Parameterized safety verification of round-based shared-memory systems

Nathalie Bertrand
Univ Rennes, Inria, CNRS, IRISA, France

Nicolas Markey
Univ Rennes, Inria, CNRS, IRISA, France

Ocan Sankur
Univ Rennes, Inria, CNRS, IRISA, France

Nicolas Waldburger
Univ Rennes, Inria, CNRS, IRISA, France

Abstract

We consider the parameterized verification problem for distributed algorithms where the goal is to develop techniques to prove the correctness of a given algorithm regardless of the number of participating processes. Motivated by an asynchronous binary consensus algorithm [3], we consider round-based distributed algorithms communicating with shared memory. A particular challenge in these systems is that 1) the number of processes is unbounded, and, more importantly, 2) there is a fresh set of registers at each round. A verification algorithm thus needs to manage both sources of infinity. In this setting, we prove that the safety verification problem, which consists in deciding whether all possible executions avoid a given error state, is PSPACE-complete. For negative instances of the safety verification problem, we also provide exponential lower and upper bounds on the minimal number of processes needed for an error execution and on the minimal round on which the error state can be covered.

2012 ACM Subject Classification Theory of computation → Verification by model checking; Theory of computation → Distributed algorithms

Keywords and phrases Verification, Parameterized models, Distributed algorithms

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

Distributed algorithms received in the last decade a lot of attention from the automated verification community. Parameterized verification emerged as a subfield that specifically addresses the verification of distributed algorithms. The main challenge is that distributed algorithms should be proven correct for any number or participating processes. Parameterized models are thus infinite by nature and parameterized verification is in general unfeasible [2]. However, one can recover decidability by considering specific classes of parameterized models, as in the seminal work by German and Sistla where identical finite state machines interact via rendezvous communications [14]. Since then, various models have been proposed to handle various communication means (see [11, 7] for surveys).

Shared memory is one possible communication means. This paper makes first steps towards the parameterized verification of *round-based* distributed algorithms in the shared-memory model; examples of such algorithms can be found in [4, 3, 16]. In particular, our approach covers Aspnes' consensus algorithm [3] which we take as a motivating example. Shared-memory models *without rounds* have been considered in the literature: the verification of safety properties for systems with a leader and many anonymous contributors interacting via a single shared register is coNP-complete [12, 13]; and for Büchi properties, it is NP-complete [10]. Randomized schedulers have also been considered for shared-memory models

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

without leaders; the verification of almost-sure coverability is in EXPSPACE, and is PSPACE-hard [9]. Finally, safety verification is PSPACE-complete for so-called distributed memory automata, that combine local and global memory [8].

Round-based algorithms make verification particularly challenging since they use fresh copies of the registers at each round, and an unbounded number of asynchronous processes means that verification must handle a system with an unbounded number of registers. This is why existing verification techniques fall short at analyzing such algorithms combining two sources of infinity: an unbounded number of processes, and an unbounded number of rounds (hence of registers).

1 int $k := 0$, bool $p \in \{0, 1\}$, $(\mathsf{rg}_b[r])_{b \in \{b_0, b_1\}, r \in \mathbb{N}}$ all initialized to $\bot$;
2 **while** true **do**
3   read from $\mathsf{rg}_{b_0}[k]$ and $\mathsf{rg}_{b_1}[k]$;
4   **if** $\mathsf{rg}_{b_0}[k] = \top$ and $\mathsf{rg}_{b_1}[k] = \bot$ **then** $p := 0$;
5   **else if** $\mathsf{rg}_{b_0}[k] = \bot$ and $\mathsf{rg}_{b_1}[k] = \top$ **then** $p := 1$;
6   write $\top$ to $\mathsf{rg}_{b_p}[k]$;
7   **if** $k > 0$ **then**
8   read from $\mathsf{rg}_{b_{1-p}}[k-1]$;
9   **if** $\mathsf{rg}_{b_{1-p}}[k-1] = \bot$ **then** return p;
10     $k := k+1$;

Algorithm 1 Aspnes' consensus algorithm [3].

Algorithm 1 gives the pseudocode of the binary consensus algorithm proposed by Aspnes [3], in which the processes communicate through shared registers. The algorithm proceeds in asynchronous rounds, which means that there is no *a priori* bound on the round difference between pairs of processes. Furthermore, reading from and writing to registers are separate operations, and a sequence of a read and a write cannot be performed atomically. Each round r has two shared registers $\mathsf{rg}_{b_i}[r]$ for $i \in \{0, 1\}$; notation b_i is used in register indices to avoid confusion with other occurrences of digits 0 and 1. All registers are initialized to a default value $\bot$, and within an execution, their value may only be updated to $\top$. Intuitively, $\mathsf{rg}_{b_i}[r] = \top$ if i is the proposed consensus value at round r.

As usual in distributed consensus algorithms, each process starts with a preference value p. At each round, a process starts by reading the value of the shared registers of that round (**Line 3**). If exactly one of them is set to $\top$, the process updates its preference p to the corresponding value (**Lines 4** and **5**). In all cases, it writes $\top$ to the current-round register that corresponds to its preference p (**Line 6**). Then, it reads the register of the previous round corresponding to the opposite preference $1-p$ (**Line 8**), and if it is $\bot$, the process decides its preference p as return value for the consensus (**Line 9**). To be able to decide its current preference value, a process thus has to win a race against others, writing to a register of its current round k while no other process has written to the register of round $k-1$ for the opposite value. Note that a process can read from and write to the registers of its current round, whereas the registers of the previous rounds are read-only.

The expected properties of such a distributed consensus algorithm are *validity*, *agreement* and *termination*. Validity expresses that if all processes start with the same preference p, then no process can return a value different from p. Agreement expresses that no two processes can return different values. Finally, termination expresses that eventually all processes should return a value. The termination of Aspnes' algorithm is only guaranteed under some fairness constraints on the adversary that schedules the moves of processes [3]. Its validity and agreement properties hold unconditionally. Our objective is to develop automated verification techniques for safety properties, which include validity and agreement.

For a single round –corresponding to one iteration of the while loop– safety properties can be proved applying techniques from [12, 13]. The additional difficulty here lies in the presence of unboundedly many rounds and thus of unboundedly many shared registers. Other settings of parameterized verification exist for round-based distributed algorithms, but none of them apply to asynchronous shared-memory distributed algorithms: they either concern fault-tolerant threshold-based algorithms [5, 6], or synchronous distributed algorithms [15, 1].

Contributions

In this paper, we introduce round-based register protocols, a formalism that models round-based algorithms in which processes communicate via shared memory. Figure 1 depicts a representation of Aspnes' algorithm in this formalism.

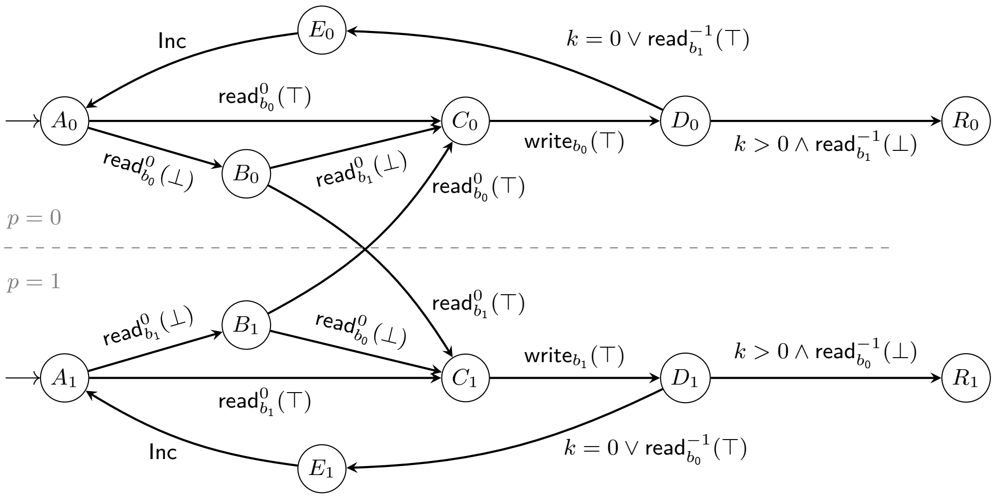


Figure 1 A round-based register protocol for Aspnes' noisy consensus algorithm. Since the first round ($k = 0$) slightly differs from the others, to avoid duplication of the state space, we allow for guards on round number k in the transition labels.

Round-based register protocols form a class of models inspired by register protocols [12, 9, 13], which were introduced to represent shared-memory distributed algorithms *without rounds*. In register protocols, states typically represent the control point of each process as well as the value of its private variables. For instance, the preference p of the process is encoded in the state space: in the top part, $p = 0$ and in the bottom part $p = 1$, as reflected by the states indices. To allow for multiple rounds and round increments, as in Line **10**, we extend register protocols with a new action Inc that labels the transitions from state E_p to state A_p, for each preference $p \in \{0, 1\}$. The processes may read from the registers of the current round but also from those of previous rounds, so reads must specify not only the register identifier but also the lookback distance to the current round: for a process in round k, $\mathsf{read}^{-d}_{b_p}(x)$ represents reading value x from register $\mathsf{rg}_{b_p}[k-d]$.

The validity and agreement properties translate as follows on the register protocols. For validity, one needs to check two properties, one for each common preference $p \in \{0, 1\}$. Namely, if all processes start in state A_0 (resp. A_1), then no processes can enter state R_0 (resp. R_1). Agreement requires that, independently from the initial state of each process in $\{A_0, A_1\}$, no executions reach a configuration with at least one process in R_0 and at least one process in R_1. Both validity and agreement are safety properties.

After introducing round-based register protocols, we study the parameterized verification of safety properties, with the objective of automatically checking whether a configuration involving an error state can be covered for arbitrarily many processes. Our main result is the PSPACE-completeness of this verification problem. We develop an algorithm exploiting the fact that the processes may only read the values of registers within a bounded window on rounds. However, a naive algorithm focusing on the v latest rounds only is hopeless: perhaps surprisingly, we show that the number of *active* rounds (*i.e.*, rounds where a non-idle process is in) may need to be as large as exponential to find an execution covering an error state. The cutoff *i.e.*, the minimal number of processes needed to cover an error state, may also be exponential. The design of our polynomial space algorithm addresses these difficulties by carefully tracking *first-write orders*, that is, the order in which registers are written to for the first time. One of the main technical difficulties of the algorithm is making sure that enough information is stored in this way, allowing the algorithm to solve the verification problem, while also staying in polynomial space.

The rest of the paper is structured as follows. To address the verification of safety properties for round-based register protocols, after introducing their syntax and semantics (Section 2.1), we first observe that they enjoy a monotonicity property (Section 2.2), which justifies the definition of a sound and complete abstract semantics (Section 2.3). We then highlight difficulties of coming up with a polynomial space decision procedure (Section 3.1). Namely, we provide exponential lower bounds on (1) the minimal round number, (2) the minimal number of processes, and (3) the minimal number of active rounds in error executions. We then introduce the central notion of *first-write orders* and its properties (Section 3.2). Section 3.3 details our polynomial-space algorithm, and Section 3.4 presents the complexity-matching lower bound. Due to space constraints, detailed proofs are in the appendix.

2 Round-based shared-memory systems

2.1 Register protocols with rounds

▶ **Definition 1** (Round-based register protocols). *A* round-based register protocol *is a tuple* $\mathcal{P} = \langle Q, q_0, \mathsf{d}, \mathrm{D}, \mathsf{v}, \Delta\rangle$ *where*

- Q *is a finite set of states with a distinguished initial state* q_0*;*
- $\mathsf{d} \in \mathbb{N}$ *is the number of shared registers per round;*
- D *is a finite data alphabet containing* d_0 *the initial value and* $\mathrm{D} \setminus \{d_0\}$ *the values that can be written to the registers;*
- v *is the visibility range (a process on round* k *may read only from rounds in* $[k - \mathsf{v}, k]$*);*
- $\Delta \subseteq Q \times \mathcal{A} \times Q$ *is the set of transitions, where* $\mathcal{A} = \{\mathsf{Inc}\} \cup \{\mathsf{read}_\alpha^{-i}(x) \mid i \in [0, \mathsf{v}], \alpha \in [1, \mathsf{d}], x \in \mathrm{D}\} \cup \{\mathsf{write}_\alpha(x) \mid \alpha \in [1, \mathsf{d}], x \in \mathrm{D} \setminus \{d_0\}\}$ *is the set of actions.*

Intuitively, in a round-based register protocol, the behavior of a process is described by a finite-state machine with a local variable k representing its current round number; note that each process has its own round number, as processes are asynchronous and can be on different rounds. Moreover, there are d registers per round, and the transitions can read and modify these registers. Transitions in round-based register protocols can be labeled with three different types of actions: the Inc action simply increments the current round number of the process; action $\mathsf{read}_\alpha^{-i}(x)$ can be performed by a process at round k when the value of register α of round $k-i$ is x; finally, with the action $\mathsf{write}_\alpha(x)$, a process at round k writes value x to the register α of round k. Note that all actions $\mathsf{read}_\alpha^{-i}(x)$ must satisfy $i \leq \mathsf{v}$; in other words, processes of round k can only read values of registers of rounds $k-\mathsf{v}$ to k.

For complexity purposes, we define the size of the protocol $\mathcal{P} = \langle Q, q_0, \mathsf{d}, \mathrm{D}, \mathsf{v}, \Delta\rangle$ as $|\mathcal{P}| = |Q| + |\mathrm{D}| + |\Delta| + \mathsf{v} + \mathsf{d}$ (thus implicitly assuming that v is given in unary).

Before defining the semantics of round-based register protocols, let us introduce some useful notations. For round number k, we write $\mathsf{rg}_\alpha[k]$ the register α of round k, we let $\mathsf{Reg}_k = \{\mathsf{rg}_\alpha[k] \mid \alpha \in [1, \mathsf{d}]\}$ denote the set of registers of round k, and $\mathsf{Reg} = \bigcup_{k\in\mathbb{N}} \mathsf{Reg}_k$ the set of all registers.

Round-based register protocols execute on several processes asynchronously. The processes communicate via the shared registers, and they progress in a fully asynchronous way through the rounds. A *location* $(q, k) \in Q \times \mathbb{N}$ describes the current state q and round number k of a process, and $\mathsf{Loc} = Q \times \mathbb{N}$ is the set of all locations. A configuration intuitively describes the location of each process, as well as the value of each register. Since processes are anonymous and indistinguishable, the locations of all processes can be represented by maps $\mathsf{Loc} \to \mathbb{N}$ describing how many processes populate each location. Formally, a *concrete configuration* is a pair $\gamma = (\mu, d) \in \mathbb{N}^{\mathsf{Loc}} \times \mathrm{D}^{\mathsf{Reg}}$ such that $\sum_{(q,k)\in\mathsf{Loc}} \mu(q, k) < \infty$. We write $\Gamma = \mathbb{N}^{\mathsf{Loc}} \times \mathrm{D}^{\mathsf{Reg}}$ for the set of all concrete configurations. For a concrete configuration $\gamma = (\mu, d)$, the location multiset μ is denoted $\mathsf{loc}(\gamma)$ and the value $d(k)(\alpha)$ of register α at round k in γ is written $\mathsf{data}_{\mathsf{rg}_\alpha[k]}(\gamma)$. The *size* of γ corresponds to the number of involved processes: $|\gamma| = \sum_{(q,k)\in\mathsf{Loc}} \mu(q, k)$. Configuration γ is *initial* if for every $(q, k) \neq (q_0, 0)$, $\mathsf{loc}(\gamma)(q, k) = 0$, and for every register ξ, $\mathsf{data}_\xi(\gamma) = d_0$. The set of initial concrete configurations therefore consists of all $\mathsf{init}_n = ((q_0, 0)^n, d_0^{\mathsf{Reg}})$. A register is *blank* when it still has initial value d_0. The *support* of the multiset $\mathsf{loc}(\gamma)$ is $\mathsf{supp}(\gamma) = \{(q, k) \mid \mathsf{loc}(\gamma)(q, k) > 0\}$. Finally, for $\gamma, \gamma' \in \Gamma$, we write $\mathsf{data}(\gamma) = \mathsf{data}(\gamma')$ whenever for all $\xi \in \mathsf{Reg}$, $\mathsf{data}_\xi(\gamma) = \mathsf{data}_\xi(\gamma')$.

The evolution from a concrete configuration to another reflects the effect of a process taking a transition in the register protocol. A *move* is thus an element $\theta = (\delta, k)$ consisting of a transition $\delta \in \Delta$ and a round number k; $\mathsf{Moves} = \Delta \times \mathbb{N}$ is the set of all moves. For two concrete configurations γ, γ', we say that γ' is a *successor* of γ if there is a move $((q, a, q'), k) \in \mathsf{Moves}$ satisfying one of the following conditions, depending on the action type:

(i) $a = \mathsf{Inc}$, $\mathsf{loc}(\gamma)(q, k) > 0$, $\mathsf{loc}(\gamma') = \mathsf{loc}(\gamma) \ominus (q, k) \oplus (q', k+1)$, and $\mathsf{data}(\gamma') = \mathsf{data}(\gamma)$;

(ii) $a = \mathsf{read}_\alpha^{-i}(x)$ with $x \in \mathrm{D}$, $\mathsf{data}_{\mathsf{rg}_\alpha[k-i]}(\gamma) = x$, $\mathsf{loc}(\gamma)(q, k) > 0$, $\mathsf{loc}(\gamma') = \mathsf{loc}(\gamma) \ominus (q, k) \oplus (q', k)$ and $\mathsf{data}(\gamma') = \mathsf{data}(\gamma)$;

(iii) $a = \mathsf{write}_\alpha(x)$ with $x \in \mathrm{D} \setminus \{d_0\}$, $\mathsf{data}_{\mathsf{rg}_\alpha[k]}(\gamma') = x$, $\mathsf{loc}(\gamma)(q, k) > 0$, $\mathsf{loc}(\gamma') = \mathsf{loc}(\gamma) \ominus (q, k) \oplus (q', k)$ and for all $\xi \in \mathsf{Reg} \setminus \{\mathsf{rg}_\alpha[k]\}$, $\mathsf{data}_\xi(\gamma') = \mathsf{data}_\xi(\gamma)$.

Here, $\oplus$ and $\ominus$ are operations on multisets, respectively adding and removing elements. The first case represents round increment for a process and the register values are unchanged. The second case represents a read: it requires that the correct value is stored in the corresponding register, that the involved process moves, and that the register values are unchanged. By convention, here, if $k - i < 0$, *i.e.*, for registers with negative round numbers, we let $\mathsf{data}_{\mathsf{rg}_\alpha[k-i]}(\gamma) = d_0$. Finally, the last case represents a write action; it only affects the corresponding register, and the state of the involved process. Note that in all cases, $|\gamma| = |\gamma'|$: the number of processes is constant. If γ' is a successor of γ by move θ, we write $\gamma \xrightarrow{\theta} \gamma'$. A *concrete execution* is an alternating sequence $\gamma_0, \theta_1, \gamma_1, \ldots, \gamma_{\ell-1}, \theta_\ell, \gamma_\ell$ of concrete configurations and moves such that for all i, $\gamma_i \xrightarrow{\theta_{i+1}} \gamma_{i+1}$. In such a case, we write $\gamma_0 \xrightarrow{*} \gamma_\ell$, and we say that γ_ℓ is reachable from γ_0. A location (q, k) is *coverable from* γ_0 when there exists $\gamma \in \mathsf{Reach}(\gamma_0)$ such that $(q, k) \in \mathsf{loc}(\gamma_0)$, and similarly a state q is *coverable from* γ_0 when there exist $k \in \mathbb{N}$ such that (q, k) is coverable from γ_0.

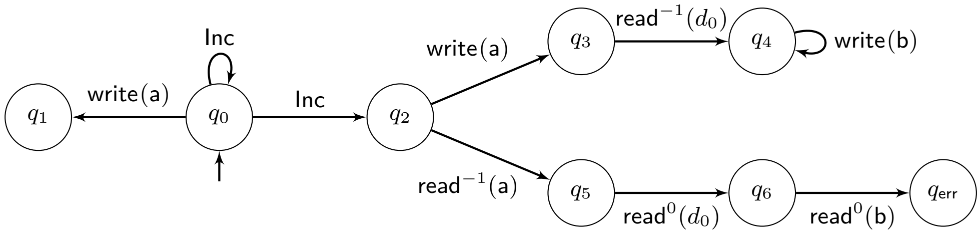


Figure 2 A simple round-based register protocol.

Given a concrete configuration $\gamma \in \Gamma$, $\mathsf{Reach_c}(\gamma)$ denotes the set of all configurations that can be reached from γ: $\mathsf{Reach_c}(\gamma) = \{\gamma' \mid \gamma \xrightarrow{*} \gamma'\}$.

