet is \textit{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 \textit{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 $\Sigma$, we let $\Sigma^*$ denote the set of finite sequences over $\Sigma$ (or words). Given $w \in \Sigma^*$, we let $|w|$ denote its length: if $w = w_0 \ldots w_{n-1} \in \Sigma^*$, 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 $(m + m')(e) = n(e) + m'(e)$ for all $e \in E$. We say that $n \leq m'$ if and only if $n(e) \leq m'(e)$ for all $e \in E$. If $n \leq m'$, then $m' - m$ is the multiset such that $(m' - m)(e) = n(e) + m(e)$ for all $e \in E$. Given a subset $E' \subseteq E$ and $m \in \mathbb{N}^E$, we denote by $|m||_{E'}$ the sum $\sum_{e \in E'} m(e)$ of elements of $E'$ present in $m$. The size of a multiset $m$ is given by $|m| = \sum_{e \in E} m(e)$. For $e \in E$, we use sometimes the notation $\{e\}$ 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 $\{a, b, b, c\}$ or $\{a, 2 \cdot b, c\}$.

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 using a finite communication alphabet $\Sigma$. Each process can either perform an internal action using the primitive $\tau$, 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 $P = (Q, \Sigma, q_0, q_f, T)$ where $Q$ is a finite set of states, $\Sigma$ is a finite alphabet, $q_0 \in Q$ 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 $P$ 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 $q \in Q \setminus \{q_0\}$. We denote by $C(P)$ the set of configurations and by $I(P)$ the set of initial configurations. Finally for $n \in \mathbb{N} \setminus \{0\}$, we use the notation $C_n(P)$ to represent the set of configurations of size $n$, i.e. $C_n(P) = \{C \in C(P) \mid ||C|| = n\}$. When the protocol is made clear from the context, we shall write $C, I$ and $C_n$.

We explain now the semantics associated with a protocol. For this matter we define the relation $\rightarrow_P \subseteq \bigcup_{n \geq 1} C_n \times \{\tau\} \cup \Sigma \cup \{\text{nb}(m) \mid m \in \Sigma\} \times C_n$ as follows (here $\text{nb}(\cdot)$ is a special symbol). Given $n \in \mathbb{N} \setminus \{0\}$ and $C, C' \in C_n$ and $m \in \Sigma$, we have:
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 \( \mathcal{F}_3(\mathcal{P}) = \{ C \in \mathcal{C}(\mathcal{P}) \mid C(q_f) > 0 \} \) characterises the configurations where one of the processes is in the final state. The second one \( \mathcal{F}_v(\mathcal{P}) = \{ C \in \mathcal{C}(\mathcal{P}) \mid C(Q \setminus \{q_f\}) = 0 \} \) 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 \( \mathcal{F}_3 \) and \( \mathcal{F}_v \). We study three problems: the state coverability problem (SCOVER), the configuration coverability problem (CCOVER) and the synchronisation problem (SYNCHRO), which all take as input a protocol \( \mathcal{P} \) and can be stated as follows:
Are there \( C_0 \in \mathcal{I} \) and \( C_f \in \mathcal{F}_\emptyset \), such that \( C_0 \rightarrow^* C_f \)?

Given \( C \in \mathcal{C} \), are there \( C_0 \in \mathcal{I} \) and \( C' \geq C \), such that \( C_0 \rightarrow^* C' \)?

Are there \( C_0 \in \mathcal{I} \) and \( C_f \in \mathcal{F}_\emptyset \), such that \( C_0 \rightarrow^* C_f \)?