We are now in a position to define our problem of interest:

Safety problem for round-based register protocols
Input: A round-based register protocol $\mathcal{P} = \langle Q, q_0, \mathsf{d}, \mathrm{D}, \mathsf{v}, \Delta \rangle$ and a state $q_{\mathsf{err}} \in Q$
Question: Is it the case that for every $n \in \mathbb{N}$, for every $\gamma \in \mathsf{Reach_c}(\mathsf{init}_n)$ and for every round number k, $\mathsf{loc}(\gamma)(q_{\mathsf{err}}, k) = 0$?

The state q_{err} is referred to as an *error state* that all executions should avoid. An *error configuration* is a configuration in which the error state q_{err} appears, and an *error execution* is an execution containing an error configuration. Given a protocol $\mathcal{P}$ and a state q_{err}, in order to check whether $(\mathcal{P}, q_{\mathsf{err}})$ is a positive instance of the safety problem, we will look for an error execution, and therefore check the dual problem: whether there exist a size n and a configuration $\gamma \in \mathsf{Reach_c}(\mathsf{init}_n)$ such that for some round number k, $\mathsf{loc}(\gamma)(q_{\mathsf{err}}, k) > 0$.

▶ **Example 2.** We illustrate round-based register protocols and their safety problem on the model depicted in Figure 2. This protocol has a single register per round ($\mathsf{d} = 1$, and the register identifier is thus omitted), and set of symbols $\mathrm{D} = \{d_0, a, b\}$. Let us give two examples of concrete executions. State q_4 is coverable from init_1 with the sequence of moves:

$$\pi_1 = \left(\langle (q_0,0)\rangle, \begin{smallmatrix}\mathsf{rg}[0]=d_0\\ \mathsf{rg}[1]=d_0\end{smallmatrix}\right) \xrightarrow{\langle q_0,\mathsf{Inc},q_2\rangle,0} \left(\langle (q_2,1)\rangle, \begin{smallmatrix}\mathsf{rg}[0]=d_0\\ \mathsf{rg}[1]=d_0\end{smallmatrix}\right) \xrightarrow{\langle q_2,\mathsf{write}(a),q_3\rangle,1} \left(\langle (q_3,1)\rangle, \begin{smallmatrix}\mathsf{rg}[0]=d_0\\ \mathsf{rg}[1]=a\end{smallmatrix}\right) \xrightarrow{\langle q_3,\mathsf{read}^{-1}(d_0),q_4\rangle,1} \left(\langle (q_4,1)\rangle, \begin{smallmatrix}\mathsf{rg}[0]=d_0\\ \mathsf{rg}[1]=a\end{smallmatrix}\right).$$

State q_6 is coverable from init_2 as witnessed by the concrete execution:

$$\pi_2 = \left(\langle (q_0,0),(q_0,0)\rangle, \begin{smallmatrix}\mathsf{rg}[0]=d_0\\ \mathsf{rg}[1]=d_0\end{smallmatrix}\right) \xrightarrow{\langle q_0,\mathsf{write}(a),q_1\rangle,0} \left(\langle (q_0,0),(q_1,0)\rangle, \begin{smallmatrix}\mathsf{rg}[0]=a\\ \mathsf{rg}[1]=d_0\end{smallmatrix}\right) \xrightarrow{\langle q_0,\mathsf{Inc},q_2\rangle,0} \left(\langle (q_2,1),(q_1,0)\rangle, \begin{smallmatrix}\mathsf{rg}[0]=a\\ \mathsf{rg}[1]=d_0\end{smallmatrix}\right) \xrightarrow{\langle q_2,\mathsf{read}^{-1}(a),q_5\rangle,1} \left(\langle (q_5,1),(q_1,0)\rangle, \begin{smallmatrix}\mathsf{rg}[0]=a\\ \mathsf{rg}[1]=d_0\end{smallmatrix}\right) \xrightarrow{\langle q_5,\mathsf{read}^{0}(d_0),q_6\rangle,1} \left(\langle (q_6,1),(q_1,0)\rangle, \begin{smallmatrix}\mathsf{rg}[0]=a\\ \mathsf{rg}[1]=d_0\end{smallmatrix}\right).$$

However, it can be observed that no concrete execution can cover both states *at the same round* whatever the number of processes, thus preventing from covering q_{err}. We justify this observation in Subsection 3.2. This example is a positive instance of the safety problem. ◀

▶ **Example 3.** The validity of Aspnes' algorithm can be expressed as two safety properties, with A_0 (resp. A_1) as initial state, and R_1 (resp. R_0) as error state. Let us argue that the protocol of Figure 1 is safe for $q_0 = A_0$ and $q_{\mathsf{err}} = R_1$; the other case is symmetric. Towards a contradiction, suppose there exists an execution $\pi : \mathsf{init}_n \xrightarrow{*} \gamma_1 \xrightarrow{\theta} \gamma_2 \xrightarrow{*} \gamma$ where γ_2 contains a process in the bottom part, and γ_2 is the first such configuration along π. Then

$\theta = ((B_0, \mathsf{read}^0_{b_1}(\top), C_1), k)$ for some k, thus implying that $\mathsf{data}_{\mathsf{rg}_{b_1}[k]}(\gamma_1) = \top$. However, b_1 can only be written to $\mathsf{rg}_{b_1}[k]$ by a process already in the bottom part, which contradicts the minimality of γ_2.

To formally encode agreement of Aspnes' algorithm as a safety property, we make two slight modifications to the protocol from Figure 1. We add an extra initial state q_0 with silent outgoing transitions to A_0 and to A_1; we also add an error state q_{err} that can be covered only if R_0 and R_1 are covered in a same execution. To do so, one can mimick the gadget at q_4 and q_6 in Figure 2, using an extra letter $b \in \mathrm{D}$ and adding Inc loops on both R_0 and R_1, allowing processes to synchronize on the same round, before writing and reading b.

Checking validity and agreement automatically for Aspnes' algorithm requires the machinery that we develop in the rest of the paper. ◀

2.2 Monotonicity

Similarly to other parameterized models, and specifically shared-memory systems [13, 9], round-based register protocols enjoy a monotonicity property called the copycat property. Intuitively, this property states that if a location can be populated with one process, then, increasing the size of the initial configuration, it can be populated by an arbitrary number of them without affecting the behaviour of the other processes. Formally:

▶ **Lemma 4** (Copycat property). *Let $q \in Q$, $k, n, N \in \mathbb{N}$ and $\gamma_{\mathsf{i}}, \gamma_{\mathsf{f}} \in \Gamma$ such that $\gamma_{\mathsf{f}} \in \mathsf{Reach}_{\mathsf{c}}(\gamma_{\mathsf{i}})$ and $(q, k) \in \mathsf{supp}(\gamma_{\mathsf{f}})$. Then there exist $\gamma'_{\mathsf{i}}, \gamma'_{\mathsf{f}} \in \Gamma$ such that $\gamma'_{\mathsf{f}} \in \mathsf{Reach}_{\mathsf{c}}(\gamma'_{\mathsf{i}})$ and:*

- $|\gamma'_{\mathsf{i}}| = |\gamma_{\mathsf{i}}| + N$, $\mathsf{supp}(\gamma'_{\mathsf{i}}) = \mathsf{supp}(\gamma_{\mathsf{i}})$, *and* $\mathsf{data}(\gamma'_{\mathsf{i}}) = \mathsf{data}(\gamma_{\mathsf{i}})$*;*
- $\mathsf{loc}(\gamma'_{\mathsf{f}}) = \mathsf{loc}(\gamma_{\mathsf{f}}) \oplus (q, k)^N$ *and* $\mathsf{data}(\gamma'_{\mathsf{f}}) = \mathsf{data}(\gamma_{\mathsf{f}})$*.*

The copycat property strongly relies on the fact that operations on the registers are non-atomic. In particular it is crucial that processes cannot atomically read and write to a given register, since that could prevent another process from copycating its behaviour.

By the copycat property, the existence of an execution covering the error state q_{err} implies the existence of similar executions for any larger number of processes, which motivates the notion of cutoff. Formally, given $(\mathcal{P}, q_{\mathsf{err}})$ a negative instance of the safety problem, the *cutoff* is the least $n_0 \in \mathbb{N}$ such that for every $n \geq n_0$ there exist $\gamma_n \in \mathsf{Reach}_{\mathsf{c}}(\mathsf{init}_n)$ and $k_n \in \mathbb{N}$ with $\mathsf{loc}(\gamma_n)(q_{\mathsf{err}}, k_n) > 0$.

Another consequence is that any value that has been written to a register can be rewritten, at the cost of increasing the number of involved processes.

▶ **Corollary 5.** *Let $n \in \mathbb{N}$, $\pi : \mathsf{init}_n \xrightarrow{*} \gamma_1 \xrightarrow{*} \gamma$ a concrete execution and $\xi \in \mathsf{Reg}$ a register such that $\mathsf{data}_\xi(\gamma_1) \neq d_0$. There exist $n' \geq n$ and a concrete execution $\pi' : \mathsf{init}_{n'} \xrightarrow{*} \gamma'$ such that $\mathsf{loc}(\gamma) \subseteq \mathsf{loc}(\gamma')$, $\mathsf{data}_\xi(\gamma') = \mathsf{data}_\xi(\gamma_1)$ and for all $\xi' \neq \xi$, $\mathsf{data}_{\xi'}(\gamma') = \mathsf{data}_{\xi'}(\gamma)$.*

2.3 Abstract semantics

The copycat property suggests that, for existential coverability properties, the precise number of processes populating a location is not relevant, only the support of the location multiset matters. As for registers, the only important information to remember is whether they still contain the initial value, or they have been written to (the support then suffices to deduce which values can be written and read). In this section, we therefore define an abstract semantics for round-based register protocols, and we prove it to be sound and complete for the safety problem.

Formally, an *abstract configuration*, or simply a *configuration*, is a pair $\sigma \in 2^{\mathsf{Loc}} \times 2^{\mathsf{Reg}}$, with location support $\mathsf{loc}(\sigma) \in 2^{\mathsf{Loc}}$ and set of written registers $\mathsf{FW}(\sigma) \in 2^{\mathsf{Reg}}$. We write Σ for the set $2^{\mathsf{Loc}} \times 2^{\mathsf{Reg}}$ of all configurations. The (unique) *initial* configuration is $\sigma_{\mathsf{init}} = (\{(q_0,0)\},\emptyset)$. Configuration σ' is a successor of configuration σ if there exists a move $\theta = ((q,a,q'),k) \in$ Moves such that one of the following conditions holds:

(i) $a = \mathsf{Inc}$, $(q,k) \in \mathsf{loc}(\sigma)$, $\mathsf{loc}(\sigma') = \mathsf{loc}(\sigma) \cup \{(q',k{+}1)\}$, and $\mathsf{FW}(\sigma') = \mathsf{FW}(\sigma)$;

(ii) $a = \mathsf{read}_\alpha^{-i}(x)$ with $x \neq d_0$, $(q,k) \in \mathsf{loc}(\sigma)$, $\mathsf{rg}_\alpha[k{-}i] \in \mathsf{FW}(\sigma)$, $\mathsf{loc}(\sigma') = \mathsf{loc}(\sigma) \cup \{(q',k)\}$, $\mathsf{FW}(\sigma') = \mathsf{FW}(\sigma)$, and there is a transition $(q_1, \mathsf{write}_\alpha(x), q_2) \in \Delta$ with $(q_1,k{-}i),(q_2,k{-}i) \in \mathsf{loc}(\sigma)$;

(iii) $a = \mathsf{read}_\alpha^{-i}(d_0)$, $(q,k) \in \mathsf{loc}(\sigma)$, $\mathsf{rg}_\alpha[k{-}i] \notin \mathsf{FW}(\sigma)$, $\mathsf{loc}(\sigma') = \mathsf{loc}(\sigma) \cup \{(q',k)\}$ and $\mathsf{FW}(\sigma') = \mathsf{FW}(\sigma)$;

(iv) $a = \mathsf{write}_\alpha(x)$ with $x \neq d_0$, $(q,k) \in \mathsf{loc}(\sigma)$, $\mathsf{loc}(\sigma') = \mathsf{loc}(\sigma) \cup \{(q',k)\}$ and $\mathsf{FW}(\sigma') = \mathsf{FW}(\sigma) \cup \{\mathsf{rg}_\alpha[k]\}$.

In this case, we write $\sigma \xrightarrow{\theta} \sigma'$. An (abstract) *execution* is an alternating sequence of configurations and moves $\rho = \sigma_0, \theta_1, \sigma_1, \ldots, \sigma_{\ell-1}, \theta_\ell, \sigma_\ell$ such that for all i, $\sigma_i \xrightarrow{\theta_{i+1}} \sigma_{i+1}$, and we write $\sigma \xrightarrow{*} \sigma_\ell$. Similarly to the concrete semantics, $\mathsf{Reach}(\sigma) = \{\sigma' \mid \sigma \xrightarrow{*} \sigma'\}$ denotes the set of *reachable configurations from* σ. Again, a location (q,k) is *coverable from* σ when there exists $\sigma' \in \mathsf{Reach}(\sigma)$ such that $(q,k) \in \mathsf{loc}(\sigma')$, and similarly a state q is *coverable from* σ when there exist $\sigma' \in \mathsf{Reach}(\sigma)$ and $k \in \mathbb{N}$ such that $(q,k) \in \mathsf{loc}(\sigma')$. We simply say that a configuration is *reachable* if it is reachable from the initial configuration σ_{init}, and that a location (resp. a state) is *coverable* if it is coverable from the initial configuration σ_{init}.

▶ **Example 6.** Consider again the protocol of Example 2. The (abstract) execution associated with the concrete execution π_1 in this example is

$$\rho_1 = \big(\{(q_0,0)\},\emptyset\big) \xrightarrow{\langle q_0,\mathsf{Inc},q_2\rangle,0} \big(\{(q_0,0),(q_2,1)\},\emptyset\big) \xrightarrow{\langle q_2,\mathsf{write}(a),q_3\rangle,1}$$
$$\big(\{(q_0,0),(q_2,1),(q_3,1)\},\{\mathsf{rg}[1]\}\big) \xrightarrow{\langle q_3,\mathsf{read}^{-1}(d_0),q_4\rangle,1} \big(\{(q_0,0),(q_2,1),(q_3,1),(q_4,1)\},\{\mathsf{rg}[1]\}\big).$$

Similarly, the execution associated with π_2 is

$$\rho_2 = \big(\{(q_0,0)\},\emptyset\big) \xrightarrow{\langle q_0,\mathsf{write}(a),q_1\rangle,0} \big(\{(q_0,0),(q_1,0)\},\{\mathsf{rg}[0]\}\big) \xrightarrow{\langle q_0,\mathsf{Inc},q_2\rangle,0}$$
$$\big(\{(q_0,0),(q_1,0),(q_2,1)\},\{\mathsf{rg}[0]\}\big) \xrightarrow{\langle q_2,\mathsf{read}^{-1}(a),q_5\rangle,1} \big(\langle(q_0,0),(q_1,0),(q_2,1),(q_5,1)\rangle,\{\mathsf{rg}[0]\}\big)$$
$$\xrightarrow{\langle q_5,\mathsf{read}^{0}(d_0),q_6\rangle,1} \big(\langle(q_0,0),(q_1,0),(q_2,1),(q_5,1),(q_6,1)\rangle,\{\mathsf{rg}[0]\}\big).$$

◀

Note that, in contrast to the concrete semantics, the location support of configurations cannot decrease along an abstract execution. One can easily be convinced that any concrete execution can be lifted to an abstract one, by possibly increasing the support, which is not a problem as long as one is interested in the verification of safety properties. Conversely, from an abstract execution, for a large enough number of processes, using the copycat property one can build a concrete execution with the same final location support. Altogether, the abstract semantics is therefore sound and complete to decide the safety problem on round-based register protocols.

▶ **Theorem 7.** *Let $\mathcal{P}$ be a round-based register protocol, q_{err} a state and $k \in \mathbb{N}$. Then:*

$$\exists n \in \mathbb{N}, \exists \gamma \in \mathsf{Reach_c}(\mathsf{init}_n) : \; (q_{\mathsf{err}},k) \in \mathsf{loc}(\gamma) \quad \iff \quad \exists \sigma \in \mathsf{Reach}(\sigma_{\mathsf{init}}) : \; (q_{\mathsf{err}},k) \in \mathsf{loc}(\sigma) \;.$$

Moreover, for negative instances of the safety problem, the proof of Theorem 7 yields an upper bound on the cutoff, which is linear in the round number at which q_{err} is covered.

▶ **Corollary 8.** *If there exists $k \in \mathbb{N}$ such that (q_{err},k) is coverable, then, letting $N = 2|Q|(k{+}1){+}1$, there exists $\pi : \mathsf{init}_N \xrightarrow{*} \gamma$ such that $(q_{\mathsf{err}},k) \in \mathsf{loc}(\gamma)$.*

3 Decidability and complexity of the safety problem

3.1 Exponential lower bounds everywhere!

To highlight the challenges in coming up with a polynomial space algorithm, we first state three exponential lower bounds when considering safety verification of round-based register protocols. Namely, we prove that (1) the minimal round are which the error state is covered, (2) the minimal number of processes needed for an error execution, and (3) the minimal number of simultaneously active rounds within an error execution, all may need to be exponential in the size of the protocol.

Exponential minimal round

▶ **Proposition 9.** *There exists a family $(\mathcal{BC}_m)_{m\geq 1}$ of round-based register protocols with q_{err} an error state, visibility range $\mathsf{v} = 0$ and number of registers per round $\mathsf{d} = 1$, such that $|\mathcal{BC}_m| = O(m)$ and the minimum round at which q_{err} can be covered is in $\Omega(2^m)$.*

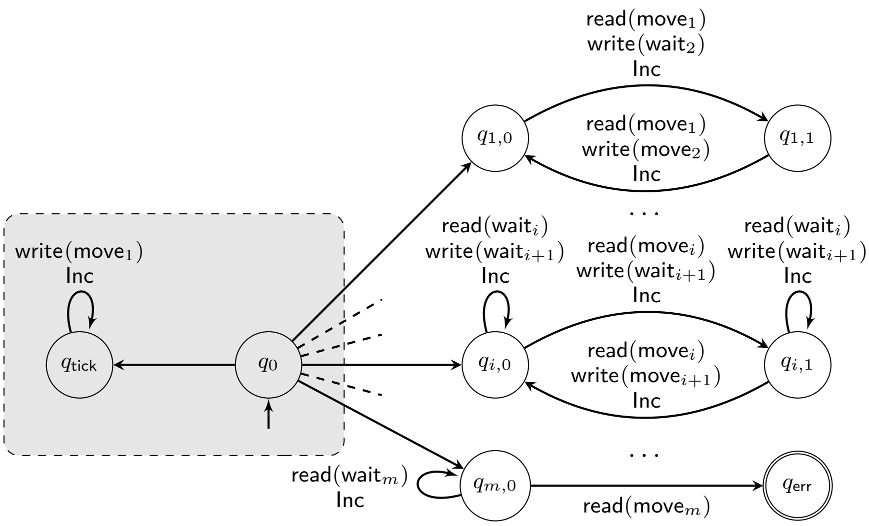


Figure 3 Protocol $\mathcal{BC}_m$ for which an exponential number of rounds is needed to cover q_{err}. For the sake of readability, transitions may be labelled by a sequence of actions: *e.g.*, the transition from $q_{i,0}$ to $q_{i,1}$ is labelled by $\mathsf{read}(\mathsf{move}_1)$, $\mathsf{write}(\mathsf{wait}_2)$, Inc. Such sequences of actions are not performed atomically: one should in principle add intermediate states to split the transition into several consecutive transitions, with one action each. We also use silent transitions (with no action label) that do not perform any action. The tick gadget in grey will be modified in subsequent figures.

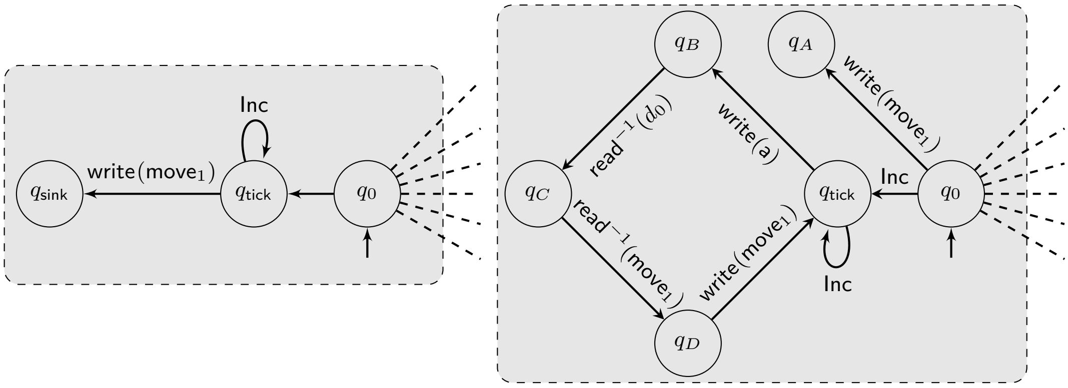
The protocol $\mathcal{BC}_m$, depicted in Figure 3, encodes a binary counter on m bits. The high-level idea of this protocol is that the counter value starts with 0 and is incremented at each round; setting the most significant bit to 1 puts a process in q_{err}. In order to cover q_{err}, any concrete execution needs at least $m+1$ processes: one in q_{tick} ticking every round, and one per bit, in states $\{q_{i,0}, q_{i,1}\}$ to represent the value of the counter's i-th bit. At round k, the value of the i-th least significant bit is 0 if at least one process is at $(q_{i,0}, k)$, and 1 if at least one process is at $(q_{i,1}, k)$. Finally, at round 2^{m-1}, setting the m-th least significant bit –of weight 2^{m-1}– to 1 corresponds to $(q_{\mathsf{err}}, 2^{m-1})$ being covered.

The following proposition is useful for the analysis of $\mathcal{BC}_m$. It states that, in register protocols where $\mathsf{v} = 0$ and $\mathsf{d} = 1$, coverable locations can be covered with a common execution.

▶ **Proposition 10.** *In a register protocol $\mathcal{P}$ with $\mathsf{v} = 0$ and $\mathsf{d} = 1$, for any finite set L of coverable locations, there exists $n \in \mathbb{N}$ and an execution $\rho : \sigma_{\mathsf{init}} \xrightarrow{*} \sigma$ such that, for all $(q, k) \in L$, $(q, k) \in \mathsf{loc}(\sigma)$.*

Our protocol $\mathcal{BC}_m$ satisfies the following property, that entails Proposition 9.

▶ **Proposition 11.** *Let $k \in [0, 2^{m-1}]$. Location (q_{err}, k) is coverable in $\mathcal{BC}_m$ iff $k = 2^{m-1}$.*



(a) An exponential number of processes is needed to cover q_{err}.

(b) An exponential number of active rounds is needed to cover q_{err}.

Figure 4 Two modifications of the tick mechanism of $(\mathcal{BC}_m)_{m\geq 1}$ yielding protocols that need respectively an exponential number of processes and an exponential number of active rounds.

Exponential cutoff

▶ **Proposition 12.** *There exists a family $(\mathcal{P}_m)_{m\geq 1}$ of round-based register protocols with q_{err} an error state, $\mathsf{v} = 0$ and $\mathsf{d} = 1$, such that $|\mathcal{P}_m| = O(m)$ and the minimal number of processes to cover an error configuration is in $\Omega(2^m)$.*

The protocol $\mathcal{P}_m$ is easily obtained from $\mathcal{BC}_m$ by modifying the tick mechanism so that each tick must be performed by a different process, as illustrated in Figure 4a. Since exponentially many ticks are needed to cover q_{err}, the cutoff is also exponential.

Exponential number of simultaneously active rounds

We have seen that the minimal round at which the error state can be covered may be exponential. Perhaps more surprisingly, we now show that the processes may need to spread over exponentially many different rounds. We formalise this with the notion of active rounds. At a configuration along a given execution, round k is *active* when some process is at round k and not idle, *i.e.*, it performs a move later in the execution. The *number of active rounds* of an execution is the maximum number of active rounds at each configuration along the execution.

Towards a polynomial space algorithm for the safety problem, a polynomial bound on the number of active rounds would allow one to guess on-the-fly an error execution by storing only non-idle processes for the current configuration. However, such a polynomial bound does not exist:

▶ **Proposition 13.** *There exists a family $(\mathcal{P}'_m)_{m\geq 1}$ of round-based register protocols with q_{err} an error state, $\mathsf{v} = 1$ and $\mathsf{d} = 1$, such that $|\mathcal{P}'_m| = O(m)$ and the minimal number of active rounds for any error execution is in $\Omega(2^m)$.*

The protocol $\mathcal{P}'_m$ is again obtained from $\mathcal{BC}_m$ by modifying the tick mechanism, as illustrated in Figure 4b. The transitions from q_{tick} to q_B and from q_B to q_C ensure that, for all $k \in [0, 2^{m-1}]$, a must be written to $\mathsf{rg}[k]$ before it is written to $\mathsf{rg}[k-1]$. The transitions from q_C to q_D and from q_D to q_{tick}, on the contrary, ensure that, for all $k \in [1, 2^{m-1}]$, move_1 must be written to $\mathsf{rg}[k-1]$ before it is written to $\mathsf{rg}[k]$. Hence, in an error execution, when move_1 is first written to $\mathsf{rg}[0]$, all rounds from 1 to 2^{m-1} must be active, and the number of active rounds is at least 2^{m-1}.

Note that Proposition 13 requires $\mathsf{v} > 0$. Generally for round-based register protocols with $\mathsf{v} = 0$, processes in different rounds do not interact and an error execution can be reordered: all moves on round 0 first, then all moves on round 1, and so on, so that the number of active rounds is at most 2. Therefore, when $\mathsf{v} = 0$, a naive polynomial-space algorithm for the safety problem consists in computing all coverable states round after round.

3.2 Compatibility and first-write orders

The *compatibility* of coverable locations expresses that they can be covered in a common execution. Formally, two locations (q_1, k_1) and (q_2, k_2) are *compatible* when there exists $\rho : \sigma_{\mathsf{init}} \xrightarrow{*} \sigma$ such that $(q_1, k_1), (q_2, k_2) \in \mathsf{loc}(\sigma)$. In contrast to several other classes of parameterized models (such as broadcast protocols for instance), for round-based register protocols, not all coverable locations are compatible, which makes the safety problem trickier.

▶ **Example 14.** The importance of compatibility can be illustrated on the protocol of Figure 2, whose safety relies on the fact that, for all $k \geq 1$, locations (q_4, k) and (q_6, k) –although both coverable– are *not* compatible. Intuitively, in order to cover (q_4, k), one must write a to $\mathsf{rg}[k]$ and then read d_0 from $\mathsf{rg}[k-1]$, while in order to cover (q_6, k), one must read a from $\mathsf{rg}[k-1]$ and then read d_0 from $\mathsf{rg}[k]$. Since d_0 cannot be written, covering (q_4, k) requires a write to $\mathsf{rg}[k]$ while $\mathsf{rg}[k-1]$ is still blank, and covering (q_6, k) requires the opposite. ◀

More generally, the order in which registers are first written to appears to be crucial for compatibility. We thus define in the sequel the *first-write order* associated with an execution, and use it to give sufficient conditions for compatibility of locations, that we express as being able to combine executions covering these locations.

▶ **Definition 15.** *For $\rho = \sigma_0, \theta_1, \cdots \theta_\ell, \sigma_\ell$ an execution, move θ_i is a* first write *(to $\mathsf{rg}_\alpha[k]$) if $\theta_i = ((q, \mathsf{write}_\alpha(x), q'), k)$ and $\mathsf{rg}_\alpha[k] \notin \mathsf{FW}(\sigma_{i-1})$. The* first-write order *of ρ is the sequence of registers $\mathsf{fwo}(\rho) = \xi_1 : \ldots : \xi_m$ such that the j-th first write along ρ writes to ξ_j.*

Following Example 6, $\mathsf{fwo}(\rho_1) = \mathsf{rg}[1]$ and $\mathsf{fwo}(\rho_2) = \mathsf{rg}[0]$. Two executions with same first-write order can be combined into a "larger" one with same first-write order.

▶ **Lemma 16.** *Let $\rho_1 : \sigma_{\mathsf{init}} \xrightarrow{*} \sigma_1$ and $\rho_2 : \sigma_{\mathsf{init}} \xrightarrow{*} \sigma_2$ be two executions such that $\mathsf{fwo}(\rho_1) = \mathsf{fwo}(\rho_2)$. Then, there exists $\rho : \sigma_{\mathsf{init}} \xrightarrow{*} \sigma$ such that $\mathsf{loc}(\sigma) = \mathsf{loc}(\sigma_1) \cup \mathsf{loc}(\sigma_2)$, $\mathsf{FW}(\sigma) = \mathsf{FW}(\sigma_1) = \mathsf{FW}(\sigma_2)$, and $\mathsf{fwo}(\rho) = \mathsf{fwo}(\rho_1) = \mathsf{fwo}(\rho_2)$.*

It follows that, for any fixed first-write order, there is a *maximal support* that can be covered by executions having that first-write order.

To extend the previous result, we exploit the fact that executions do not read registers arbitrarily far back. It is sufficient to require the first-write orders to have the same projections on all round windows of size v. Formally, for a first-write order f, and two round numbers $k, k' \in \mathbb{N}$ with $k \leq k'$, $\mathsf{proj}_{[k,k']}(\mathsf{f})$ denotes the restriction of f to registers from rounds k to k'.

▶ **Lemma 17.** *Let $\rho_1 \colon \sigma_{\mathsf{init}} \xrightarrow{*} \sigma_1$ and $\rho_2 \colon \sigma_{\mathsf{init}} \xrightarrow{*} \sigma_2$ be two executions of a register protocol with visibility range* v*, such that, for all $k \in \mathbb{N}$, $\mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{fwo}(\rho_1)) = \mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{fwo}(\rho_2))$. Then, there exists $\rho \colon \sigma_{\mathsf{init}} \xrightarrow{*} \sigma$ such that $\mathsf{loc}(\sigma) = \mathsf{loc}(\sigma_1) \cup \mathsf{loc}(\sigma_2)$, $\mathsf{FW}(\sigma) = \mathsf{FW}(\sigma_1) = \mathsf{FW}(\sigma_2)$, and, for all $k \in \mathbb{N}$, $\mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{fwo}(\rho)) = \mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{fwo}(\rho_1)) = \mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{fwo}(\rho_2))$.*

▶ **Example 18.** Agreement of Aspnes' algorithm is closely related to the notion of location (in)compatibility. Intuitively, one requires that no pair of locations (R_0, k_0) and (R_1, k_1) are compatible. Their incompatibility is a consequence of a difference between the first-write orders of the executions that respectively cover them. First, for every $k \geq 1$ and every execution $\rho : \sigma_{\mathsf{init}} \xrightarrow{*} \sigma \xrightarrow{*} \sigma'$, if $\mathsf{rg}_{b_i}[k] \in \mathsf{FW}(\sigma)$ and $\mathsf{rg}_{b_{1-i}}[k-1] \notin \mathsf{FW}(\sigma)$, then $\mathsf{rg}_{b_{1-i}}[k] \notin \mathsf{FW}(\sigma')$; indeed, since $\mathsf{rg}_{b_{1-i}}[k] \notin \mathsf{FW}(\sigma)$, all locations in $\mathsf{loc}(\sigma)$ whose states correspond to $p = 1-i$ are either on round $\leq k-1$ or on round k not on state E_{1-i}, and $\perp$ can no longer be read from $\mathsf{rg}_{b_{1-i}}[k]$; by induction, for all $k' \geq k$, $\mathsf{rg}_{b_{1-i}}[k'] \notin \mathsf{FW}(\sigma')$. Let $\rho_0 : \sigma_{\mathsf{init}} \xrightarrow{*} \sigma_0$ and $\rho_1 : \sigma_{\mathsf{init}} \xrightarrow{*} \sigma_1$ such that, for all $i \in \{0,1\}$, $(R_i, k_i) \in \mathsf{loc}(\sigma_i)$. For all $i \in \{0,1\}$, moves $\theta_i := ((C_i, \mathsf{write}_{b_i}(\top), D_i), k_i)$ and $\theta_i' := ((D_i, \mathsf{read}_{b_{1-i}}^{-1}(\perp), R_i), k_i)$ are in ρ_i, and θ_i appears before θ_i' in ρ_i. Therefore, by letting i such that $k_i \leq k_{1-i}$, ρ_i requires that $\mathsf{rg}_{b_i}[k_i]$ is first-written while $\mathsf{rg}_{b_{1-i}}[k_i - 1]$ is still blank, and therefore that $\mathsf{rg}_{b_i}[k_{1-i}]$ is left blank, while ρ_{1-i} requires a first write on $\mathsf{rg}_{b_i}[k_{1-i}]$, which proves that (R_0, k_0) and (R_1, k_1) are incompatible. Note that $\mathsf{fwo}(\rho_0)$ and $\mathsf{fwo}(\rho_1)$ do not have the same projection on $[k_{1-i} - 1, k_{1-i}]$, which justifies that Lemma 17 does not apply. ◀

3.3 Polynomial-space algorithm

We now present the main contribution of this paper.

▶ **Theorem 19.** *The safety problem for round-based register protocols is in PSPACE.*

To establish Theorem 19, because PSPACE is closed under complement and thanks to Savitch's theorem, it suffices to provide a nondeterministic procedure that finds an error execution (if one exists) within polynomial space. We do this in two steps: first, we give a nondeterministic procedure that iteratively guesses projections of a first-write order and computes the set of coverable locations under those projections, but does not terminate; second, we justify how to run this procedure in polynomial space and that it can be stopped after an exponential number of iterations (thus encodable by a polynomial space binary counter).

The high-level idea of the nondeterministic procedure is to iteratively guess a first-write order f, and to simultaneously compute the set of coverable locations under f. Thanks to Lemma 17, rather than considering a precise first-write order, the algorithm guesses its projections on windows of size v. Concretely, at iteration k, the algorithm guesses $F_k = \mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{f})$ and computes the set $S_k(F_k)$ of states that can be covered at round k under f. These sets are computed incrementally along the prefixes of F_k, called *progressions*, which are considered in increasing order. For each prefix, we check whether a first write to the last register is *feasible*, that is, whether some coverable location is the source of such a write; we reject the computation otherwise.

Algorithm 2 provides the skeleton of this procedure. In **Line 3** of Algorithm 2, the sequence of registers F_k is constructed from F_{k-1} by removing the registers at round $(k-\mathsf{v}-1)$ and non-deterministically inserting some registers at round k. By convention, in the special case where $k = 0$, F_0 is set to a sequence of registers of round 0. From **Line 4** on, one considers the successive progressions of F_k, *i.e.*, prefixes of increasing length, **Line 5** setting

Variables computed : $\mathcal{F} = (F_k)_{k\in\mathbb{N}}$, $(S_k(f))_{k\in\mathbb{N}, f\in\mathsf{Prefixes}(F_k)}$
1 **Initialisation**: $S_0(\varepsilon) := \{q_0\}$; $\forall (k,f) \neq (0,\varepsilon)$, $S_k(f) := \emptyset$; ;
2 **for** k *from* 0 *to* $+\infty$ **do**
3 | non-deterministically choose F_k from F_{k-1} ;
4 | **for** i *from* 0 *to* $\mathsf{length}(F_k)$ **do**
5 | | $f := \mathsf{prefix}_i(F_k)$;
6 | | **if** $f \neq \epsilon$ **then**
7 | | | Let $f = g\!:\!\xi$, and set $S_k(f) := S_k(f) \cup S_k(g)$;
8 | | add to $S_k(f)$ the states that can be covered from round $k{-}1$ by Inc moves;
9 | | **if** *first write to* $\mathsf{last}(f)$ *is feasible* **then**
10 | | | saturate $S_k(f)$ by read and write moves;
11 | | **else**
12 | | | Reject;

Algorithm 2 Non-deterministic polynomial space algorithm to compute the set of coverable states round by round.

f to the prefix of F_k of length i. At **Line 7**, the set of coverable states at round k for progression $f = g\!:\!\xi$ is inherited from the one for progression g.

The next line requires an extra definition. For every $k \in \mathbb{N}$ and every prefix f of F_k, the *synchronisation* $\phi^k_{k-1}(f)$ is the longest prefix of F_{k-1} that coincides with f on rounds $k{-}\mathsf{v}$ to $k{-}1$, *i.e.* such that $\mathsf{proj}_{[k-\mathsf{v},k-1]}(\phi^k_{k-1}(f)) = \mathsf{proj}_{[k-\mathsf{v},k-1]}(f)$. This is always well defined since F_k is obtained from F_{k-1} by removing registers of round $k{-}\mathsf{v}{-}1$, and inserting registers of round k. So $\phi^k_{k-1}(f)$ can be obtained from f by removing registers of round k, and inserting back those of round $k{-}\mathsf{v}{-}1$ that, in F_{k-1}, are before the first register of round in $[k-\mathsf{v}, k-1]$ that is not in f. Similarly, we define the prefixes of f corresponding to previous rounds. For every $r < k{-}1$ and every prefix f of F_k, the *synchronisation* $\phi^k_r(f)$ is defined inductively by $\phi^k_r(f) := \phi^{r+1}_r(\phi^k_{r+1}(f))$, so that $\phi^k_r(f) := \phi^{r+1}_r(\phi^{r+2}_{r+1}(\ldots(\phi^{k-1}_{k-2}(\phi^k_{k-1}(f)))\ldots))$. Last, by convention, $\phi^k_k(f) := f$.

▶ **Example 20.** We illustrate the notion of synchronisation function on a toy example. Consider the sequence of registers $F_1 = \alpha_1\!:\!\beta_1\!:\!\gamma_0\!:\!\delta_0\!:\!\epsilon_1\!:\!\zeta_0$, where the subscripts denote the rounds, and assume that $\mathsf{v} = 1$. The sequence F_2 is obtained from F_1 by removing the round 0 registers $\gamma_0, \delta_0, \zeta_0$, and by inserting some registers of round 2. For instance, one nondeterministically construct $F_2 = \alpha_1 : \eta_2 : \beta_1 : \theta_2 : \epsilon_1$. In that case, for instance $\phi^2_1(\alpha_1 : \eta_2 : \beta_1) = \alpha_1 : \beta_1 : \gamma_0 : \delta_0$; in words, when we are at iteration 2 with progression $\alpha_1\!:\!\eta_2\!:\!\beta_1$, the corresponding progression at iteration 1 is $\alpha_1\!:\!\beta_1\!:\!\gamma_0\!:\!\delta_0$. Also, $\phi^2_1(\alpha_1\!:\!\eta_2) = \alpha_1$ and $\phi^2_1(\alpha_1\!:\!\eta_2\!:\!\beta_1\!:\!\theta_2) = \alpha_1\!:\!\beta_1\!:\!\gamma_0\!:\!\delta_0\!:\!\epsilon_1\!:\!\zeta_0$.

On iteration further, one could have $F_3 = \eta_2\!:\!\kappa_3\!:\!\theta_2$ and thus $\phi^3_1(\eta_2\!:\!\kappa_3) = \phi^2_1(\phi^3_2(\eta_2 : \kappa_3)) = \phi^2_1(\alpha_1\!:\!\eta_2\!:\!\beta_1) = \alpha_1\!:\!\beta_1\!:\!\gamma_0\!:\!\delta_0$. ◀

Now, $S_k(f)$ is defined in two steps. First, **Line 8** adds to $S_k(f)$ the states that can be immediately obtained by an Inc move from states coverable at round $k{-}1$. Formally, $S_k(f) := S_k(f) \cup \{q' \in Q \mid \exists q \in S_{k-1}(\phi^k_{k-1}(f)), (q, \mathsf{Inc}, q') \in \Delta\}$. **Line 9** then checks that a first write to the last register in f is feasible; that is, if $f = g\!:\!\mathsf{rg}_\alpha[k]$, then, one checks whether there exists a write transition $(q, \mathsf{write}_\alpha(x), q') \in \Delta$ with $x \neq d_0$ and $q \in S_k(g)$. Second, in **Line 10**, we saturate $S_k(f)$ by all possible moves at round k. Formally, we add every state $q' \in Q \setminus S_k(f)$ such that there exist $q \in S_k(f)$ and $(q, a, q') \in \Delta$ where action a satisfies one of the following conditions:

- $a = \mathsf{read}^{-j}_\alpha(d_0)$ and $\mathsf{rg}_\alpha[k{-}j]$ does not appear in f;

- $a = \mathsf{read}_\alpha^{-j}(x)$ with $x \neq d_0$, $\mathsf{rg}_\alpha[k{-}j]$ appears in f and there exist $q_1, q_2 \in S_{k-j}(\phi^k_{k-j}(f))$ such that $(q_1, \mathsf{write}_\alpha(x), q_2) \in \Delta$;
- $a = \mathsf{write}_\alpha(x)$ and $\mathsf{rg}_\alpha[k]$ appears in f.

In **Line 12**, the computation is rejected since the guessed first-write order is not feasible.

Characterisation of the sets $S_k(F_k)$ computed in Algorithm 2

For a family of first-write order projections $\mathcal{F} = (F_k)_{k\in\mathbb{N}}$ and a round k, we define $Q\mathsf{cover}(\mathcal{F}, k) = \{q \mid \exists \rho \colon \sigma_{\mathsf{init}} \xrightarrow{*} \sigma \text{ s.t. } (q,k) \in \mathsf{loc}(\sigma) \text{ and } \forall r \leq k, \mathsf{proj}_{[r-\mathsf{v},r]}(\mathsf{fwo}(\rho)) = F_r\}$. In words, $Q\mathsf{cover}(\mathcal{F}, k)$ is the set of states that can be covered at round k by an execution whose first-write order projects to the family $\mathcal{F}$ on windows of size v.