<table>
<thead>
<tr>
<th>Problem name</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCover</td>
<td>Are there ( C_0 \in \mathcal{I} ) and ( C_f \in \mathcal{F}_\emptyset ), such that ( C_0 \rightarrow^* C_f )?</td>
</tr>
<tr>
<td>CCover</td>
<td>Given ( C \in \mathcal{C} ), are there ( C_0 \in \mathcal{I} ) and ( C' \geq C ), such that ( C_0 \rightarrow^* C' )?</td>
</tr>
<tr>
<td>Synchro</td>
<td>Are there ( C_0 \in \mathcal{I} ) and ( C_f \in \mathcal{F}_\emptyset ), such that ( C_0 \rightarrow^* C_f )?</td>
</tr>
</tbody>
</table>

**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 \( q_3 \) 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 \( q_3 \), any other attempt to put another process in this state will induce a reception of message \( a \) by the process already in \( q_3 \), which will hence leave \( q_3 \). Finally, note that for any \( n \in \mathbb{N} \), the configuration \( \{ n \cdot q_3 \} \) 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 \( \text{EXPSPACE} \)-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 \( \text{CAct}(X) \) to represent the set of associated actions given by \( \{ x+, x-, x=0 \mid x \in X \} \cup \{ \bot \} \). Intuitively, \( x+ \) increments the value of the counter \( x \), while \( x- \) decrements it and \( x=0 \) 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 \( M = (\text{Loc}, X, \Delta, \ell_{in}) \) such that Loc is a finite set of locations, \( \ell_{in} \in \text{Loc} \) is an initial location, \( X \) is a finite set of counters, and \( \Delta \subseteq \text{Loc} \times \text{CAct}(X) \times \text{Loc} \) is finite set of transitions.

We will say that a CM is test-free (shortly test-free CM) whenever \( \Delta \cap \text{Loc} \times \{ x=0 \mid x \in X \} \times \text{Loc} = \emptyset \). A configuration of a CM \( M = (\text{Loc}, X, \Delta, \ell_{in}) \) is a pair \( (\ell, v) \) where \( \ell \in \text{Loc} \) 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 \( |M| = |\text{Loc}| + |X| + |\Delta| \). Given two configurations \( (\ell, v) \) and \( (\ell', v') \) and a transition \( \delta \in \Delta \), we define \( (\ell, v) \xrightarrow{\delta} M (\ell', v') \) if and only if \( \delta = (\ell, \text{op}, \ell') \) and one of the following holds:
when the context is clear we shall write \( \to \) instead of \( \trans \). We let \( \inv{x} \) be the valuation such that \( \inv{x}(x) = 0 \) for all \( x \in X \). An execution is a finite sequence of configurations \( (\ell_0, v_0) \to (\ell_1, v_1) \to \ldots \to (\ell_k, v_k) \). It is said to be initial if \( (\ell_0, v_0) = (\ell_{in}, \inv{x}) \). A configuration \( (\ell, v) \) is called reachable if \( (\ell_{in}, \inv{x}) \trans^* (\ell, v) \).

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 \( \text{Cover}[M] \), for \( M \in \{ \text{CM, test-free CM, NB-CM, NB-R-CM} \} \). It takes as input a machine \( M \) in \( M \) (with initial location \( \ell_{in} \) and working over a set \( X \) of counters) and a location \( \ell_f \) and it checks whether there is a valuation \( v \in \mathbb{N}^X \) such that \( (\ell_{in}, \inv{x}) \trans^*(\ell_f, v) \).

In the rest of this section, we will prove that \( \text{Cover}[\text{NB-R-CM}] \) is \( \text{Exp-space-complete} \). To this end, we first establish that \( \text{Cover}[\text{NB-CM}] \) is in \( \text{Exp-space} \), by an adaptation of Rackoff’s proof which shows that coverability in Vector Addition Systems is in \( \text{Exp-space} \) [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.** \( \text{Cover}[\text{NB-CM}] \) and \( \text{Cover}[\text{NB-R-CM}] \) are in \( \text{Exp-space} \).

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

**Theorem 3.4 ([8, 17]).** \( \text{Cover}[2\text{Exp-space bounded test-free CM}] \) is 2\text{Exp-space-hard}. 

---

**Definition 3.2.** A non-blocking test-free counter machine (shortly NB-CM) is a tuple \( M = (\text{Loc}, X, \Delta_b, \Delta_{nb}, \ell_{in}) \) such that \( (\text{Loc}, X, \Delta_b, \ell_{in}) \) is a test-free CM and \( \Delta_{nb} \subseteq \text{Loc} \times \{ \text{nb}(x-) \mid x \in X \} \times \text{Loc} \) 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 \( (\ell, v) \in \text{Loc} \times \mathbb{N}^X \). Given two configurations \( (\ell, v) \) and \( (\ell', v') \) and \( \delta \in \Delta_b \cup \Delta_{nb} \), we extend the transition relation \( (\ell, v) \trans_M (\ell', v') \) over the set \( \Delta_{nb} \) in the following way: for \( \delta = (\ell, \text{nb}(x-), \ell') \in \Delta_{nb} \), we have \( (\ell, v) \trans_M (\ell', v') \) if and only if \( v'(x) = \max(0, v(x) - 1) \), and \( v'(x') = v(x') \) for all \( x' \in X \setminus \{ x \} \).

We say that \( M \) is an NB-CM with restore (shortly NB-R-CM) when \( (\ell, \bot, \ell_{in}) \in \Delta \) for all \( \ell \in \text{Loc} \), 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 \( \Delta \) (resp. an NB-CM with sets of transitions \( \Delta_b \) and \( \Delta_{nb} \)), we will write \( (\ell, v) \trans_M (\ell', v') \) whenever there exists \( \delta \in \Delta \) (resp. \( \delta \in \Delta_b \cup \Delta_{nb} \)) such that \( (\ell, v) \trans_M (\ell', v') \) and use \( \trans_M^* \) to represent the reflexive and transitive closure of \( \trans_M \). When the context is clear we shall write \( \trans \) instead of \( \trans_M \). We let \( \inv{x} \) be the valuation such that \( \inv{x}(x) = 0 \) for all \( x \in X \). An execution is a finite sequence of configurations \( (\ell_0, v_0) \trans (\ell_1, v_1) \trans \ldots \trans (\ell_k, v_k) \). It is said to be initial if \( (\ell_0, v_0) = (\ell_{in}, \inv{x}) \). A configuration \( (\ell, v) \) is called reachable if \( (\ell_{in}, \inv{x}) \trans^* (\ell, v) \).
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

\[ N = (\text{Loc}, X, \Delta_b, \Delta_{nb}, \ell_{in}, L_{out}) \]

such that (Loc, X, 0) is a NB-CM, \( L_{out} \subseteq \text{Loc} \), and there is no outgoing transition from states in \( L_{out} \).

Now fix a 2EXP-bounded test-free CM \( M = (\text{Loc}, X, \Delta, \ell_{in}), \ell_f \in \text{Loc} \) the location to be covered. There is some \( c \), such that, any reachable configuration \((\ell, v)\) satisfies \( v(x) < 2^{c|M|} \) for all \( x \in X \), fix \( n = c|M| \). We build a NB-R-CM \( N \) as pictured in Figure 2. The goal of the procedural NB-CM \( \text{RstInc} \) 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 procedural non-blocking counter machines that are NB-CM with several identified output states: formally, a procedural-NB-CM is a tuple \( N = (\text{Loc}, X, \Delta_b, \Delta_{nb}, \ell_{in}, L_{out}) \) such that (Loc, X, 0) is a NB-CM, \( L_{out} \subseteq \text{Loc} \), and there is no outgoing transition from states in \( L_{out} \).

**Procedural-NB-CM TestSwap\(_i\)(x).** We use a family of procedural-NB-CM defined in [17, 8]: for all \( 0 \leq i < n \), for all \( x \in \overline{Y}_i \), \( \text{TestSwap}_i(x) \) is a procedural-NB-CM with an initial location \( \ell_{in}^{T_2,i,x} \), and two output locations \( \ell_{out}^{T_2,i,y} \) and \( \ell_{out}^{T_2,i,x} \). It tests if the value of \( x \) is equal to 0, using the fact that the sum of the values of \( x \) and \( \overline{x} \) is equal to \( 2^2 \). If \( x = 0 \), it swaps the values of \( x \) and \( \overline{x} \), and the execution ends in the output location \( \ell_{out}^{T_2,i,y} \). Otherwise, counters values are left unchanged and the execution ends in \( \ell_{out}^{T_2,i,x} \). In any case, other counters are not modified by the execution. Note that \( \text{TestSwap}_i(x) \) makes use of variables in \( \bigcup_{1 \leq j < i} Y_j \cup \overline{Y}_i \).