Observe that the only non-deterministic choice in Algorithm 2 is the choice of the sequences F_k; hence, for a given $\mathcal{F} = (F_k)_{k\in\mathbb{N}}$, there is at most one non-rejecting computation whose first-write order projections agrees with family $\mathcal{F}$. In that case, we say that the $\mathcal{F}$-computation of Algorithm 2 is non-rejecting.

▶ **Theorem 21.** *For $\mathcal{F} = (F_k)_{k\in\mathbb{N}}$ a family of projections, if the $\mathcal{F}$-computation of Algorithm 2 is non-rejecting, then the computed sets $(S_k(F_k))_{k\in\mathbb{N}}$ satisfy, for all $k \in \mathbb{N}$, $S_k(F_k) = Q\mathsf{cover}(\mathcal{F}, k)$. Also, for any execution ρ from σ_{init}, letting $\mathcal{F} = (\mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{fwo}(\rho)))_{k\geq 0}$, the $\mathcal{F}$-computation of Algorithm 2 is non-rejecting.*

Building on Algorithm 2, our objective it to design a polynomial space algorithm to decide the safety problem for round-based register protocols. Theorem 21 shows the correctness of the nondeterministic procedure in the following sense: a non-rejecting computation computes all coverable states for the guessed first-write order, and any possible first-write order admits a corresponding non-rejecting computation. To conclude however, the space complexity should be polynomial in the size of the protocol, and termination must be guaranteed by some stopping criterion.

Staying within space budget. As presented, Algorithm 2 needs unbounded space to execute since it stores all sequences of first-write orders F_k and all sets $S_k(f)$. To justify that polynomial space is sufficient, we first observe that some computed values can be ignored after each iteration. Precisely, iteration k only uses variables of iteration $k{-}1$ for increments and of iterations $k{-}\mathsf{v}$ to $k{-}1$ for read/write moves. Thus, at the end of iteration k, all variables indexed with round $k{-}\mathsf{v}$ can be forgotten. It is thus sufficient to store the variables of $\mathsf{v}{+}1$ consecutive rounds.

To conclude, observe also that the maximum length of any sequence F_k is $\mathsf{d}(\mathsf{v}{+}1)$. Therefore each F_k has at most $\mathsf{d}(\mathsf{v}{+}1){+}1$ prefixes, and there are at most $(\mathsf{d}(\mathsf{v}{+}1){+}1)(\mathsf{v}{+}1)$ sets $S_r(f)$ with $r \in [k{-}\mathsf{v}, k]$ for a fixed round number k. We also do not need to store the value of k. All in all, the algorithm can be implemented in space complexity $O(Q \cdot \mathsf{d} \cdot \mathsf{v}^2)$.

Ensuring termination. To exhibit a stopping criterion, we apply the pigeonhole principle to conclude that after a number of iterations at most exponential in $Q \cdot \mathsf{d} \cdot \mathsf{v}^2$, the elements stored in memory repeat from a previous iteration, so that the algorithm starts looping. If q_{err} was not covered at that point, it cannot be covered in further iterations. One can thus use an iteration counter, encoded in polynomial space in the size of the protocol, to count iterations and return a decision when the counter reaches its largest value.

Note that, for negative instances of the safety problem, this gives an exponential upper bound on the round number at which q_{err} is covered. Combined with Corollary 8, it yields an exponential upper bound on the cutoff too. Both match the lower bounds established in Propositions 9 and 12.

▶ **Corollary 22.** *Let $\mathcal{P}$ be a round-based register protocol, and q_{err} an error state. If $(\mathcal{P}, q_{\mathsf{err}})$ is a negative instance of the safety problem, then there exist $K, N \in \mathbb{N}$ both exponential in $|\mathcal{P}|$ such that there exist $k \leq K$ and a concrete execution $\pi : \mathsf{init}_N \xrightarrow{*} \gamma$ such that $(q_{\mathsf{err}}, k) \in \mathsf{loc}(\gamma)$.*

With the space constraints and stopping criterion discussed above, the nondeterministic algorithm decides the safety problem for round-based register protocols. Indeed, it suffices to execute Algorithm 2 up until iteration K and check whether q_{err} appears in one the sets $S_k(F_k)$. If q_{err} is found in some $S_k(F_k)$ with $k \leq K$, then $q_{\mathsf{err}} \in Q\mathsf{cover}(\mathcal{F}, k)$, where $(\mathcal{F}_r)_{r \leq k}$ is the family of projections picked by the computation of the algorithm. Thus, the protocol is unsafe. Conversely, if the protocol is unsafe, then there exist $k \leq K$ and $\rho : \sigma_{\mathsf{init}} \xrightarrow{*} \sigma$ such that $(q_{\mathsf{err}}, k) \in \mathsf{loc}(\sigma)$. Letting $\mathcal{F} = (\mathsf{proj}_{[r-\mathsf{v},r]}(\mathsf{fwo}(\rho)))_{r \in \mathbb{N}}$, the $\mathcal{F}$-computation of the algorithm is non-rejecting, and since $q_{\mathsf{err}} \in Q\mathsf{cover}(\mathcal{F}, k)$, one has $q_{\mathsf{err}} \in S_k(F_k)$.

3.4 PSPACE lower bound

▶ **Theorem 23.** *The safety problem for round-based register protocols is* PSPACE*-hard, even for fixed* $\mathsf{v} = 0$ *and fixed* $\mathsf{d} = 1$*.*

Proof. The proof is by reduction from the validity of QBF.

From a 3-QBF instance, we define a round-based register protocol $\mathcal{P}_{\mathsf{QBF}}$ with an error state q_{err} so that the answer to the safety problem is no if and only if the answer to QBF-validity is yes, *i.e.*, state q_{err} is coverable if, and only if, the QBF instance is valid. This proves that the safety problem is coPSPACE-hard, and therefore that it is PSPACE-hard since PSPACE $=$ coPSPACE.

The protocol $\mathcal{P}_{\mathsf{QBF}}$ that we construct from a QBF instance is partly inspired by the binary counter from Figure 3. Recall that in $\mathcal{BC}_m$, each bit is represented by a subprotocol, and every round corresponds to an increment of the counter value. In $\mathcal{P}_{\mathsf{QBF}}$, each variable is represented by a subprotocol, and every round corresponds to considering a different valuation and evaluating whether it makes the inner SAT formula true. $\mathcal{P}_{\mathsf{QBF}}$ uses a single register per round ($\mathsf{d} = 1$), and the subprotocol corresponding to variable x writes at each round the truth value of x in the considered valuation. The protocol is designed to enumerate all relevant valuations, and take the appropriate decision about the validity.

We fix an instance ϕ of 3-QBF over the $2m$ variables $\{x_0, \cdots, x_{2m-1}\}$

$$\phi = \forall x_{2m-1} \exists x_{2m-2} \forall x_{2m-3} \exists x_{2m-4} \dots \forall x_1 \exists x_0 \bigwedge_{1 \leq j \leq p} a_j \vee b_j \vee c_j \ ,$$

with for every $j \in [1, p]$, $a_j, b_j, c_j \in \{x_i, \neg x_i \mid i \in [0, 2m-1]\}$ are the literals and write ψ for the inner 3-SAT formula.

From ϕ we construct a round-based register protocol on the data alphabet

$$\mathrm{D} := \{\mathsf{wait}_i, \mathsf{yes}_i, \mathsf{no}_i \mid i \in [0, 2m]\} \cup \{\mathtt{x_i}, \neg\mathtt{x_i} \mid i \in [0, 2m-1]\} \cup \{d_0\} \ ,$$

that in particular contains two symbols $\mathtt{x_i}$ and $\neg\mathtt{x_i}$ for each variable x_i. Moreover, we let $\mathsf{v} = 0$ and $\mathsf{d} = 1$.

Thanks to Proposition 10, when $\mathsf{v} = 0$ and $\mathsf{d} = 1$, all coverable locations are compatible, for every finite number of coverable locations, there exists an execution that covers all these locations. We therefore do not have to worry about with which execution a location is coverable, and we will simply write that a location *is coverable* or *is not coverable* and that a symbol *can be written* or *cannot be written* to a given register.

The protocol we construct is represented in Figure 5; it contains several gadgets that we detail in the sequel. Before that we provide a high-level view of $\mathcal{P}_{\mathsf{QBF}}$. In $\mathcal{P}_{\mathsf{QBF}}$, each

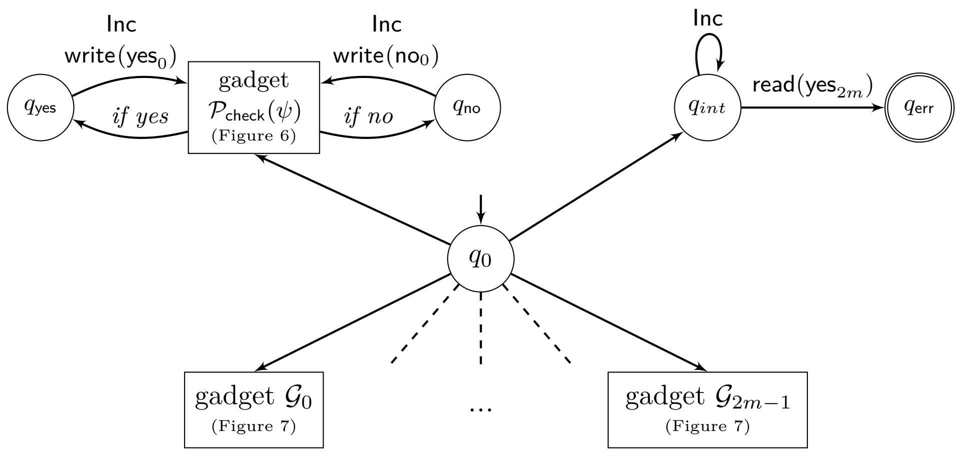


Figure 5 Overview of the protocol $\mathcal{P}_{\mathsf{QBF}}$. All transitions to gadgets go to their initial states.

variable x_i is represented by a subprotocol $\mathcal{G}_i$, and every round corresponds to considering a different valuation and evaluating whether it makes the inner SAT formula true with the gadget $\mathcal{P}_{\mathsf{check}}(\psi)$. The gadget $\mathcal{G}_i$ writes at each round the truth value of x_i in the considered evaluation. The protocol enumerates all valuations: a given round k will correspond to one valuation of the variables of ψ, in which variable x is true if $\mathtt{x}$ can be written to $\mathsf{rg}[k]$, and false if $\neg\mathtt{x}$ can be written to $\mathsf{rg}[k]$. The enumeration of the valuations and corresponding evaluations of ψ are performed so as to take the appropriate decision about the validity of the global formula ϕ.

We start by describing the gadget $\mathcal{P}_{\mathsf{check}}(\psi)$, depicted in Figure 6, that checks whether ψ is satisfied by the valuation under consideration. State q_{yes} corresponds to ψ evaluated to

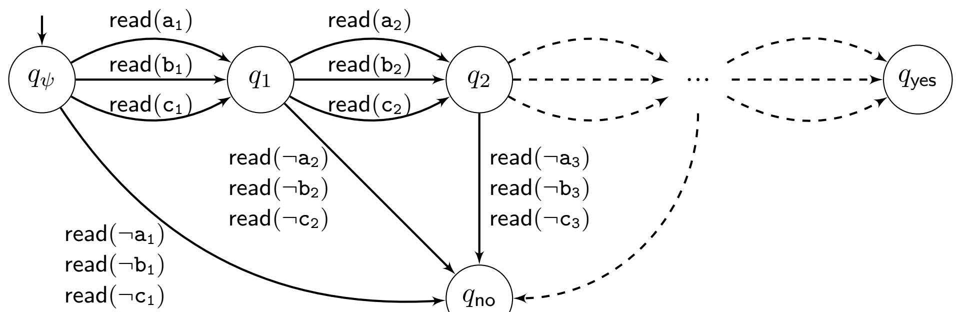


Figure 6 Gadget $\mathcal{P}_{\mathsf{check}}(\psi)$ that checks whether ψ is satisfied by the current valuation.

true and q_{no} corresponding to ψ evaluated to false. Note that we allow transitions labelled by sequences of actions; for instance the transition from state q_ψ to state q_{no} consists of three consecutive reads. The following lemma proves that the gadget $\mathcal{P}_{\mathsf{check}}(\psi)$ indeed checks how ψ evaluates for the current valuation.

▶ **Lemma 24.** *Let $k \in \mathbb{N}$. Suppose that (q_ψ, k) is coverable and that we have a valuation ν of the variables of ψ such that, for every $i \in [0, 2m-1]$:*

- *if $\nu(x_i) = 1$, then $\mathtt{x_i}$ can be written to $\mathsf{rg}[k]$, and $\neg\mathtt{x_i}$ cannot,*
- *if $\nu(x_i) = 0$, then $\neg\mathtt{x_i}$ can be written to $\mathsf{rg}[k]$, and $\mathtt{x_i}$ cannot.*

Then (q_{yes}, k) is coverable if and only if $\nu \models \psi$, and (q_{no}, k) is coverable if and only if $\nu \models \neg\psi$.

We now explain how valuations are enumerated, and how the different quantifiers are handled. The procedure next, given valuation ν, computes the next valuation $\mathsf{next}(\nu)$ that needs to be checked. Eventually, the validity of the formula will be determined by checking whether $\nu_0 \models \psi$ (where ν_0 assigns 0 to all variables) and $\mathsf{next}^k(\nu_0) \models \psi$ for increasing values of $k \geq 1$.

Let ν a valuation of all variables, and define the valuation $\mathsf{next}(\nu)$. Let ϕ_i denote the subformula $Qx_i \ldots \forall x_1 \exists x_0 \psi$ where $Q = \exists$ if i is even, and $Q = \forall$ otherwise. We write $\nu \models \phi_i$ when ϕ_i is true when its free variables $x_{2m-1}, \ldots, x_{i+1}$ are set to their values in ν. The procedure next uses variables $b_i \in \{\mathsf{yes}, \mathsf{no}, \mathsf{wait}\}$ for each $i \in [0, 2m]$, whose role is the following. We will set $b_0 = \mathsf{yes}$ if $\nu \models \psi$, and $b_0 = \mathsf{no}$ otherwise. For any $1 \leq i \leq 2m-1$, $b_i = \mathsf{yes}$ means $\nu \models \phi_i$; $b_i = \mathsf{no}$ means $\nu \not\models \phi_i$; while $b_i = \mathsf{wait}$ means that more valuations need to be checked to determine whether $\nu \models \phi_i$ or not. Given a valuation ν, the procedure next computes, at each iteration i, the truth value of x_i in valuation $\mathsf{next}(\nu)$ and the value of b_{i+1}. After $2m$ iterations, this provides the new valuation $\mathsf{next}(\nu)$ against which ψ must be checked. Formally, $b_0 = \mathsf{yes}$ if $\nu \models \psi$, and $b_0 = \mathsf{no}$ otherwise, and for all $i \in [0, 2m-1]$:

- If $b_i = \mathsf{wait}$, then $\mathsf{next}(\nu)(x_i) := \nu(x_i)$ and $b_{i+1} := \mathsf{wait}$.
- Otherwise
  - If i is even (existential quantifier).
    - if $b_i = \mathsf{yes}$, then $\mathsf{next}(\nu)(x_i) := 0$ and $b_{i+1} := \mathsf{yes}$,
    - if $b_i = \mathsf{no}$ and $\nu(x_i) = 0$, then $\mathsf{next}(\nu)(x_i) := 1$ and $b_{i+1} := \mathsf{wait}$,
    - if $b_i = \mathsf{no}$ and $\nu(x_i) = 1$, then $\mathsf{next}(\nu)(x_i) := 0$ and $b_{i+1} := \mathsf{no}$.
  - if i is odd (universal quantifier),
    - if $b_i = \mathsf{no}$, then $\mathsf{next}(\nu)(x_i) := 0$ and $b_{i+1} := \mathsf{no}$,
    - if $b_i = \mathsf{yes}$ and $\nu(x_i) = 0$, then $\mathsf{next}(\nu)(x_i) := 1$ and $b_{i+1} := \mathsf{wait}$,
    - if $b_i = \mathsf{yes}$ and $\nu(x_i) = 1$, then $\mathsf{next}(\nu)(x_i) := 0$ and $b_{i+1} := \mathsf{yes}$.

Note that variable b_{2m} is computed but not used in the computation. Its value will play the role of a result, e.g., in Lemma 25.

The following lemma formalizes how validity can be checked using next. It is easily proven by induction on m.

▶ **Lemma 25.** *ϕ is valid if and only if, when iterating* next *from valuation ν_0, one eventually obtains a computation of* next *that sets b_{2m} to* yes. *Otherwise, one eventually obtains a computation of* next *that sets b_{2m} to* no.

▶ **Example 26.** Let us illustrate the next operator and Lemma 25 on a small example. Assume

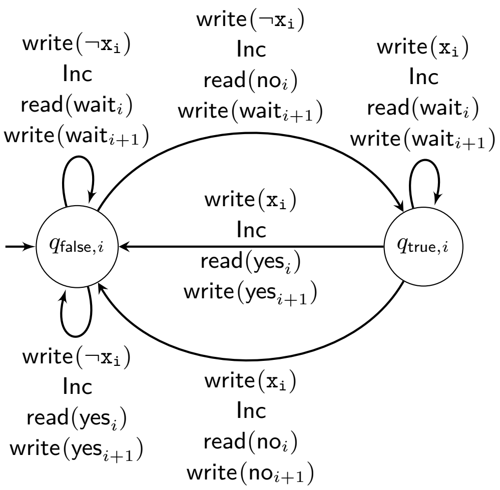
$$\phi = \exists x_2 \forall x_1 \exists x_0\ \neg x_2 \wedge \neg x_1 \wedge (x_1 \vee \neg x_0),$$

which is not a valid formula. To determine that ϕ is not valid, we start by checking the valuation $\nu_0 = (0, 0, 0)$, writing ν_0 as the tuple $(\nu_0(x_0), \nu_0(x_1), \nu_0(x_2))$. Let $\nu = \mathsf{next}(\nu_0)$. ν_0 satisfies the inner formula, hence we set $b_0 = \mathsf{yes}$. By following the procedure of next, we obtain $\nu(x_0) = 0$, $b_1 = \mathsf{yes}$ in the first iteration (in fact, $\nu_0 \models \phi_0$); and $\nu(x_1) = 1$, $b_2 = \mathsf{wait}$ in the second iteration. In fact, even though $\nu_0 \models \psi$, because x_1 is quantified universally, we cannot yet conclude: we must also check whether ψ holds by setting x_1 to 1. This is what $b_2 = \mathsf{wait}$ means, and this is why $\nu(x_1)$ is set to 1. Lastly, we obtain $\nu(x_2) = 0$ and $b_3 = \mathsf{wait}$, therefore $\nu = (0, 1, 0)$.

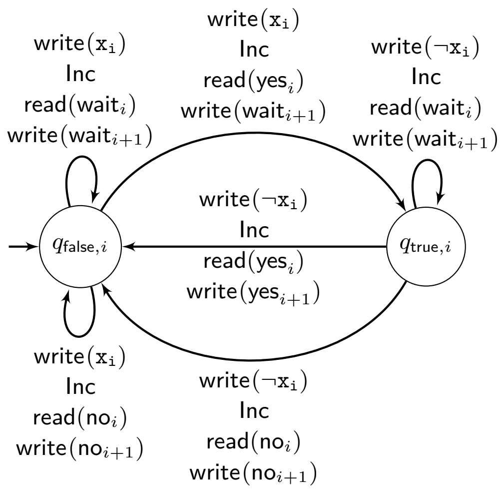
Let $\nu' = \mathsf{next}(\nu) = \mathsf{next}^2(\nu_0)$. We observe that $\nu \not\models \psi$ and set $b_0 = \mathsf{no}$. We then have $\nu'(x_0) = 1$, $b_1 = \mathsf{wait}$, and therefore $\nu'(x_1) = 1$ and $\nu'(x_2) = 0$. In the end, $\nu' = (0, 1, 1)$.

The computation of $\mathsf{next}^3(\nu_0)$ then sets x_2 to 1 because no valuation with $x_2 = 0$ satisfied the formula. We obtain $\mathsf{next}^3(\nu_0) = (1, 0, 0)$ and $\mathsf{next}^4(\nu_0) = (1, 0, 1)$. The computation of $\mathsf{next}^5(\nu_0)$ sets b_{2m} to no, establishing that ϕ is not valid. ◀

Now, we define, for all $i \in [0, 2m-1]$, a gadget $\mathcal{G}_i$ that will play the role of variable x_i. At each round, gadget $\mathcal{G}_i$ receives from gadget $\mathcal{G}_{i-1}$ a value in $\{\mathsf{wait}_i, \mathsf{yes}_i, \mathsf{no}_i\}$ (except for gadget $\mathcal{G}_0$ which receives this value from $\mathcal{P}_{\mathsf{check}}(\psi)$). It transmits a value in $\{\mathsf{wait}_{i+1}, \mathsf{yes}_{i+1}, \mathsf{no}_{i+1}\}$ to $\mathcal{G}_{i+1}$, and modifies the value of variable x_i accordingly, writing either $\mathtt{x_i}$ or $\neg\mathtt{x_i}$ to the register. These gadgets $\mathcal{G}_i$ are given in Figure 7a if x_i is existentially quantified (*i.e.*, i even),



(a) Gadget $\mathcal{G}_i$ for existentially quantified variable x_i (*i.e.*, i even).