**Procedural NB-CM Rst\(_i\).** We use these machines to define a family of procedural-NB-CM called \( (\text{Rst}_i)_{0 \leq i \leq n} \) that reset the counters in \( Y_i \cup \overline{Y}_i \), assuming that their values are less than or equal to \( 2^2 \). Let \( 0 \leq i \leq n \), we let \( \text{Rst}_i = (\text{Loc}^{R,i}, X', \Delta_b^{R,i}, \Delta_{nb}^{R,i}, \ell_{in}^{R,i}, (\ell_{out}^{R,i})) \). The machine \( \text{Rst}_0 \) is pictured Figure 3. For all \( 0 \leq i < n \), the machine \( \text{Rst}_{i+1} \) uses counters from \( Y_i \cup \overline{Y}_i \) and procedural-NB-CM \( \text{TestSwap}_i(\overline{Z}_i) \) and \( \text{TestSwap}_i(Y_i) \) to control the number of times variables from \( Y_{i+1} \cup \overline{Y}_{i+1} \) are decremented. It is pictured Figure 4. Observe that since \( Y_n = X \), and \( \overline{Y}_n = \emptyset \), the machine \( \text{Rst}_n \) 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 \( Y_i \), \( Z_i \), and \( \overline{Z}_i \) are set to \( 2^{2^i} \) and \( y_i, z_i \), and \( \overline{z}_i \) 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 \( \text{TestSwap}_i(x) \). Moreover, the location \( \ell_{out}^{T_2,i,y} \) will only be reached when the counter \( \overline{y}_i \)
is set to $0$, and this will happen after $2^i$ iterations of the outer loop, again thanks to the properties of $\text{TestSwap}_i(x)$. So, all in all, variables from $Y_i$ and $\overline{Y}_{i+1}$ will take a non-blocking decrement $2^{2^i}2^{i+1}$ times, that is $2^{2^i+1}$.

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

The construction ensures that when one enters $\text{Rst}_i$ with a valuation 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 $\ell_{\text{out}}^i$ is reached with a valuation $v'$ such that: $v'(x) = 0$ for all $x \in Y_i \cup \overline{Y}_i$ and $v'(x) = v(x)$ for all $x \notin Y_i \cup \overline{Y}_i$. 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 $\text{Inc}_i$.** The properties we seek for $\text{Rst}_i$ 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 $0 \leq i \leq n$, $\text{Inc}_i$ is a procedural-NB-CM with initial location $\ell_{\text{Inc},i}$, and unique output location $\ell_{\text{out}}^\text{Inc,i}$. They enjoy the following property: for $0 \leq i < n$, when one enters $\text{Inc}_i$ with a valuation $v$ in which the all variables in $\bigcup_{0 \leq j < i} Y_j \cup \overline{Y}_j$ are initialized and $v(x) = 0$ for all $x \in \overline{Y}_i$, then the location $\ell_{\text{out}}^\text{Inc,i}$ is reached with a valuation $v'$ such that $v'(x) = 2^i$ for all $x \in \overline{Y}_i$, and $v'(x) = v(x)$ for all other $x \in X'$. 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 $\text{RstInc}$.** Finally, let $\text{RstInc}$ be a procedural-NB-CM with initial location $\ell_0$ and output location $\ell_0$, over the set of counters $X'$ and built as an alternation of $\text{Rst}_i$ and $\text{Inc}_i$ for $0 \leq i < n$, finished by $\text{Rst}_n$. It is depicted in Figure 5. Thanks to the properties of the machines $\text{Rst}_i$ and $\text{Inc}_i$, in the output location of each $\text{Inc}_i$ machine, the counters in $\overline{Y}_i$ are set to $2^i$, which allow counters in $Y_{i+1} \cup \overline{Y}_{i+1}$ to be set to $0$ in the output location of $\text{Rst}_{i+1}$. Hence, in location $\ell_{\text{out}}^{\text{Rst,inc}}$, counters in $Y_n = X$ are set to $0$.