(b) Gadget $\mathcal{G}_i$ for universally quantified variable x_i (*i.e.*, i odd).

Figure 7 Illustration of the gadgets $\mathcal{G}_i$.

and Figure 7b if x_i is universally quantified (*i.e.*, i odd). Using those gadgets $\mathcal{G}_i$ and $\mathcal{P}_{\mathsf{check}}(\psi)$ together with the earlier described gadget $\mathcal{P}_{\mathsf{check}}(\psi)$, we define the protocol $\mathcal{P}_{\mathsf{QBF}}$ represented in Figure 5.

Finally, the following lemma justifies the correctness of our construction by formalising the relation between next and $\mathcal{P}_{\mathsf{QBF}}$.

▶ **Lemma 27.** *Let $k \in \mathbb{N}$ and $\nu_k := \mathsf{next}^k(\nu_0)$, the valuation obtained by applying next k times from $\nu_0 := 0^{2m}$. For all $i \in [0, 2m-1]$:*

- *$(q_{\mathsf{false},i}, k)$ is coverable if and only if $\nu_k(x_i) = 0$,*
- *$(q_{\mathsf{true},i}, k)$ is coverable if and only if $\nu_k(x_i) = 1$,*
- *$\neg\mathtt{x_i}$ can be written to $\mathsf{rg}[k]$ if and only if $\nu_k(x_i) = 0$,*
- *$\mathtt{x_i}$ can be written to $\mathsf{rg}[k]$ if and only if $\nu_k(x_i) = 1$.*

Moreover, if $k > 0$, then for all $j \in [0, 2m]$:

- *yes_j can be written to $\mathsf{rg}[k]$ if and only if computation $\nu_k = \mathsf{next}(\nu_{k-1})$ sets b_j to yes,*
- *no_j can be written to $\mathsf{rg}[k]$ if and only if computation $\nu_k = \mathsf{next}(\nu_{k-1})$ sets b_j to no,*
- *wait_j can be written to $\mathsf{rg}[k]$ if and only if computation $\nu_k = \mathsf{next}(\nu_{k-1})$ sets b_j to wait.*

Combining Lemma 27 with Lemma 25 proves that there exists a register to which yes_{2m} can be written if and only if ϕ is valid. Also, q_{err} is coverable in $\mathcal{P}_{\mathsf{QBF}}$ if and only if there exists a register to which yes_{2m} can be written, concluding the proof of Theorem 23. ◀

It may seem surprising that the safety problem is PSPACE-hard already for $\mathsf{d} = 1$ and $\mathsf{v} = 0$, *i.e.*, with a single register and no visibility on previous rounds. For single register protocols without rounds, safety properties can be verified in polynomial time with a simple saturation algorithm. This complexity blowup highlights the expressive power of rounds, independently of the visibility on previous rounds.

Theorems 19 and 23 yield the precise complexity of the safety problem.

▶ **Corollary 28.** *The safety problem for round-based register protocols is PSPACE-complete.*

4 Conclusion

This paper makes a first step towards the automated verification of round-based shared-memory distributed algorithms. We introduce the model of round-based register protocols and solves its parameterized safety verification problem. Precisely, we prove that this problem is PSPACE-complete, providing in particular a non-trivial polynomial space decision algorithm. We also establish exponential lower and upper bounds on the cutoff and on the minimal round at which an error is reached.

Many interesting extensions could be considered, such as assuming the presence of a leader as in [13], or considering other properties than safety. In particular, for algorithms such as Aspnes', beyond validity and agreement that are safety properties, one would need to be able to handle liveness properties (possibly under a fairness assumption) to prove termination.

References

1. C. Aiswarya, Benedikt Bollig, and Paul Gastin. An automata-theoretic approach to the verification of distributed algorithms. *Information and Computation*, 259(3):305–327, 2018. doi:10.1016/j.ic.2017.05.006.
2. Krzysztof Apt and Dexter C. Kozen. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters*, 22(6):307–309, May 1986. doi:10.1016/0020-0190(86)90071-2.
3. James Aspnes. Fast deterministic consensus in a noisy environment. *Journal of Algorithms*, 45(1):16–39, 2002. doi:10.1016/S0196-6774(02)00220-1.
4. James Aspnes. Randomized protocols for asynchronous consensus. *Distributed Computing*, 16(2-3):165–175, 2003. doi:10.1007/s00446-002-0081-5.
5. Nathalie Bertrand, Igor Konnov, Marijana Lazic, and Josef Widder. Verification of randomized consensus algorithms under round-rigid adversaries. *International Journal on Software Tools Technology Transfer*, 23(5):797–821, 2021. doi:10.1007/s10009-020-00603-x.
6. Nathalie Bertrand, Bastien Thomas, and Josef Widder. Guard automata for the verification of safety and liveness of distributed algorithms. In *Proceedings of the 32nd International Conference on Concurrency Theory (CONCUR'21)*, volume 203 of *LIPIcs*, pages 15:1–15:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CONCUR.2021.15.
7. Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder. *Decidability of Parameterized Verification.* Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2015. doi:10.2200/S00658ED1V01Y201508DCT013.
8. Benedikt Bollig, Fedor Ryabinin, and Arnaud Sangnier. Reachability in distributed memory automata. In *Proceedings of the 29th EACSL Annual Conference on Computer Science Logic (CSL'21)*, volume 183 of *LIPIcs*, pages 13:1–13:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CSL.2021.13.

9 Patricia Bouyer, Nicolas Markey, Mickael Randour, Arnaud Sangnier, and Daniel Stan. Reachability in networks of register protocols under stochastic schedulers. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP'16)*, volume 55 of *LIPIcs*, pages 106:1–106:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.106.

10 Antoine Durand-Gasselin, Javier Esparza, Pierre Ganty, and Rupak Majumdar. Model checking parameterized asynchronous shared-memory systems. *Formal Methods Systems Design*, 50(2-3):140–167, 2017. doi:10.1007/s10703-016-0258-3.

11 Javier Esparza. Keeping a crowd safe: On the complexity of parameterized verification (invited talk). In *Proceedings of the 31st International Symposium on Theoretical Aspects of Computer Science (STACS'14)*, volume 25 of *LIPIcs*, pages 1–10. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014. doi:10.4230/LIPIcs.STACS.2014.1.

12 Javier Esparza, Pierre Ganty, and Rupak Majumdar. Parameterized verification of asynchronous shared-memory systems. In *Proceedings of the 25th International Conference on Computer Aided Verification (CAV'13)*, volume 8044 of *Lecture Notes in Computer Science*, pages 124–140. Springer-Verlag, July 2013. doi:10.1007/978-3-642-39799-8_8.

13 Javier Esparza, Pierre Ganty, and Rupak Majumdar. Parameterized verification of asynchronous shared-memory systems. *Journal of the ACM*, 63(1):10:1–10:48, 2016. doi:10.1145/2842603.

14 Steven M. German and A. Prasad Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 39(3):675–735, July 1992. doi:10.1145/146637.146681.

15 Ondrej Lengál, Anthony Widjaja Lin, Rupak Majumdar, and Philipp Rümmer. Fair termination for parameterized probabilistic concurrent systems. In *Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'17)*, volume 10205 of *Lecture Notes in Computer Science*, pages 499–517, 2017. doi:10.1007/978-3-662-54577-5_29.

16 Michel Raynal and Julien Stainer. A Simple Asynchronous Shared Memory Consensus Algorithm Based on Omega and Closing Sets. In *2012 Sixth International Conference on Complex, Intelligent, and Software Intensive Systems*, pages 357–364, 2012. doi:10.1109/CISIS.2012.198.

Technical appendix

This appendix contains details and full proofs that were ommitted in the paper due to space constraints. New statements are numbered with the appendix section letter where they appear followed by a number. Statements that appear in the paper are restated here with their original number.

Additional notions and notations

We start by defining several notions used in several proofs.

A *schedule* is a finite sequence of moves $\theta_1 \cdot \ldots \cdot \theta_\ell$. The schedule $\mathsf{sched}(\rho)$ associated with an execution $\rho = \sigma_0, \theta_1, \sigma_1, \ldots, \sigma_{\ell-1}, \theta_\ell, \sigma_\ell$, is the sequence $\theta_1 \cdot \ldots \cdot \theta_\ell$. We similarly define the schedule $\mathsf{sched}(\pi)$ associated with a concrete execution π.

A schedule s is *applicable* from a configuration σ if there exist an execution ρ and a configuration σ' such that $\rho\colon \sigma \xrightarrow{*} \sigma'$. We then write $\rho\colon \sigma \xrightarrow{\mathsf{s}} \sigma'$ or simply $\sigma \xrightarrow{\mathsf{s}} \sigma'$. Applicability of a schedule from a concrete configuration is defined analogously. Since single moves are particular case of schedules, this also defines applicability of a move to a concrete or abstract configuration.

Given a schedule s and $k \le k'$, $\mathsf{proj}_{[k,k']}(\mathsf{s})$ is the schedule obtained by removing from s on moves whose rounds are not in $[k,k']$, i.e., all moves of the form $((q,a,q'),r)$ with $r \notin [k,k']$. Given $\rho : \sigma \xrightarrow{*} \sigma'$ and $k \in \mathbb{N}$, $\mathsf{proj}_{[0,k]}(\mathsf{sched}(\rho))$ is applicable from σ; write $\mathsf{proj}_{[0,k]}(\rho)$ the execution from σ of schedule $\mathsf{proj}_{[0,k]}(\mathsf{sched}(\rho))$.

Given two executions $\rho : \sigma \xrightarrow{*} \sigma'$ and $\rho' : \sigma' \xrightarrow{*} \sigma''$, we write $\rho \cdot \rho' : \sigma \xrightarrow{*} \sigma''$ the execution of schedule $\mathsf{sched}(\rho) \cdot \mathsf{sched}(\rho')$.

A Proofs and details for Section 2

A.1 Copycat property

▶ **Lemma 4** (Copycat property)**.** *Let $q \in Q$, $k, n, N \in \mathbb{N}$ and $\gamma_\mathsf{i}, \gamma_\mathsf{f} \in \Gamma$ such that $\gamma_\mathsf{f} \in \mathsf{Reach_c}(\gamma_\mathsf{i})$ and $(q,k) \in \mathsf{supp}(\gamma_\mathsf{f})$. Then there exist $\gamma'_\mathsf{i}, \gamma'_\mathsf{f} \in \Gamma$ such that $\gamma'_\mathsf{f} \in \mathsf{Reach_c}(\gamma'_\mathsf{i})$ and:*

- $|\gamma'_\mathsf{i}| = |\gamma_\mathsf{i}| + N$, $\mathsf{supp}(\gamma'_\mathsf{i}) = \mathsf{supp}(\gamma_\mathsf{i})$, *and* $\mathsf{data}(\gamma'_\mathsf{i}) = \mathsf{data}(\gamma_\mathsf{i})$*;*
- $\mathsf{loc}(\gamma'_\mathsf{f}) = \mathsf{loc}(\gamma_\mathsf{f}) \oplus (q,k)^N$ *and* $\mathsf{data}(\gamma'_\mathsf{f}) = \mathsf{data}(\gamma_\mathsf{f})$.

Proof. The key observation is that if a process at location (q,k) takes a move, it can be mimicked right away by any other process also at location (q,k).

Since $\gamma_\mathsf{f} \in \mathsf{Reach_c}(\gamma_\mathsf{i})$, there exists a schedule s such that $\gamma_\mathsf{i} \xrightarrow{s} \gamma_\mathsf{f}$. The proof is by induction on the length (*i.e.*, the number of moves) of s. For the base case where $|s|=0$, we have $\gamma_\mathsf{i} = \gamma_\mathsf{f}$, and it suffices to let $\mathsf{loc}(\gamma'_\mathsf{i}) = \mathsf{loc}(\gamma_\mathsf{i}) \oplus (q,k)^N$ and $\mathsf{data}(\gamma'_\mathsf{i}) = \mathsf{data}(\gamma_\mathsf{i})$.

Suppose now that $\gamma_\mathsf{i} \xrightarrow{s} \gamma_\mathsf{f}$ with $|s| \ge 1$, and that the property holds for schedules of length at most $|s|-1$.

If $\mathsf{loc}(\gamma_\mathsf{i})(q,k) > 0$, then it suffices to define γ'_i such that $\mathsf{loc}(\gamma'_\mathsf{i}) = \mathsf{loc}(\gamma_\mathsf{i}) \oplus (q,k)^N$ and $\mathsf{data}(\gamma'_\mathsf{i}) = \mathsf{data}(\gamma_\mathsf{i})$, and to define γ'_f as the result of applying schedule s from γ'_i, *i.e.*, such that $\gamma'_\mathsf{i} \xrightarrow{s} \gamma'_\mathsf{f}$, keeping the N fresh copies of (q,k) unchanged all along the new execution.

Otherwise, there must exist a move θ in the schedule s such that $\theta = ((q',a,q),k)$ for some state $q' \in Q$ and some action a. We let k' be k unless $a = \mathsf{Inc}$, in which case $k' = k+1$. We decompose s into $s = s_p \cdot \theta \cdot s_s$, and consider the prefix execution $\rho_p\colon \gamma_\mathsf{i} \xrightarrow{s_p} \gamma_p$. Then $|s_p| \le |s|-1$, and by induction hypothesis, there exist γ'_i, γ'_p and s'_p with $\gamma'_\mathsf{i} \xrightarrow{s'_p} \gamma'_p$,

$\mathsf{loc}(\gamma'_p) = \mathsf{loc}(\gamma_p) \oplus (q,k')^N$ and $\mathsf{data}(\gamma'_p) = \mathsf{data}(\gamma_p)$. Moreover, $|\gamma'_i| = |\gamma_i| + N$, $\mathsf{supp}(\gamma'_\mathsf{i}) = \mathsf{supp}(\gamma_\mathsf{i})$ and $\mathsf{data}(\gamma'_\mathsf{i}) = \mathsf{data}(\gamma_\mathsf{i})$. Since move θ is applicable to γ_p, θ^{N+1} is applicable to γ'_p. Letting $s' = s'_p \cdot \theta^{N+1} \cdot s_s$, we obtain that $\gamma'_\mathsf{i} \xrightarrow{s'} \gamma'_\mathsf{f}$ with $\mathsf{loc}(\gamma'_\mathsf{f}) = \mathsf{loc}(\gamma_\mathsf{f}) \oplus (q,k)^N$ and $\mathsf{data}(\gamma'_\mathsf{f}) = \mathsf{data}(\gamma_\mathsf{f})$, which concludes the proof. ◀

A.2 Soundness and completeness of the abstraction

▶ **Theorem 7.** *Let $\mathcal{P}$ be a round-based register protocol, q_err a state and $k \in \mathbb{N}$. Then:*

$$\exists n \in \mathbb{N}, \exists \gamma \in \mathsf{Reach_c}(\mathsf{init}_n) : \ (q_\mathsf{err}, k) \in \mathsf{loc}(\gamma) \quad \Longleftrightarrow \quad \exists \sigma \in \mathsf{Reach}(\sigma_\mathsf{init}) : \ (q_\mathsf{err}, k) \in \mathsf{loc}(\sigma) \ .$$

Proof. The direct implication is simpler to prove: one can easily mimick a concrete execution in the abstraction. The right-to-left implication relies on the copycat property, Lemma 4, and Corollary 5, to accomodate the differences between the concrete and abstract semantics.

In the following, for every concrete configuration $\gamma \in \Gamma$, we write $\mathsf{abst}(\gamma) \in \Sigma$ for the corresponding (abstract) configuration defined by $\mathsf{loc}(\mathsf{abst}(\gamma)) = \mathsf{supp}(\gamma)$ and $\mathsf{FW}(\mathsf{abst}(\gamma)) = \{\xi \in \mathsf{Reg} \mid \mathsf{data}_\xi(\gamma) \neq d_0\}$. We start with the direct implication, proving that a concrete execution from init_n can be directly converted into an abstract execution that covers more locations.

▶ **Lemma A.1.** *Let $n \in \mathbb{N}$ and $\pi\colon \mathsf{init}_n \xrightarrow{*} \gamma$. Writing $\pi = \gamma_0, \theta_1, \gamma_1, \dots, \gamma_{\ell-1}, \theta_\ell, \gamma_\ell$ with $\gamma_0 = \mathsf{init}_n$ and $\gamma_\ell = \gamma$, there exists $\rho\colon \sigma_\mathsf{init} \xrightarrow{*} \sigma$ such that $\mathsf{FW}(\mathsf{abst}(\gamma)) = \mathsf{FW}(\sigma)$ and, for every $i \in [0,\ell]$, $\mathsf{loc}(\mathsf{abst}(\gamma_i)) \subseteq \mathsf{loc}(\sigma)$.*

Proof of Lemma A.1. We construct an abstract execution that mimicks each move of the concrete execution π. We proceed by induction on the length of π, that is on the number of moves in its schedule $\mathsf{sched}(\pi)$. The base case, where π contains no moves, is trivial, letting $\sigma := \sigma_\mathsf{init}$.

Assume now that $|\pi| > 0$, and that the lemma holds for any concrete execution with at most $|\pi|$ moves. We isolate the last move of π to decompose π as $\mathsf{init}_n \xrightarrow{\mathsf{s}_p} \gamma_p \xrightarrow{\theta} \gamma$, with $\theta \in \mathsf{Moves}$, and write $\pi_p\colon \mathsf{init}_n \xrightarrow{\mathsf{s}_p} \gamma_p$. By induction hypothesis on π_p, there exists $\rho_p\colon \sigma_\mathsf{init} \xrightarrow{*} \sigma_p$ satisfying the property. Let us write $\theta = ((q,a,q'),k)$. We now claim that there exists $\sigma \in \Sigma$ such that $\sigma_p \xrightarrow{\theta} \sigma$, *i.e.*, θ is applicable from σ_p. Indeed, θ is applicable from γ_p, hence $\mathsf{loc}(\gamma_p)(q,k) > 0$ and by induction hypothesis $(q,k) \in \mathsf{loc}(\sigma_p)$; moreover:

- if $a = \mathsf{write}_\alpha(x)$, then $\mathsf{rg}_\alpha[k] \in \mathsf{FW}(\mathsf{abst}(\gamma_p)) = \mathsf{FW}(\sigma_p)$,
- if $a = \mathsf{read}_\alpha^{-i}(d_0)$, then $\mathsf{rg}_\alpha[k-i] \notin \mathsf{FW}(\mathsf{abst}(\gamma_p)) = \mathsf{FW}(\sigma_p)$,
- if $a = \mathsf{read}_\alpha^{-i}(x)$ with $x \neq d_0$, then $\mathsf{rg}_\alpha[k-i] \in \mathsf{FW}(\mathsf{abst}(\gamma_p)) = \mathsf{FW}(\sigma_p)$ and $\mathsf{data}_{\mathsf{rg}_\alpha[k-i]}(\gamma_p) = x$ hence there exist $q_1, q_2 \in Q$ such that $\mathsf{sched}(\rho_p)$ contains move $((q_1, \mathsf{write}_\alpha(x), q_2), k-i)$, and by induction hypothesis, $(q_1, k-i), (q_2, k-i) \in \mathsf{loc}(\sigma_p)$.

Therefore, there exists σ such that $\sigma_p \xrightarrow{\theta} \sigma$. Finally, σ satisfies the conditions of the lemma. First, since $\mathsf{FW}(\mathsf{abst}(\gamma_p)) = \mathsf{FW}(\sigma_p)$, we have $\mathsf{FW}(\mathsf{abst}(\gamma)) = \mathsf{FW}(\sigma)$. Second, $\mathsf{loc}(\sigma_p) \subseteq \mathsf{loc}(\sigma)$. Last, $\mathsf{loc}(\mathsf{abst}(\gamma_p)) \subseteq \mathsf{loc}(\sigma_p)$, and if a process goes to location (q,k) with move $\gamma_p \xrightarrow{\theta} \gamma$, then $(q,k) \in \mathsf{loc}(\sigma)$ thanks to the abstract step $\sigma_p \xrightarrow{\theta} \sigma$, and hence $\mathsf{loc}(\mathsf{abst}(\gamma)) \subseteq \mathsf{loc}(\sigma)$. ◀

Lemma A.1 directly entails the left-to-right implication of Theorem 7. The following lemma states the converse implication:

▶ **Lemma A.2.** *Let $\sigma \in \Sigma$ and $\rho\colon \sigma_\mathsf{init} \to \sigma$. There exist $n \in \mathbb{N}$, $\gamma \in \Gamma$ and $\pi\colon \mathsf{init}_n \to \gamma$ such that $\mathsf{FW}(\mathsf{abst}(\gamma)) = \mathsf{FW}(\sigma)$ and $\mathsf{loc}(\mathsf{abst}(\gamma)) = \mathsf{loc}(\sigma')$.*

Proof of Lemma A.2. Similarly to the previous proof, we would like to construct a concrete execution that mimicks each move of the (abstract) execution. To do so however, we need to handle two difficulties. First, in the concrete semantics and in contrast to the abstract one, a step can remove a location from the current configuration; we overcome this problem by adding a extra process in the given location, using the copycat property (Lemma 4). Second, in the concrete semantics, reading $x \in \mathrm{D} \setminus \{d_0\}$ from register ξ requires x to actually be the value stored in ξ, while the abstract semantics only requires a move writing x to ξ to be available; here again, we overcome this using Lemma 4 and Corollary 5 to add in the concrete execution a process that writes x to ξ.

Let $\rho\colon \sigma_{\mathsf{init}} \to \sigma$. We proceed by induction on the number of moves of ρ. If ρ contains no moves, then $\sigma = \sigma_{\mathsf{init}}$, and it suffices to take $\gamma = \mathsf{init}_1$.

Suppose now that $|\mathsf{sched}(\rho)| > 0$, and that the lemma holds for every execution of schedule of length at most $|\mathsf{sched}(\rho)|-1$, and write $\mathsf{sched}(\rho) = \mathsf{s}_p \cdot \theta$. By induction hypothesis, there exist $n \in \mathbb{N}$ and $\widetilde{\rho}_p\colon \mathsf{init}_n \xrightarrow{\mathsf{s}_p} \gamma_p$ such that $\mathsf{FW}(\mathsf{abst}(\gamma_p)) = \mathsf{FW}(\sigma_p)$ and $\mathsf{loc}(\sigma_p) \subseteq \mathsf{loc}(\mathsf{abst}(\gamma_p))$. Write $\theta = ((q, a, q'), k)$; we know that $\mathsf{loc}(\gamma_p)(q, k) > 0$. By Lemma 4, we can modify γ_p so that $\mathsf{loc}(\gamma_p)(q, k) > 1$ (this may require to increase the number of processes n by 1). It remains to prove that there exists γ such that $\gamma_p \to \gamma$, $\mathsf{FW}(\sigma) = \mathsf{FW}(\mathsf{abst}(\gamma))$ and $\mathsf{loc}(\sigma) = \mathsf{loc}(\mathsf{abst}(\gamma))$. We split cases, depending on the action a of θ:

- If $a = \mathsf{Inc}$, consider γ such that $\gamma_p \xrightarrow{\theta} \gamma$ (this is possible because $(q, k) \in \mathsf{loc}(\gamma_p)$); we then have $(q', k{+}1) \in \mathsf{supp}(\gamma)$ but also $(q, k) \in \mathsf{supp}(\gamma)$ (because $\mathsf{loc}(\gamma_p)(q, k) > 1$) hence $\mathsf{loc}(\sigma) = \mathsf{loc}(\mathsf{abst}(\gamma))$ and $\mathsf{FW}(\mathsf{abst}(\gamma)) = \mathsf{FW}(\mathsf{abst}(\gamma_p)) = \mathsf{FW}(\sigma_p) = \mathsf{FW}(\sigma)$.
- If $a = \mathsf{write}_\alpha(x)$, as above consider γ such that $\gamma_p \xrightarrow{\theta} \gamma$; we then have that $\mathsf{data}_{\mathsf{rg}_\alpha[k]}(\gamma) = x$ hence $\mathsf{rg}_\alpha[k] \in \mathsf{FW}(\mathsf{abst}(\gamma))$, allowing to prove that $\mathsf{FW}(\mathsf{abst}(\gamma)) = \mathsf{FW}(\mathsf{abst}(\gamma_p)) \cup \{\mathsf{rg}_\alpha[k]\} = \mathsf{FW}(\sigma_p) \cup \{\mathsf{rg}_\alpha[k]\} = \mathsf{FW}(\sigma)$.
- If $a = \mathsf{read}_\alpha^{-i}(d_0)$, thanks to $\sigma_p \xrightarrow{\theta} \sigma$, we have $\mathsf{rg}_\alpha[k{-}i] \notin \mathsf{FW}(\sigma_p)$ hence $\mathsf{data}_{\mathsf{rg}_\alpha[k-i]}(\gamma_p) = d_0$, hence it is again possible to consider γ such that $\gamma_p \xrightarrow{\theta} \gamma$.
- If $a = \mathsf{read}_\alpha^{-i}(x)$, because $\sigma_p \xrightarrow{\theta} \sigma$, there exists $(q_1, \mathsf{write}_\alpha(x), q_2) \in \Delta$ such that $(q_1, k{-}i), (q_2, k{-}i) \in \mathsf{loc}(\sigma_p)$. Since $\mathsf{loc}(\sigma_p) = \mathsf{loc}(\mathsf{abst}(\gamma_p))$, $\mathsf{loc}(\gamma_p)(q_1, k{-}i) > 0$ and thanks to Lemma 4 we can change γ_p in order to have $\mathsf{loc}(\gamma_p)(q_1, k{-}i) > 1$. By writing $\theta' := ((q_1, \mathsf{write}_\alpha(x), q_2), k{-}i)$, consider γ such that $\gamma_p \xrightarrow{\theta' \cdot \theta} \gamma$. Since $\mathsf{loc}(\gamma_p)(q_1, k{-}i) > 1$, we have $(q_1, k{-}i), (q_2, k{-}i) \in \mathsf{supp}(\gamma)$. Therefore, $\mathsf{loc}(\mathsf{abst}(\gamma)) = \mathsf{loc}(\mathsf{abst}(\gamma_p)) \cup \{(q, k)\} = \mathsf{loc}(\mathsf{abst}(\gamma_p)) \cup \{(q, k)\} = \mathsf{loc}(\sigma_p) \cup \{(q, k)\} = \mathsf{loc}(\sigma)$. Moreover, since $\sigma_p \xrightarrow{\theta} \sigma$, we have $\mathsf{rg}_\alpha[k{-}i] \in \mathsf{FW}(\sigma)$ hence $\mathsf{FW}(\mathsf{abst}(\gamma)) = \mathsf{FW}(\mathsf{abst}(\gamma_p)) \cup \{\mathsf{rg}_\alpha[k{-}i]\} = \mathsf{FW}(\sigma_p) \cup \{\mathsf{rg}_\alpha[k{-}i]\} = \mathsf{FW}(\sigma)$.

This ends the proof of the right-to-left implication of Theorem 7 and of the theorem itself. ◀

A.3 Upper bound on cutoff

▶ **Corollary 8.** *If there exists $k \in \mathbb{N}$ such that (q_{err}, k) is coverable, then, letting $N = 2|Q|(k{+}1){+}1$, there exists $\pi : \mathsf{init}_N \xrightarrow{*} \gamma$ such that $(q_{\mathsf{err}}, k) \in \mathsf{loc}(\gamma)$.*

Proof. If q_{err} is coverable at round k in the concrete semantics, then thanks to Theorem 7, there exist $\sigma \in \Sigma$ and $\rho\colon \sigma_{\mathsf{init}} \xrightarrow{*} \sigma$ such that $(q_{\mathsf{err}}, k) \in \mathsf{loc}(\sigma)$. Let $\mathsf{s}' = \mathsf{proj}_{[0,k]}(\mathsf{sched}(\rho))$ be the schedule obtained from $\mathsf{sched}(\rho)$ by removing all moves on rounds after round k. We have $\sigma_{\mathsf{init}} \xrightarrow{\mathsf{s}'} \sigma'$ with $(q_{\mathsf{err}}, k) \in \mathsf{loc}(\sigma')$. Let now s'' be the schedule obtained from s' restricting to moves that cover a new location, *i.e.* a location that was not covered by previous moves. We have that $\sigma_{\mathsf{init}} \xrightarrow{\mathsf{s}''} \sigma''$ with $\mathsf{loc}(\sigma'') = \mathsf{loc}(\sigma')$, and $|\mathsf{s}''| \leq |Q|(k{+}1)$.

To conclude, observe that in the proof of Lemma A.2, for $|\mathsf{sched}(\rho)| = 0$ we let $n = 1$ (a single process suffices) and we later increased the value of n by at most 2 per move in $\mathsf{sched}(\rho)$ (we applied Lemma 4 at most twice). Applying this observation to $\rho'' : \sigma_{\mathsf{init}} \xrightarrow{\mathsf{s}''} \sigma''$ implies that, for $N := 2|Q|(k+1)+1$, there exists $\gamma \in \mathsf{Reach_c}(\mathsf{init}_N)$ such that $(q_{\mathsf{err}}, k) \in \mathsf{loc}(\gamma)$. ◀

B Proofs and details for Section 3

B.1 Proof of Proposition 10

▶ **Proposition 10.** *In a register protocol $\mathcal{P}$ with $\mathsf{v} = 0$ and $\mathsf{d} = 1$, for any finite set L of coverable locations, there exists $n \in \mathbb{N}$ and an execution $\rho : \sigma_{\mathsf{init}} \xrightarrow{*} \sigma$ such that, for all $(q, k) \in L$, $(q, k) \in \mathsf{loc}(\sigma)$.*

Proof. It suffices to prove the following statement: for all $\rho_1 : \sigma_{\mathsf{init}} \xrightarrow{*} \sigma_1$ and $\rho_2 : \sigma_{\mathsf{init}} \xrightarrow{*} \sigma_2$, there exists $\rho : \sigma_{\mathsf{init}} \xrightarrow{*} \sigma$ such that $\mathsf{loc}(\sigma_1) \cup \mathsf{loc}(\sigma_2) \subseteq \mathsf{loc}(\sigma)$.

Thanks to $\mathsf{v} = 0$, moves on round k can only read the register of round k, hence all executions can be reorganised with their moves on round 0 first, then their moves on round 1, and so on. Let K the maximum round of moves in ρ_1 and ρ_2, and proceed by induction on K.

Suppose first $K = 0$: ρ_1 and ρ_2 only contain moves on round 0. If neither ρ_1 nor ρ_2 write on $\mathsf{rg}[0]$, one can simply concatenate the schedules. Otherwise, suppose that ρ_1 writes on $\mathsf{rg}[0]$, and write $\mathsf{sched}(\rho_1) = \mathsf{s}_1 \cdot \theta_1 \cdot \mathsf{s}'_1$ where θ_1 is the first write in $\mathsf{sched}(\rho_1)$. Consider the following schedule: $\mathsf{s} := \mathsf{s}_1 \cdot \mathsf{sched}(\rho_2) \cdot \theta_1 \cdot \mathsf{s}'_1$. We have that:

- s_1 is a prefix of $\mathsf{sched}(\rho_1)$ which is valid from σ_{init};
- s_1 does not write and $\mathsf{sched}(\rho_2)$ is valid from σ_{init} hence $\mathsf{s}_1 \cdot \mathsf{sched}(\rho_2)$ is valid from σ_{init};
- $\mathsf{s}_1 \cdot \mathsf{sched}(\rho_1)$ only writes on register 0, which is overwritten by θ_1, hence s is valid from σ_{init}.

Suppose that ρ_1 and ρ_2 have moves on rounds 0 to $K + 1$, and that the property is true for K. Reorganize ρ_1 and ρ_2 so that they start with moves on round 0, followed by moves on round 1 and so on. Decompose ρ_1 into $\rho_{1,\leq K} : \sigma_{\mathsf{init}} \xrightarrow{*} \sigma'_1$ and $\rho_{1,K+1} : \sigma'_1 \xrightarrow{*} \sigma_1$, where $\rho_{1,\leq K}$ only has moves on rounds $\leq K$ and $\rho_{1,K+1}$ only has moves on round $K + 1$, and similarly for ρ_2. By induction hypothesis, there exists $\rho_{\leq K} : \sigma_{\mathsf{init}} \xrightarrow{*} \sigma'$ with only moves on rounds $\leq K$ such that $\mathsf{loc}(\sigma'_1) \cup \mathsf{loc}(\sigma'_2) \subseteq \mathsf{loc}(\sigma')$. Since σ' has register $\mathsf{rg}[K + 1]$ blank, $\mathsf{sched}(\rho_{1,K+1})$ and $\mathsf{sched}(\rho_{2,K+1})$ are both applicable from σ' . By reapplying the reasoning of $K = 0$ onto $\rho_{1,K+1}$ and $\rho_{2,K+1}$, which may only write on $\mathsf{rg}[k+1]$, we obtain an execution $\rho_{K+1} : \sigma' \xrightarrow{*} \sigma$ with $\mathsf{loc}(\sigma_1) \cup \mathsf{loc}(\sigma_2) \subseteq \mathsf{loc}(\sigma)$. Combining $\rho_{\leq K}$ with ρ_{K+1} gives the desired execution, concluding the proof.

Note that it is also possible to see Proposition 10 as a consequence of Lemma 17; indeed, with $\mathsf{v} = 0$ and $\mathsf{d} = 1$, the condition of equality of first-write order projections becomes that ρ_1 and ρ_2 have to write to the same set of registers, which we can always enforce by adding dummy writes to our protocol. ◀

B.2 Binary counter

Recall the protocol $\mathcal{BC}_m$ from Figure 3 that encodes a binary counter over m bits. We now prove that 2^{m-1} rounds are needed and sufficient to cover q_{err}.

▶ **Proposition 11.** *Let $k \in [0, 2^{m-1}]$. Location (q_{err}, k) is coverable in $\mathcal{BC}_m$ iff $k = 2^{m-1}$.*

Proof. Thanks to Proposition 10, when $\mathsf{v} = 0$ and $\mathsf{d} = 1$, all coverable locations are compatible, for every finite number of coverable locations, there exists an execution that covers all these locations. We therefore do not have to worry about with which execution a location is coverable, and we will simply write that a location *is coverable* or *is not coverable* and that a symbol *can be written* or *cannot be written* to a given register.

The set of coverable locations can be characterised as follows:

▶ **Lemma B.3.** *Let $i \in [1, m]$, $k \in [0, 2^{m-1}]$ and r the remainder of the Euclidean division of k by 2^i. In $\mathcal{BC}_m$, one has the following equivalences:*

$$(q_{i,0}, k) \text{ is coverable} \iff 0 \leq r \leq 2^{i-1} - 1 \;;$$

$$(q_{i,1}, k) \text{ is coverable} \iff 2^{i-1} \leq r \leq 2^i - 1 \;.$$

Proof of Lemma B.3. The proof is by induction on pairs (k, i), ordered lexicographically.

Observe first that, for all $i \in [1, m]$, $(q_{i,0}, 0)$ is coverable and $(q_{i,1}, 0)$ is not. Moreover, for all $k \in [0, 2^m]$, $(q_{1,0}, k)$ is coverable exactly for even k, and $(q_{1,1}, k)$ is coverable exactly for odd k.

Let $k > 0$, $i \in [2, m]$ and suppose that the lemma holds for all pairs (k', i') with $k' < k$ or $k' = k$ and $i' < i$. The only way to write move_i to $\mathsf{rg}[k]$ is when a process moves from $(q_{i-1,1}, k-1)$ to $(q_{i-1,0}, k)$. By induction hypothesis, this means that the remainder of the Euclidean division of $k-1$ by 2^{i-1} is in $[2^{i-2}, 2^{i-1} - 1]$ and the remainder of the Euclidean division of k by 2^{i-1} is in $[0, 2^{i-2}]$, which is equivalent to k being divisible by 2^{i-1}. To sum up, move_i can be written to $\mathsf{rg}[k]$ exactly when k is a multiple of 2^{i-1}. Similarly, wait_i can be written to $\mathsf{rg}[k]$ exactly when k is not divisible by 2^{i-1}.

Let r be the remainder of the Euclidean division of k by 2^i. We distinguish cases according to the value of r:

- if $r = 0$, then the remainder of $k-1$ by 2^i is in $[2^{i-1}, 2^i - 1]$ hence $(q_{i,1}, k-1)$ can be covered and $(q_{i,0}, k-1)$ cannot; since k is divisible by 2^{i-1}, move_i can be written to $\mathsf{rg}[k]$ but wait_i cannot, so that $(q_{i,0}, k)$ can be covered and $(q_{i,1}, k)$ cannot;
- if $1 \leq r \leq 2^{i-1} - 1$, then the remainder of $k-1$ by 2^i is in $[0, 2^{i-1} - 1]$ hence $(q_{i,0}, k-1)$ can be covered and $(q_{i,1}, k-1)$ cannot; since k is not divisible by 2^{i-1}, wait_i can be written to $\mathsf{rg}[k]$ but move_i cannot, so that $(q_{i,0}, k)$ can be covered and $(q_{i,1}, k)$ cannot;
- if $r = 2^{i-1}$, then the remainder of $k-1$ by 2^i is in $[0, 2^{i-1} - 1]$ hence $(q_{i,0}, k-1)$ can be covered and $(q_{i,1}, k-1)$ cannot; since k is divisible by 2^{i-1}, move_i can be written to $\mathsf{rg}[k]$ but wait_i cannot, so that $(q_{i,1}, k)$ can be covered and $(q_{i,0}, k)$ cannot;
- if $2^{i-1} + 1 \leq r \leq 2^i - 1$, then the remainder of $k-1$ by 2^i is in $[2^{i-1}, 2^i - 1]$ hence $(q_{i,1}, k-1)$ can be covered and $(q_{i,0}, k-1)$ cannot; since k is divisible by 2^{i-1}, wait_i can be written to $\mathsf{rg}[k]$ but move_i cannot, $(q_{i,1}, k)$ can be covered and $(q_{i,0}, k)$ cannot. ◀

Applied with $i = m$, Lemma B.3 implies Proposition 9: indeed the only value k in $[0, 2^{m-1}]$ such that the Euclidian division of k by 2^m yields a remainder of at least 2^{m-1} is 2^{m-1}. ◀

B.3 Compatibility and first-write orders

Let us introduce a few more notions related to first-write orders. Given a sequence of registers $\mathsf{f} = \xi_1 : \ldots : \xi_\ell$, a *swap* of f is any sequence $\xi_1 : \ldots : \xi_{i-1} : \xi_{i+1} : \xi_i : \xi_{i+2} : \ldots \xi_\ell$ with $\mathsf{round}(\xi_i) > \mathsf{round}(\xi_{i+1}) + \mathsf{v}$; in words, a swap is obtained from f by swapping two registers more than v rounds apart to put the one with earliest round first. A finite sequence of registers f is *swap-proof* when no swap is possible from f.

We first prove that executions with same first-write orders are compatible.

▶ **Lemma 16.** *Let* $\rho_1 \colon \sigma_{\mathsf{init}} \xrightarrow{*} \sigma_1$ *and* $\rho_2 \colon \sigma_{\mathsf{init}} \xrightarrow{*} \sigma_2$ *be two executions such that* $\mathsf{fwo}(\rho_1) = \mathsf{fwo}(\rho_2)$*. Then, there exists* $\rho \colon \sigma_{\mathsf{init}} \xrightarrow{*} \sigma$ *such that* $\mathsf{loc}(\sigma) = \mathsf{loc}(\sigma_1) \cup \mathsf{loc}(\sigma_2)$*,* $\mathsf{FW}(\sigma) = \mathsf{FW}(\sigma_1) = \mathsf{FW}(\sigma_2)$*, and* $\mathsf{fwo}(\rho) = \mathsf{fwo}(\rho_1) = \mathsf{fwo}(\rho_2)$*.*

Proof. To establish the result, the only problematic moves are reads from blank registers and first writes; indeed, if ρ, ρ' leave all registers blank, one can simply concatenate their schedules into $\mathsf{sched}(\rho) \cdot \mathsf{sched}(\rho')$. To overcome the difficulty of first writes, we explain below how to interleave ρ and ρ', considering parts of ρ and ρ' where the sets of blank registers agree.

In this proof, for two configurations $\sigma, \sigma' \in \Sigma$ such that $\mathsf{FW}(\sigma) = \mathsf{FW}(\sigma')$, we write $\sigma \cup \sigma'$ for the configuration τ defined by $\mathsf{loc}(\tau) = \mathsf{loc}(\sigma) \cup \mathsf{loc}(\sigma')$ and $\mathsf{FW}(\tau) = \mathsf{FW}(\sigma) = \mathsf{FW}(\sigma')$.