From [17, 8], each procedural machine $\text{TestSwap}_i(x)$ and $\text{Inc}_i$ 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 $(\ell_0, 0_X) \sim_M^* (\ell_f, v)$ for some $v \in \mathbb{N}^X$, if and only if $(\ell_{\text{Inc},i}^\text{Inc,i}, 0_X) \sim_N^* (\ell_f, v')$ for some $v' \in \mathbb{N}^{X'}$. Using Theorem 3.4, we obtain:
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 $\{q_1\} + \{q_2\} + \cdots + \{q_k\}$. Observe that there might be $q_i = q_j$ for $i \neq j$. We build the NB-CM $M = (\text{Loc}, X, \Delta_\ell, \Delta_\ell, \ell_{in})$ with $X = Q$. A configuration $C$ of $\mathcal{P}$ is meant to be represented in $M$ by $(\ell_{in}, v)$, with $v(q) = C(q)$ for all $q \in Q$. The only meaningful location of $M$ is then $\ell_{in}$. The other ones are here to ensure correct updates of the counters when simulating a transition. We let $\text{Loc} = \{\ell_{in}\} \cup \{\ell_{t,v}, \ell_{t,v}^a, \ell_{t,v}^b | t = (q, a, q'), v = (p, a, p') \in T\} \cup \{\ell_{q} | t = (q, \tau, q') \in T\} \cup \{\ell_{1}, \ldots, \ell_{n}\}$, with final location $\ell_f$, where $R(m)$ for a message $m \in \Sigma$ has been defined in Section 2. The sets $\Delta_\ell$ and $\Delta_\ell$ 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
reached configuration \((\ell_{in}, v)\) 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 \((\ell_{in}, v)\) is reachable in \(M\), then a configuration \(C \geq v\) is reachable in \(\mathcal{P}\). Then, if it is possible to reach a configuration \((\ell_{in}, v)\) in \(M\) whose counters are high enough to cover \(\ell_{f}\), then the corresponding final execution in \(\mathcal{P}\) will reach a configuration \(C \geq v\), which hence covers \(C_f\).

\textbf{Theorem 4.1.} \(CCover\) over rendez-vous protocols is in \(\text{ExpSpace}\).

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

The reduction from \(\text{Cover}[\text{NB-R-CM}]\) to \(\text{SCover}\) 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 \(Q_M = \text{Loc} \cup \{\ell_b \mid \delta \in \Delta_b\}\), \(\Sigma_M = \{\text{inc}_x, \overline{\text{inc}}_x, \text{dec}_x, \overline{\text{dec}}_x, \text{nbdec}_x \mid x \in X\}\), and \(T_M = \{(\ell, \text{line}_x, \ell_b), (\ell, \text{?inc}_x, \ell), (\ell, \text{?dec}_x, \ell) \mid \delta = (\ell_b, x - , \ell_b) \in \Delta_b \} \cup \{((\ell, \text{dec}_x, \ell_b), (\ell, \text{?dec}_x, \ell)) \mid \delta = (\ell, x - , \ell_j) \in \Delta_b \} \cup \{((\ell, \text{nbdec}_x, \ell_j), (\ell, \text{nb}(x - ), \ell_j)) \in \Delta_{nb} \} \cup \{((\ell, \ell_{in}), \ell_f) \mid (\ell, \bot, \ell) \in \Delta_b \}\) for each counter \(x\). Here, the reception of a message \(\text{inc}_x\) (respectively \(\text{dec}_x\)) works as an acknowledgement, ensuring that a process has indeed received the message \(\text{inc}_x\) (respectively \(\text{dec}_x\)), and that the corresponding counter has been incremented (resp. decremented). For non-blocking decrement, obviously no acknowledgement is required. The protocol \(\mathcal{P} = (Q, \Sigma, q_{in}, \ell, T)\) is then defined with \(Q = Q_M \cup \{1_x, q_e, q'_e \mid x \in X\} \cup \{q_{in}, q_L\}\), \(\Sigma = \Sigma_M \cup \{L, R\}\) and \(T\) is the set of transitions \(T_M\) along with the transitions pictured in Figure 11. Note that there is a transition \((\ell, ?L, q_L)\) 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 \(\ell_{in}\), 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

![Figure 11](image-url)
We then abstract the set of reachable configurations as a set of states of the underlying protocol. We say that a protocol answers a rendez-vous request for a process to be in \(q\), along with a state \(q\), we remember the new process in the set of states \(M\). So every initial execution of \(P\) corresponds to an initial execution of \(M\).

**Theorem 4.2.** \(SCover\) and \(CCover\) over rendez-vous protocols are \(\text{EXPSpace complete}\).

## 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 \((q_0, a_0, q_1)(q_1, a_1, q_2)\ldots(q_k, a_k, q_{k+1})\), 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 \(\gamma\) is a pair \((S, \text{Toks})\) such that:

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

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 \(\text{Toks}\)). 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 \(\text{Toks}\), along with a state \(q\), we remember the
last message $m$ having been sent in the path leading from $q_0$ to $q$, which is necessarily in $Q_W$. Observe that, since several paths can lead to $q$, there can be $(q, m_1), (q, m_2) \in \text{Toks}$ with $m_1 \neq m_2$. We denote by $\Gamma$ the set of abstract sets of configurations.

Let $\gamma = (S, \text{Toks})$ be an abstract set of configurations. Before we go into the configurations represented by $\gamma$, we need some preliminary definitions. We note $\text{st}(\text{Toks})$ the set $\{ q \in Q_W \mid$ there exists $m \in \Sigma$ such that $(q, m) \in \text{Toks} \}$ of control states appearing in $\text{Toks}$. Given a state $q \in Q$, we let $\text{Rec}(q)$ 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 $\text{st}(\text{Toks})$, we say $q_1$ and $q_2$ are conflict-free in $\gamma$ if there exist $m_1, m_2 \in \Sigma$ such that $m_1 \neq m_2$, $(q_1, m_1), (q_2, m_2) \in \text{Toks}$ and $m_1 \notin \text{Rec}(q_2)$ and $m_2 \notin \text{Rec}(q_1)$. We now say that a configuration $C \in \mathcal{C}(P)$ respects $\gamma$ 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. $q \in \text{st}(\text{Toks})$ and $C(q) = 1$ and for all $q' \in \text{st}(\text{Toks}) \setminus \{ q \}$ 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 $\text{st}(\text{Toks})$). Let $\lbrack \gamma \rbrack$ be the set of configurations respecting $\gamma$. Note that in $\lbrack \gamma \rbrack$, for $q$ in $S$ there is no restriction on the number of processes that can be put in $q$ and if $q$ in $\text{st}(\text{Toks})$, it can host at most one process. Two states from $\text{st}(\text{Toks})$ 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 $\gamma = (S, \text{Toks})$ is consistent if (i) for all $(q, m) \in \text{Toks}$, there exists a path $(q_0, a_0, q_1)(q_1, a_1, q_2)\ldots(q_k, a_k, q)$ in $\mathcal{P}$ such that $q_0 \in S$ and $a_0 = \uparrow m$ and for all $1 \leq i \leq k$, we have that $a_i = \uparrow m_i$ and that there exists $(q'_i, m_i, q''_i) \in T$ with $q'_i \in S$, and (ii) for two tokens $(q, m), (q', m') \in \text{Toks}$ either $m \in \text{Rec}(q')$ and $m' \in \text{Rec}(q)$, or, $m \notin \text{Rec}(q')$ and $m' \notin \text{Rec}(q)$. Condition (i) ensures that processes in $S$ can indeed lead to a process in the states from $\text{st}(\text{Toks})$. Condition (ii) ensures that if in a configuration $C$, some states in $\text{st}(\text{Toks})$ 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 $C' \in \lbrack \gamma \rbrack$ such that $C' \geq C$ if and only if $C \in \lbrack \gamma \rbrack$. Checking that $C \in \lbrack \gamma \rbrack$ 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 $\gamma_0 = (\{ q_0 \}, \emptyset)$. Formal definition of the function $F$ relies on intermediate sets $S'' \subseteq Q$ and $\text{Toks}'' \subseteq Q_W \times \Sigma$, which are the smallest sets satisfying the conditions described in Table 1.