Consider ρ_1 and ρ_2 as in the statement. We let $\mathsf{f} = \xi_1 : \ldots : \xi_\ell$ with $\xi_1, \ldots, \xi_\ell \in \mathsf{Reg}$ be the first-write order of both ρ_1 and ρ_2. The two executions are then "decomposed" according to their first-write order: $\rho_1 = \rho_{1,0} \cdot \ldots \cdot \rho_{1,\ell}$ and $\rho_2 = \rho_{2,0} \cdot \ldots \cdot \rho_{2,\ell}$. Formally, for every $i \in [0, \ell]$, $\rho_{1,i}$ and $\rho_{2,i}$ do not write to registers ξ_{i+1} to ξ_ℓ, and do not read d_0 from registers ξ_1 to ξ_i. Also, for every $i \in [1, \ell]$, $\rho_{1,i}$ and $\rho_{2,i}$ start with a write to register ξ_i.

For every $i \in [1, \ell]$, we consider the following prefix executions, $\rho_{1,0} \cdot \ldots \cdot \rho_{1,i} \colon \sigma_{\mathsf{init}} \xrightarrow{*} \sigma_{1,i}$ and $\rho_{2,0} \cdot \ldots \cdot \rho_{2,i} \colon \sigma_{\mathsf{init}} \xrightarrow{*} \sigma_{2,i}$. More precisely, $\rho_{1,0} \cdot \ldots \cdot \rho_{1,i}$ (resp. $\rho_{2,0} \cdot \ldots \cdot \rho_{2,i}$) is the prefix execution of ρ_1 (resp. of ρ_2) stopping just before the first write to ξ_{i+1}. Note that, for every $i \in [0, \ell]$, $\mathsf{fwo}(\rho_{1,0} \cdot \ldots \cdot \rho_{1,i}) = \mathsf{fwo}(\rho_{2,0} \cdot \ldots \cdot \rho_{2,i})$ hence $\mathsf{FW}(\sigma_{1,i}) = \mathsf{FW}(\sigma_{2,i})$ and $\sigma_{1,i} \cup \sigma_{2,i}$ is defined.

We now prove the following property by induction on i: there exists an execution $\widetilde{\rho}_i \colon \sigma_{\mathsf{init}} \to \sigma_{1,i} \cup \sigma_{2,i}$ such that $\mathsf{fwo}(\widetilde{\rho}_i) = \mathsf{fwo}(\rho_{1,0} \cdot \ldots \cdot \rho_{1,i}) = \mathsf{fwo}(\rho_{2,0} \cdot \ldots \cdot \rho_{2,i})$.

Assume the property holds for $i < \ell$ and let us prove it for $i{+}1$. By induction hypothesis, there exists $\widetilde{\rho}_i \colon \sigma_{\mathsf{init}} \to \sigma_{1,i} \cup \sigma_{2,i}$. Letting $\mathsf{s}_1 := \mathsf{sched}(\rho_{1,i+1})$ and $\mathsf{s}_2 := \mathsf{sched}(\rho_{2,i+1})$, we claim that $\sigma_{1,i} \cup \sigma_{2,i} \xrightarrow{\mathsf{s}_1 \cdot \mathsf{s}_2} \sigma_{1,i+1} \cup \sigma_{2,i+1}$. First, $\sigma_{1,i} \xrightarrow{\mathsf{s}_1} \sigma_{1,i+1}$. Since $\mathsf{FW}(\sigma_{1,i} \cup \sigma_{2,i}) = \mathsf{FW}(\sigma_i) = \{\xi_1, \ldots, \xi_i\}$, $\sigma_{1,i} \cup \sigma_{2,i} \xrightarrow{\mathsf{s}_1} \sigma_{1,i+1} \cup \sigma_{2,i}$. Moreover, $\mathsf{FW}(\sigma_{1,i+1} \cup \sigma_{2,i}) = \{\xi_1, \ldots, \xi_i, \xi_{i+1}\}$ and since s_2 starts with a write to register ξ_{i+1}, it never reads d_0 from ξ_{i+1} hence $\sigma_{1,i+1} \cup \sigma_{2,i} \xrightarrow{\mathsf{s}_2} \sigma_{1,i+1} \cup \sigma_{2,i+1}$. In the end, letting $\widetilde{\mathsf{s}_i} = \mathsf{sched}(\widetilde{\rho}_i)$, we have $\widetilde{\rho}_{i+1} \colon \sigma_{\mathsf{init}} \xrightarrow{\widetilde{\mathsf{s}_i} \cdot \mathsf{s}_1 \cdot \mathsf{s}_2} \sigma_{1,i+1} \cup \sigma_{2,i+1}$; we also have $\mathsf{fwo}(\widetilde{\rho}_{i+1}) = \mathsf{fwo}(\rho_{1,0} \cdot \ldots \cdot \rho_{1,i+1}) = \mathsf{fwo}(\rho_{2,0} \cdot \ldots \cdot \rho_{2,i+1})$ concluding the proof. ◀

▶ **Lemma 17.** *Let* $\rho_1 \colon \sigma_{\mathsf{init}} \xrightarrow{*} \sigma_1$ *and* $\rho_2 \colon \sigma_{\mathsf{init}} \xrightarrow{*} \sigma_2$ *be two executions of a register protocol with visibility range* v*, such that, for all* $k \in \mathbb{N}$*,* $\mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{fwo}(\rho_1)) = \mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{fwo}(\rho_2))$*. Then, there exists* $\rho \colon \sigma_{\mathsf{init}} \xrightarrow{*} \sigma$ *such that* $\mathsf{loc}(\sigma) = \mathsf{loc}(\sigma_1) \cup \mathsf{loc}(\sigma_2)$*,* $\mathsf{FW}(\sigma) = \mathsf{FW}(\sigma_1) = \mathsf{FW}(\sigma_2)$*, and, for all* $k \in \mathbb{N}$*,* $\mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{fwo}(\rho)) = \mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{fwo}(\rho_1)) = \mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{fwo}(\rho_2))$*.*

Proof. To prove Lemma 17, we first prove that ρ_1 and ρ_2 can be replaced with executions whose first-write order is swap-proof, while preserving their last configuration. This relies on the following lemma:

▶ **Lemma B.4.** *If* $\rho \colon \sigma \xrightarrow{*} \tau$ *satisfies* $\mathsf{fwo}(\rho) = p : \xi : \xi' : s$ *with* p, s *sequences of registers,* $\xi, \xi' \in \mathsf{Reg}$ *and* $\mathsf{round}(\xi) > \mathsf{round}(\xi'){+}\mathsf{v}$*, then there exists* $\widetilde{\rho} \colon \sigma \xrightarrow{*} \tau$ *with* $\mathsf{fwo}(\widetilde{\rho}) = p : \xi' : \xi : s$*.*

Proof of Lemma B.4. Write $k := \mathsf{round}(\xi)$ and $k' := \mathsf{round}(\xi')$ for the rounds of registers ξ and ξ'; by assumption, $k > k'{+}\mathsf{v}$. The prefix of ρ before the first write to ξ and the suffix of ρ after the first write to ξ' will be preserved in $\widetilde{\rho}$. Therefore, we focus on the middle part, and suppose that $\mathsf{fwo}(\rho) = \xi : \xi'$ and that $\mathsf{sched}(\rho)$ ends with a first write to ξ'. Decompose $\mathsf{sched}(\rho) = \theta \cdot \mathsf{s} \cdot \theta'$ where θ is the first write to ξ and θ' is the first write to ξ'. Let $\tilde{\mathsf{s}} := \mathsf{s}_{<k} \cdot \theta' \cdot \theta \cdot \mathsf{s}_{\geq k}$, where $\mathsf{s}_{<k} := \mathsf{proj}_{[0,k-1]}(\mathsf{s})$ and $\mathsf{s}_{\geq k} := \mathsf{proj}_{[k+1,+\infty[}(\mathsf{s})$. We claim that $\tilde{\mathsf{s}}$ is applicable from σ. Indeed:

- $\mathsf{s}_{<k} \cdot \theta'$ is applicable from σ because $\mathsf{proj}_{[0,k-1]}(\mathsf{sched}(\rho)) = \mathsf{s}_{<k} \cdot \theta'$ and moves on rounds smaller than k are not impacted by what happens on rounds larger than or equal to k;
- θ is applicable after $\mathsf{s}_{<k} \cdot \theta'$ because it is a write action applicable from σ;
- $\mathsf{s}_{\geq k}$ is applicable after $\mathsf{s}_{<k} \cdot \theta' \cdot \theta$ because it is applicable after $\mathsf{s}_{<k} \cdot \theta$ ($\mathsf{s}_{<k}$ only adds new locations, it does not first write) and since $k' < k-\mathsf{v}$, moves of $\mathsf{s}_{\geq k}$ cannot see the first write to ξ'.

Let $\widetilde{\rho}\colon \sigma \xrightarrow{\tilde{\mathsf{s}}} \tilde{\tau}$. Since $\tilde{\mathsf{s}}$ contains the same moves as $\mathsf{sched}(\rho)$, $\tilde{\tau} = \tau$. Finally, $\mathsf{fwo}(\widetilde{\rho}) = \xi' : \xi$, which concludes the proof. ◀

Lemma B.4 states that one can perform swaps in the first-write order of an execution while preserving the final configuration. To prove Lemma 17, we iteratively apply Lemma B.4 on ρ_1 and ρ_2 until obtaining a swap-proof first-write order. The following lemma states that this iterative process yields a unique swap-proof first-write order when starting with ρ_1 or ρ_2.

▶ **Lemma B.5.** *Let* f *and* g *be two finite sequences of registers such that, for all* $k \in \mathbb{N}$, $\mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{f}) = \mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{g})$. *There exists a swap-proof sequence of registers* h *that can be obtained by iteratively applying swaps from* f *and also by iteratively applying swaps from* g.

Proof of Lemma B.5. Swaps decrease the number of inversions, *i.e.*, of pairs of registers (ξ, ξ') with $\mathsf{round}(\xi) > \mathsf{round}(\xi')-\mathsf{v}$ and ξ precedes ξ'. Therefore, iteratively applying swaps from f one obtains a swap-proof sequence of registers h_f after finitely many swaps. Similarly, iteratively applying swaps from g on obtains a swap-proof sequence of registers h_g. Let us prove that $\mathsf{h}_\mathsf{f} = \mathsf{h}_\mathsf{g}$.

Observe first that swaps preserve the projection of windows of size v. Therefore, for all $k \in \mathbb{N}$, $\mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{h}_\mathsf{f}) = \mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{f}) = \mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{g}) = \mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{h}_\mathsf{g})$.

We now prove by induction on the maximum round K present in h_f and h_g that $\mathsf{h}_\mathsf{f} = \mathsf{h}_\mathsf{g}$. The degenerate case $\mathsf{h}_\mathsf{f} = \mathsf{h}_\mathsf{g} = \varepsilon$ is trivial.

Now, suppose that h_f and h_g are not empty, and write K the maximum round of registers in h_f and h_g. Write $\mathsf{h}'_\mathsf{f} := \mathsf{proj}_{[0,K-1]}(\mathsf{h}_\mathsf{f})$ and $\mathsf{h}'_\mathsf{g} := \mathsf{proj}_{[0,K-1]}(\mathsf{h}_\mathsf{g})$; as observed above, we have $\mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{h}'_\mathsf{f}) = \mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{h}'_\mathsf{g})$ for all $k \in \mathbb{N}$. We claim that h'_f and h'_g are swap-proof. Indeed, if h'_f contained a factor $\xi : \xi'$ with $\mathsf{round}(\xi) > \mathsf{round}(\xi')+\mathsf{v}$, then h_f has a factor $\xi : p : \xi'$ where p is a non-empty sequence of registers of round K. Moreover, since K is the maximum round in h_f, $\mathsf{round}(\xi') < K-\mathsf{v}$ hence ξ' and the last register of p contradict h_f being swap-proof. The proof for h'_g is identical.

Applying the induction hypothesis to h'_f and h'_g, we obtain $\mathsf{h}'_\mathsf{f} = \mathsf{h}'_\mathsf{g}$. Towards a contradiction, suppose there exist $\xi, \xi' \in \mathsf{Reg}$ such that ξ appears before ξ' in h_f and after ξ' in h_g. Then either $\mathsf{round}(\xi) = K$ or $\mathsf{round}(\xi') = K$; wlog, suppose $\mathsf{round}(\xi) = K$ and $\mathsf{round}(\xi') < K-\mathsf{v}$. Letting $\xi : p : \xi'$ the factor of f between ξ and ξ', we can suppose that all registers in p are on rounds strictly less than K, otherwise replace ξ by the last register in p on round K. Since $\mathsf{h}'_\mathsf{f} = \mathsf{h}'_\mathsf{g}$, all registers in p are before ξ' in h'_g, hence before ξ; therefore the first register in p is on a round stricly less than $K-\mathsf{v}$. This is a contradiction, since it would imply the existence of a possible swap in h_f. ◀

Thanks to Lemma B.5, when applying iteratively swaps on $\mathsf{fwo}(\rho_1)$ and $\mathsf{fwo}(\rho_2)$, we obtain the same swap-proof sequence of registers h. Let us denote by $\mathsf{fwo}(\rho_1) = \mathsf{f}_1, \mathsf{f}_2, \ldots, \mathsf{f}_\ell = \mathsf{h}$ and $\mathsf{fwo}(\rho_2) = \mathsf{g}_1, \mathsf{g}_2, \ldots, \mathsf{g}_{\ell'} = \mathsf{h}$ the sequences of first-write orders corresponding to these transformations. Thus, for every $i \in [1, \ell-1]$, f_{i+1} is a swap from f_i, and for every $j \in [1, \ell'-1]$, g_{j+1} is a swap from g_j. Thanks to Lemma B.4, there exist $\rho_{1,1}, \ldots, \rho_{1,\ell}$ such that, for every $i \in [1, \ell]$, $\rho_{1.i}\colon \sigma_{\mathsf{init}} \xrightarrow{*} \sigma_1$ and $\mathsf{fwo}(\rho_{1,i}) = \mathsf{f}_i$. Similarly, there exist $\rho_{2,1}, \ldots, \rho_{2,\ell'}$ such that, for every $i \in [1, \ell']$, $\rho_{2,i} : \sigma_{\mathsf{init}} \xrightarrow{*} \sigma_2$ and $\mathsf{fwo}(\rho_{2,i}) = \mathsf{g}_i$. Applying Lemma 16 to $\rho_{1,\ell}$ and $\rho_{2,\ell'}$ concludes the proof of Lemma 17. ◀

B.4 Characterisation of the sets $S_k(F_k)$ computed in Algorithm 2

▶ **Theorem 21.** *For $\mathcal{F} = (F_k)_{k\in\mathbb{N}}$ a family of projections, if the $\mathcal{F}$-computation of Algorithm 2 is non-rejecting, then the computed sets $(S_k(F_k))_{k\in\mathbb{N}}$ satisfy, for all $k \in \mathbb{N}$, $S_k(F_k) = Q\mathsf{cover}(\mathcal{F}, k)$. Also, for any execution ρ from σ_{init}, letting $\mathcal{F} = (\mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{fwo}(\rho)))_{k\geq 0}$, the $\mathcal{F}$-computation of Algorithm 2 is non-rejecting.*

Proof. In this proof, given $\mathcal{F} = (F_k)_{k\in\mathbb{N}}$, $k \in \mathbb{N}$ and f a prefix of F_k, we consider the *partial computation* of Algorithm 2 up until iteration (k, f), that corresponds to the computation that chooses projections F_r for all $r \leq k$ and that artificially stops at the end of iteration (k, f).

We define, for every $k \in \mathbb{N}$ and for every f prefixes of F_k, the set

$$\mathcal{R}_k(f) := \{q \mid \exists\sigma \in \Sigma,\ (q, k) \in \mathsf{loc}(\sigma),\ \exists\rho : \sigma_{\mathsf{init}} \xrightarrow{*} \sigma,\ \forall r \leq k, \mathsf{proj}_{[r-\mathsf{v},r]}(\mathsf{fwo}(\rho)) = \phi^k_r(f)\}$$

of states that can be covered at round k with an execution consistent with f.

For all $k \in \mathbb{N}$ and $\sigma \in \Sigma$, we let $\mathsf{st}_k(\sigma) := \{q \in Q \mid (q, k) \in \mathsf{loc}(\sigma)\}$. Given two executions $\rho = \sigma_0, \theta_1, \ldots, \sigma_\ell$, a *prefix execution* of ρ is an execution of the form $\rho_p := \sigma_0, \theta_1, \ldots, \sigma_{i_p}$ with $i_p \leq \ell$; similarly, $\rho_s := \sigma_{i_p}, \theta_{i_p+1}, \ldots \sigma_\ell$ is a *suffix execution* of ρ, and we write $\rho = \rho_p \cdot \rho_s$.

Let us prove that, for all $k \in \mathbb{N}$, for all f prefixes of F_k, $\mathcal{R}_k(f) = S_k(f)$. First, the following technical lemma states that any execution that satisfies the first-write order constraints of $\mathcal{R}_k(f)$ with $f = g : \epsilon$ admits a prefix execution satisfying the first-write order constraints of $\mathcal{R}_k(g)$.

▶ **Lemma B.6.** *Let $k \in \mathbb{N}$, f, g prefixes of F_k such that g is a strict prefix of f. Let an abstract execution $\rho : \sigma_{\mathsf{init}} \xrightarrow{*} \sigma$ such that, for all $r \leq k$, $\mathsf{proj}_{[r-\mathsf{v},r]}(\mathsf{fwo}(\rho)) = \phi^k_r(f)$. There exists ρ_p a prefix execution of f such that, for all $r \leq k$, $\mathsf{proj}_{[r-\mathsf{v},r]}(\mathsf{fwo}(\rho_p)) = \phi^k_r(g)$ and, decomposing $\rho = \rho_p \cdot \rho_s$, ρ_s starts with a first write to the first register in f that is not in g.*

Proof of Lemma B.6. Let $\mathsf{f} := \mathsf{fwo}(\rho)$. According to the proof of Lemma 17, we can assume $\mathsf{fwo}(\rho)$ to be swap-proof (see the definition of this notion in Subsection B.3). Moreover, wlog we can always assume that $\mathsf{fwo}(\rho)$ only has registers of rounds $\leq k$, by removing from ρ all moves on rounds $> k$.

Let $\mathsf{g} : \xi$, with ξ a register of round $r_\xi := \mathsf{round}(\xi)$, the shortest prefix of f such that, for all $r \leq k$, $\mathsf{proj}_{[r-\mathsf{v},r]}(\mathsf{g})$ is a prefix of $\phi^k_r(g)$, but ξ is not in $\phi^k_{r_\xi}(g)$. We claim that $r_\xi \geq k - \mathsf{v}$. Indeed, otherwise, $\phi^k_{r_\xi+\mathsf{v}}(g) : \xi = \mathsf{proj}_{[r_\xi,r_\xi+\mathsf{v}]}(\mathsf{g} : \xi)$ would be a prefix of $F_{r_\xi+\mathsf{v}}$ (since $\mathsf{proj}_{[r_\xi,r_\xi+\mathsf{v}]}(\mathsf{f}) = \phi^k_{r_\xi+\mathsf{v}}(f)$ is a prefix of $F_{r_\xi+\mathsf{v}}$) that coincides with $\phi^k_{r_\xi+\mathsf{v}+1}(g)$ on common rounds, contradicting the maximality of $\phi^k_{r_\xi+\mathsf{v}}(g)$.

Towards a contradiction, suppose now that there exists $s \leq k$ such that $\mathsf{proj}_{[s-\mathsf{v},s]}(\mathsf{g})$ is a strict prefix of $\phi^k_s(g)$. Write $\phi^k_s(g) = \mathsf{proj}_{[s-\mathsf{v},s]}(\mathsf{g}) : \xi' : h$ with ξ' a register and h a sequence of registers. Since $\mathsf{proj}_{[s-\mathsf{v},s]}(\mathsf{f}) = \phi^k_s(f)$, ξ' appears in f; we decompose $\mathsf{f} = \mathsf{g} : \xi : c : \xi' : d$ where c and d are sequences of registers. Since $\phi^k_s(g)$ is a prefix of $\mathsf{proj}_{[s-\mathsf{v},s]}(\mathsf{f}) = \phi^k_s(f)$, $\xi : c$ contains no registers of rounds in $[s-\mathsf{v}, s+\mathsf{v}]$; in particular $r_\xi \notin [s-\mathsf{v}, s+\mathsf{v}]$ and $r_\xi \geq k-\mathsf{v}$ hence $r_\xi > s+\mathsf{v}$ and, because f is swap-proof, c only has registers of rounds greater than $s+\mathsf{v}$. But then, the two last elements of $c : \xi'$ allow for a swap, which is a contradiction.

Therefore, for all $r \leq k$, $\mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{g}) = \phi^k_r(g)$. It suffices to define $\rho =: \rho_p \cdot \rho_s$ as the prefix execution of ρ such that the first move in ρ_s is the first write to the first register in f not in g. ◀

In order to prove the first statement of Theorem 21, we characterise the sets $S_k(f)$ for all k and f under the assumption that the computation does not reject.