From $S$ and $\text{Toks}$, rules described in Table 1 add states and tokens to $S''$ and $\text{Toks}''$ from the outgoing transitions from states in $S$ and $\text{st}(\text{Toks})$. It must be that every state added to $S''$ can host an unbounded number of processes, and every state added to $\text{Toks}''$ can host at least one process, furthermore, two conflict-free states in $\text{Toks}''$ should be able to host at least one process at the same time.
Hence we have that $F$ and rendez-vous on letter $c$ from a configuration with one process in state $q$ are conflict-free in $S''$ once two processes have been put on states $q$ and $q'$. On the other hand we can put as many processes as we want in the state $q$ by repeating the transition $Toks_1$ with a process in the waiting state $q$, iterating rendez-vous on letter $c$, and this process arrived by a path in which the last request of rendez-vous was $!a$ or $!b$. Since $\{a, b\} \subseteq \text{Rec}(q_3)$, 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 $q_7$ 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 \((S', \text{Toks}')\), where the construction of \( S' \) from \((S'', \text{Toks}'')\), is given by Table 2 and \( \text{Toks}' = \text{Toks}'' \setminus (S \times \Sigma) \), i.e. all states added to \( S' \) are removed from \( \text{Toks}' \) so a state belongs either to \( S' \) or to \( \text{st}(\text{Toks}') \).

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

<table>
<thead>
<tr>
<th>Construction of state ( S' ), the smallest set including ( S'' ) and such that:</th>
</tr>
</thead>
<tbody>
<tr>
<td>6. for all ((q_1, m_1), (q_2, m_2) \in \text{Toks}'') such that ( m_1 \neq m_2 ) and ( m_2 \notin \text{Rec}(q_1) ) and ( m_1 \in \text{Rec}(q_2) ), we have ( q_1 \in S' );</td>
</tr>
<tr>
<td>7. for all ((q_1, m_1), (q_2, m_2), (q_3, m_3) \in \text{Toks}'') s.t ( m_1 \neq m_2 ) and ((q_2, ?m_1, q_3) \in T), we have ( q_1 \in S' );</td>
</tr>
<tr>
<td>8. for all ((q_1, m_1), (q_2, m_2), (q_3, m_3) \in \text{Toks}'') such that ( m_1 \neq m_2 ) and ( m_1 \neq m_3 ) and ( m_2 \neq m_3 ) and ( m_1 \notin \text{Rec}(q_2) ), ( m_2 \in \text{Rec}(q_3) ) and ( m_2 \notin \text{Rec}(q_1) ), and ( m_3 \in \text{Rec}(q_2) ) and ( m_4 \in \text{Rec}(q_1) ), we have ( q_1 \in S' );</td>
</tr>
</tbody>
</table>

**Example 5.3.** Now the case of state \( q_5 \) evoked in the previous example leads to application of rule 7, since \((q_5, c), (q_1, a) \in \text{Toks}''\), and \((q_1, a), (q_1, ?c, q_4) \in T\). Finally, we get that \( F(S', \text{Toks}') = F(F((\{q_5\}, \emptyset))) = \{(q_1, a), (q_2, b), (p_1, m_1), (p_2, m_2), (p_3, m_3)\} \) (using rule 3b), to finally get \( F((\{q_5\}, \emptyset)) = \{(q_4, q_2, q_5, q_6, q_7), \{(q_1, a), (q_1, b), (q_3, a), (q_3, b)\}\). Since \( q_1 \) and \( q_4 \) are not conflict-free, they won’t be reachable together in a configuration.

We consider now the wait-only protocol \( P_2 \) depicted on Figure 13. In that case, to compute \( F((\{q_{in}\}, \emptyset)) \) we will first have \( S'' = \{q_{in}\} \) and \( \text{Toks}'' = \{(q_1, a), (q_2, b), (p_1, m_1), (p_2, m_2), (p_3, m_3)\} \) (using rule 3b), to finally get \( F((\{q_{in}\}, \emptyset)) = \{(q_4, q_2, q_5, q_6, q_7), \{(q_1, a), (q_1, b), (q_3, a), (q_3, b)\}\). Applying rule 6 to tokens \((q_1, a)\) and \((q_2, b)\) from \( \text{Toks}'' \), 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 \( \text{Toks}'' \) 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 \( (S', \text{Toks}') = F(S, \text{Toks}) \) then \( S \subseteq S' \) (with \( S \neq S' \)) or \( \text{Toks} \subseteq \text{Toks}' \).
3. For all consistent \( \gamma \in \Gamma \), if \( C \in \gamma \) and \( C \rightarrow C' \) then \( C' \in [F(\gamma)] \).
4. For all consistent \( \gamma \in \Gamma \), if \( C' \in [F(\gamma)] \), then there exists \( C'' \in C \) and \( C \in \gamma \) such that \( C'' \geq C' \) and \( C \rightarrow^* C'' \).

### 5.4 Polynomial Time Algorithm

We now present our polynomial time algorithm to solve \( \text{CCover} \) for wait-only protocols. We define the sequence \((\gamma_i)_{i \in \mathbb{N}}\) as follows: \( \gamma_0 = (\{q_{in}\}, \emptyset) \) and \( \gamma_{i+1} = F(\gamma_i) \) for all \( i \in \mathbb{N} \). First note that \( \gamma_0 \) is consistent and that \( \gamma_0 \) is the set of initial configurations. Using Lemma 5.4, we deduce that \( \gamma_i \) is consistent for all \( i \in \mathbb{N} \). Furthermore, each time we apply \( F \) to an abstract set of configurations \((S, \text{Toks})\) either \( S \) or \( \text{Toks} \) increases, or \((S, \text{Toks})\) stabilises. Hence for all \( n \geq |Q|^2 \times |\Sigma| \), we have \( \gamma_{n+1} = F(\gamma_n) = \gamma_n \). Let \( \gamma_f = \gamma_{|Q|^2 \times |\Sigma|} \). Using Lemma 5.4, we get:
Lemma 5.5. Given \( C \in \mathcal{C} \), there exists \( C_0 \in \mathcal{I} \) and \( C' \geq C \) such that \( C_0 \rightarrow^* C' \) if and only if there exists \( C'' \in [\gamma_f] \) such that \( C'' \geq C \).

We need to iterate \(|Q|^2 \cdot |\Sigma|\) times the function \( F \) to compute \( \gamma_f \) 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. \( \text{CCover and SCover restricted to wait-only protocols are in PTIME.} \)

6 Undecidability of Synchro

It is known that \( \text{COVER[CM]} \) 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, \( x_1 \) and \( x_2 \). Actually, it is already undecidable whether there is an execution \((\ell_{in}, 0_{(x_1, x_2)}) \rightarrow^* (\ell_f, 0_{(x_1, x_2)}) \). Reduction from this last problem gives the following result.

Theorem 6.1. \( \text{SYNCHRO is undecidable, even for wait-only protocols.} \)

Fix \( M = (\text{Loc}, \ell_0, \{x_1, x_2\}, \Delta) \) with \( \ell_f \in \text{Loc} \) the final state. W.l.o.g., we assume that there is no outgoing transition from state \( \ell_f \) in the machine. The protocol \( \mathcal{P} \) is described in Figures 14–16. The states \( \{0, p_i, l_i, p'_i \mid i = 1, 2\} \) 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 \( \ell_f \). 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 \( \text{?zero}_i \), 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 \( \odot \). 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 \( \ell_{in} \) state, another must arrive in the \( w' \) state, as a witness that the simulation has begun. This witness must reach \( \ell_f \) 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 \( \ell_f \) will receive the message “\( w \)” and reach the deadlock state \( \odot \). Thus, if two processes simulate the Minsky-CM, there will be two witnesses, and they won’t be able to reach \( \ell_f \) together.
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, \textsc{Ccover} and \textsc{Scover} become \textsc{Expspace}-complete and \textsc{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 \textsc{Ccover} can then be solved in polynomial time. Future work includes finding further restrictions that would yield decidability of \textsc{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 \textsc{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