▶ **Lemma B.7.** *Let $\mathcal{F} = (F_k)_{k\in\mathbb{N}}$ a family of projections, $k \in \mathbb{N}$ a f a prefix of F_k. If the partial computation of Algorithm 2 up until iteration (k, f) does not reject, then $S_k(f) = \mathcal{R}_k(f)$.*

Proof of Lemma B.7. We first prove $S_k(f) \subseteq \mathcal{R}_k(f)$, by induction on (k, f) with $k \in \mathbb{N}$ and f a prefix of F_k, using the lexicographical order: $(k, f) < (k', f')$ if $k < k'$ or $k = k'$ and f is a strict prefix of f'.

To do so, we build a family of abstract executions $\rho_k(f) : \sigma_{\mathsf{init}} \to \sigma_k(f)$ such that, for all k, f, for all $r \leq k$, $\mathsf{proj}_{[r-\mathsf{v},r]}(\mathsf{fwo}(\rho_k(f))) = \phi^k_r(f)$ and, for all $q \in S_k(f)$, $(q, k) \in \mathsf{loc}(\sigma_k(f))$. More precisely, the property proven by induction is that, if the partial $\mathcal{F}$-computation up until (k, f) is non-rejecting, then there exists an abstract execution $\rho_k(f) : \sigma_{\mathsf{init}} \to \sigma_k(f)$ such that:

- for all $r \leq k$, $\mathsf{proj}_{[r-\mathsf{v},r]}(\mathsf{fwo}(\rho_k(f))) = \phi^k_r(f)$,
- $S_k(f) \subseteq \mathsf{st}_k(\sigma_k(f))$,
- for all $r \leq k$, $\mathsf{proj}_{[0,r]}(\rho_k(f)) = \rho_r(\phi^k_r(f))$,
- for all prefixes g of f, $\rho_k(g)$ is a prefix of $\rho_k(f)$.

For simplicity, we initialize our induction with $k = -1$, in which case we have $F_{-1} = \varepsilon$ and $S_{-1}(\varepsilon) = \emptyset$; simply let $\rho_{-1}(\varepsilon)$ the empty execution.

Let (k, f) with $k \geq 0$ and f a prefix of F_k such that the partial $\mathcal{F}$-computation up until (k, f) is non-rejecting, and suppose that the property is true for all $(k', f') < (k, f)$. In the following, for all prefix h of F_k and $k' \leq k$, write $\tilde{\rho}_{k'}(h) := \rho_{k'}(\phi^k_{k'}(h))$. $\tilde{\rho}_{k'}(h)$ corresponds to the execution inductively build for round k and progression $\phi^{k'}_k(h)$, which is the progression on round k' that corresponds to progression h on k.

We build $\rho_k(f)$ step by step following the steps of iteration (k, f) of Algorithm 2. First, if $f \neq \varepsilon$, write $f = g{:}\xi$ with x a register. Let $\rho^{(1)} = \rho_k(g)$. By hypothesis, $\mathsf{proj}_{[0,k-1]}(\rho^{(1)}) = \tilde{\rho}_{k-1}(h)$, which is a prefix of $\tilde{\rho}_{k-1}(f)$ because $\phi^k_{k-1}(g)$ is a prefix of $\phi^k_{k-1}(f)$. Let ρ_{suf} be the corresponding suffix execution of $\tilde{\rho}_{k-1}(f)$, *i.e.*, $\tilde{\rho}_{k-1}(f) = \tilde{\rho}_{k-1}(g) \cdot \rho_{\mathsf{suf}}$. $\mathsf{sched}(\rho_{\mathsf{suf}})$ is applicable from $\sigma^{(1)}$ because ρ_{suf} only has moves on rounds 0 to $k-1$, is applicable after $\tilde{\rho}_{k-1}(h)$ and the projection of $\rho^{(1)}$ on rounds 0 to $k-1$ is $\tilde{\rho}_{k-1}(h)$. Let $\rho^{(2)} : \sigma_{\mathsf{init}} \xrightarrow{\mathsf{sched}(\rho_{\mathsf{suf}})} \sigma^{(2)}$. By induction hypothesis on g, $S_k(g) \subseteq \mathsf{st}_k(\sigma^{(2)})$; also, $\mathsf{proj}_{[0,k-1]}(\rho^{(2)}) = \tilde{\rho}_{k-1}(f)$.

If $f = \varepsilon$, let $\rho^{(2)} := \tilde{\rho}_{k-1}(f)$, which also gives $\mathsf{proj}_{[0,k-1]}(\rho^{(2)}) = \tilde{\rho}_{k-1}(f)$. Either way, $\mathsf{st}_k(\sigma^{(2)})$ contains all states that have been added to $S_k(f)$ at the end of Line **7**.

Let $\rho^{(3)} : \sigma_{\mathsf{init}} \xrightarrow{*} \sigma^{(3)}$ be the execution of schedule obtained by appending to $\mathsf{sched}(\rho^{(2)})$ all moves of the form $((q, \mathsf{Inc}, q'), k-1)$ with $q \in S_{k-1}(\phi^k_{k-1}(f))$. This is possible because $S_{k-1}(\phi^k_{k-1}(f)) \subseteq \mathsf{st}_{k-1}(\sigma^{(2)})$, by induction hypothesis applied on $(k-1, \phi^k_{k-1}(f))$ and thanks to $\mathsf{proj}_{[0,k-1]}(\rho^{(2)}) = \tilde{\rho}_{k-1}(f)$. We obtain that $\mathsf{st}_k(\sigma^{(3)})$ contains all states that are in $S_k(f)$ after Line **8**.

Write $\theta_1, \dots, \theta_\ell$ the moves detected by Line **10**, in this order. We prove the following property by induction on $i \in [0, \ell]$: there exists σ_i such that $\sigma^{(3)} \xrightarrow{*} \sigma_i$, all registers of rounds $k-\mathsf{v}$ to k in $\mathsf{FW}(\sigma_i)$ are in f and after the step of Line **10** detecting θ_i, $S_k(f) \subseteq \mathsf{st}_k(\sigma_i)$. The proof is by induction on i, the case $i = 0$ being a consequence of $S_k(f) \subseteq \mathsf{st}_k(\sigma^{(3)})$ after Line **8**. Suppose that the property is true until $i-1$. Write $\theta_i = ((q, a, q'), k)$. Since the algorithm detected θ_i, $q \in S_k(f)$ right before step i of Line **10**, and by induction hypothesis $(q, k) \in \mathsf{loc}(\sigma_{i-1})$. Moreover:

- if $a = \mathsf{write}_\alpha(x)$, then let σ_i such that $\sigma_{i-1} \xrightarrow{\theta_i} \sigma_i$; $\mathsf{rg}_\alpha[k]$ is in f hence all registers in $\mathsf{FW}(\sigma_i)$ of rounds $k-\mathsf{v}$ to k are in f;

- if $a = \mathsf{read}^{-j}_\alpha(d_0)$, then $\mathsf{rg}_\alpha[k-j]$ is not in f hence it is not in $\mathsf{FW}(\sigma_{i-1})$ and θ_i is applicable from σ_{i-1}, it then suffices to let σ_i such that $\sigma_{i-1} \xrightarrow{\theta_i} \sigma_i$;
- if $a = \mathsf{read}^0_\alpha(x)$ with $x \neq d_0$, there exist $q_1, q_2 \in \mathsf{st}_k(\sigma_{i-1})$ such that $(q_1, \mathsf{write}_\alpha(x), q_2) \in \Delta$ and $\mathsf{rg}_\alpha[k]$ in f; hence, q_1, q_2 are in σ_{i-1} and, by letting $\theta = ((q_1, \mathsf{write}_\alpha(x), q_2), k)$, $\theta \cdot \theta_i$ is applicable from σ_{i-1}, and it suffices to let σ_i such that $\sigma_{i-1} \xrightarrow{\theta \cdot \theta_i} \sigma_i$ (θ is here to make sure that $\mathsf{rg}_\alpha[k]$ is not blank);
- if $a = \mathsf{read}^{-j}_\alpha(x)$ with $x \neq d_0$ and $j > 0$, there exist $q_1, q_2 \in S_{k-j}(\phi^k_{k-j}(f))$ such that $(q_1, \mathsf{write}_\alpha(x), q_2) \in \Delta$ and $\mathsf{rg}_\alpha[k-j]$ in f; but $\mathsf{proj}_{[0,k-j]}(\rho^{(2)}) = \mathsf{proj}_{[0,k-j]}(\tilde{\rho}_{k-1}(f))$ since $j > 0$, and by induction hypothesis on $(k-1, \phi^k_{k-1}(f))$, $\mathsf{proj}_{[0,k-j]}(\tilde{\rho}_{k-1}(f)) = \rho_{k-j}(\phi^{k-1}_{k-j}(\phi^k_{k-1}(f))) = \rho_{k-j}(\phi^k_{k-j}(f))$, hence by induction hypothesis on $(k-j, \phi^k_{k-j}(f))$, $(q_1, k-j), (q_2, k-j) \in \mathsf{loc}(\sigma^{(2)}) \subseteq \mathsf{loc}(\sigma^{(3)})$, therefore θ_i is applicable from σ_{i-1} and one can let σ_i such that $\sigma_{i-1} \xrightarrow{\theta_i} \sigma_i$.

Therefore, there exists $\rho^{(4)} : \sigma^{(3)} \xrightarrow{*} \sigma^{(4)}$ where $\sigma^{(4)} = \sigma_\ell$ satisfies $\mathsf{st}_k(\sigma^{(4)}) = S_k(f)$ at the end of iteration (k, f) of Algorithm 2. By construction, $\rho^{(4)}$ only has moves on round k. Define $\rho_k(f)$ as the concatenation of $\rho^{(3)}$ and $\rho^{(4)}$. Note that $\mathsf{proj}_{[0,k-1]}(\rho_k(f)) = \mathsf{proj}_{[0,k-1]}(\rho^{(3)}) = \rho_{k-1}(\phi^k_{k-1}(f))$. We now check that $\rho_k(f)$ satisfies the required properties:

- by induction, for all $r < k$, $\mathsf{proj}_{[r-\mathsf{v},r]}(\mathsf{fwo}(\rho_r(k))) = \mathsf{proj}_{[r-\mathsf{v},r]}(\mathsf{fwo}(\rho_{k-1}(\phi^k_{k-1}(f)))) = \phi^{k-1}_r(\phi^k_{k-1}(f)) = \phi^k_r(f)$;
- since $\sigma_k(f) = \sigma_\ell$, $S_k(f) \subseteq \mathsf{st}_k(\sigma_k(f))$;
- by construction, for all prefixes g of f, $\rho_k(g)$ is a prefix of $\rho k f$,
- for all $r < k$, $\mathsf{proj}_{[0,r]}(\rho_k(f)) = \mathsf{proj}_{[0,r]}(\mathsf{proj}_{[0,k-1]}(\rho_k(\phi^k_)(f))) = \rho_r(\phi^k_r(f))$ by induction on $(k-1, \phi^k_{k-1}(f))$; also, $\mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{fwo}(\rho_k(f))) = f = \phi^k_k(f)$, indeed:
  - if $f = \varepsilon$ then the only first writes of $\rho_k(f)$ are in $\rho_{k-1}(\phi^k_{k-1}(\varepsilon))$ and by induction hypothesis $\mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{fwo}(\rho_k(f))) = \mathsf{proj}_{[k-\mathsf{v},k]}(\phi^k_{k-1}(\varepsilon)) = \varepsilon$;
  - if $f = g{:}\xi$ with $\mathsf{round}(\xi) < k$, the first writes of $\rho_k(f)$ are those of $\mathsf{fwo}(\rho_k(g))$ followed by those in $\mathsf{fwo}(\rho_{k-1}(\phi^k_{k-1}(f)))$ not in $\mathsf{fwo}(\rho_k(g))$ ($\rho^{(4)}$ adds no new first write); by induction on $k-1$ and by definition of $\phi^k_{k-1}(f)$, $\mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{fwo}(\rho_{k-1}(\phi^k_{k-1}(f)))) = \mathsf{proj}_{[k-\mathsf{v},k]}(\phi^k_{k-1}(f)) = \mathsf{proj}_{[k-\mathsf{v},k-1]}(f) = \mathsf{proj}_{[k-\mathsf{v},k-1]}(g) : \xi$. Hence, we get that $\mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{fwo}(\rho_k(f))) = g{:}\xi = f$;
  - if $f = g : \xi$ with $\mathsf{round}(\xi) = k$, then $\mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{fwo}(\rho_k(f)))$ is equal to g plus the first writes in $\rho^{(4)}$ not in g; $\rho^{(4)}$ only writes to registers in f, and since the partial ocmputation is non-rejecting, a first write is detected at Line **9** and $\rho^{(4)}$ writes on ξ, hence $\mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{fwo}(\rho_k(f))) = f$.

We now prove $\mathcal{R}_k(f) \subseteq S_k(f)$.

Suppose by contradiction that there exist $k \in \mathbb{N}$ and f a prefix of F_k such that the partial computation up until (k, f) is non-rejecting and $\mathcal{R}_k(f) \not\subseteq S_k(f)$. Let k, f minimal (for the lexicographical order) satisfying the previous statement. There exists an abstract execution $\rho : \sigma_{\mathsf{init}} \to \sigma$ such that $\mathsf{st}_k(\sigma) \not\subseteq S_k(f)$ and, for all $r \leq k$, $\mathsf{proj}_{[r-\mathsf{v},r]}(\mathsf{fwo}(\rho)) = \phi^k_r(f)$. By minimality of k, for all $r < k$, $\mathsf{st}_r(\sigma) \subseteq S_r(\phi^k_r(f))$: it suffices to consider execution $\mathsf{proj}_{[0,r]}(\rho)$. Also, for all g strict prefixes of f, thanks to Lemma B.6, there exists $\rho_p : \sigma_{\mathsf{init}} \xrightarrow{*} \sigma_p$ a prefix execution of ρ such that, for all $r \leq k$, $\mathsf{proj}_{[r-\mathsf{v},r]}(\mathsf{fwo}(\rho)) = \phi^k_r(g)$, hence, by minimality of f, $S_k(g) \subseteq \mathsf{st}_k(\sigma)$.

Consider q the first state covered by ρ on round k that is not in $S_k(f)$, *i.e.*, write $\rho : \sigma_{\mathsf{init}} \xrightarrow{\mathsf{s}_p} \sigma_p \xrightarrow{\theta} \sigma_m \xrightarrow{\mathsf{s}_s} \sigma_s$ with $\mathsf{st}_k(\sigma_p) \subseteq S_k(f)$ and $q \in \mathsf{st}_k(\sigma_m) \setminus S_k(f)$. We distinguish cases according to θ:

- if $\theta = ((q', \mathsf{Inc}, q), k-1)$, then $q' \in \mathsf{st}_{k-1}(\sigma) \subseteq S_{k-1}(\phi^k_{k-1}(f))$, hence $q \in S_k(f)$ thanks to Line **8**, which is a contradiction;
- if $\theta = ((q', \mathsf{write}_\alpha(x), q), k)$, then $q' \in S_k(f)$, and since $\mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{fwo}(\rho)) = f$, $\mathsf{rg}_\alpha[k]$ is in f, hence q is added to $S_k(f)$ at Line **10**, which is a contradiction;
- if $\theta = ((q', \mathsf{read}^{-j}_\alpha(d_0), q), k)$, then $q' \in S_k(f)$ and by writing $\rho_p : \sigma_{\mathsf{init}} \xrightarrow{*} \sigma_p$ and $h := \mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{fwo}(\rho_p))$, we have that $\mathsf{rg}_\alpha[k-j]$ is not in h since θ is applicable from σ_p, hence q is added at Line **10** to $S_k(h) \subseteq S_k(f)$, which is a contradiction;
- if $\theta = ((q', \mathsf{read}^{-j}_\alpha(x), q), k)$ with $x \neq d_0$, then $q' \in S_k(f)$, and there exist q_1, q_2 such that $(q_1, k-j), (q_2, k-j) \in \mathsf{loc}(\sigma_p)$ and $(q_1, \mathsf{write}_\alpha(x), q_2) \in \Delta$; by minimality of k, $q_1, q_2 \in S_{k-j}(\phi^k_{k-j}(f))$, and since $\mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{fwo}(\rho)) = f$, $\mathsf{rg}_\alpha[k-j]$ is in f; hence q is added to $S_k(f)$ at Line **10**, which is a contradiction.

◀

The second statement of Theorem 21 is a consequence of the following lemma:

▶ **Lemma B.8.** *Let $\mathcal{F} = (F_k)_{k\in\mathbb{N}}$ a family of first-write order projections, $k \in \mathbb{N}$, f a prefix of F_k. Suppose that there exists an execution ρ from σ_{init} such that, for all $r \leq k$, $\mathsf{proj}_{[r-\mathsf{v},r]}(\mathsf{fwo}(\rho)) = \phi^k_r(F_k)$. Then the partial $\mathcal{F}$-computation of Algorithm 2 up until iteration (k, f) is non-rejecting.*

Proof of Lemma B.8. We proceed by induction on (k, f). Again, for simplicity, we initialize the induction with $k = -1$ and $f = \varepsilon$, in which case the partial computation does nothing hence is non-rejecting. Let $k \in \mathbb{N}$, f a prefix of F_k and suppose that the property is true for all $(k', f') < (k, f)$. Suppose that there exist an abstract execution ρ starting on σ_{init} such that, for all $r \leq k$, $\mathsf{proj}_{[r-\mathsf{v},r]}(\mathsf{fwo}(\rho)) = \phi^k_r(F_k)$.

First, consider the case $f = \varepsilon$. Apply the induction hypothesis on $(k-1, F_{k-1})$ with witness ρ, the partial $\mathcal{F}$-computation up until $(k-1, F_{k-1})$ is non-rejecting. Because there in no first write to check in ε, iteration (k, ε) does not reject at Line **9** and the partial $\mathcal{F}$-computation up until (k, ε) is non-rejecting.

Now, treat the case $f = g{:}\xi$. By induction hypothesis on g, the partial $\mathcal{F}$-computation up until (k, g) is non-rejecting. Thanks to Lemma B.6, since g is a prefix of F_k, there exist ρ_p, ρ_s such that $\rho = \rho_p \cdot \rho_s$, for all $r \leq k$, $\mathsf{proj}_{[r-\mathsf{v},r]}(\mathsf{fwo}(\rho_p)) = \phi^k_r(g)$, and ρ_s starts with a first write on ξ.

If ξ is on a round $< k$, then iteration (k, f) has no first write to check at Line **9**, and the partial $\mathcal{F}$-computation up until (k, f) is non-rejecting. If ξ is on round k, write $\rho_p : \sigma_{\mathsf{init}} \xrightarrow{*} \sigma_p$, and let θ the first move in ρ_s, which is a first write on ξ. By applying Lemma B.7, since ρ_p satisfies the condition in $\mathcal{R}_k(g)$, all the states in $\mathsf{st}_k(\sigma_p)$ are in $S_k(g)$. Since θ is applicable from σ_p, it is detected by the algorithm at Line **9** during iteration (k, f). Therefore, the partial $\mathcal{F}$-computation up until (k, f) is non-rejecting. ◀

To conclude the proof of Theorem 21, letting an abstract execution ρ from σ_{init}, it suffices to apply Lemma B.8 to $\mathcal{F} = (\mathsf{proj}_{[j-\mathsf{v},j]}(\mathsf{fwo}(\rho)))_{k\in\mathbb{N}}$ and to all (k, f). This proves that all partials $\mathcal{F}$-computations are non-rejecting, hence that the $\mathcal{F}$-computation is non-rejecting. ◀

B.5 Proof of PSPACE-hardness

▶ **Lemma 24.** *Let $k \in \mathbb{N}$. Suppose that (q_ψ, k) is coverable and that we have a valuation ν of the variables of ψ such that, for every $i \in [0, 2m-1]$:*
- *if $\nu(x_i) = 1$, then $\mathtt{x_i}$ can be written to $\mathsf{rg}[k]$, and $\neg\mathtt{x_i}$ cannot,*
- *if $\nu(x_i) = 0$, then $\neg\mathtt{x_i}$ can be written to $\mathsf{rg}[k]$, and $\mathtt{x_i}$ cannot.*

Then (q_{yes}, k) is coverable if and only if $\nu \models \psi$, and (q_{no}, k) is coverable if and only if $\nu \models \neg\psi$.

Proof of Lemma 24. If $\nu \models \psi$ then for all $i \in [1,p]$, ν must set to true one of the literals a_i, b_i and c_i. By hypothesis, for all $i \in [1,p]$, one symbol among $\mathtt{a_i}$, $\mathtt{b_i}$ and $\mathtt{c_i}$ can be written to $\mathsf{rg}[k]$, and (q_ψ, k) is coverable hence (q_{yes}, k) is coverable too. Moreover, for all $i \in [1,p]$, one symbol among $\neg\mathtt{a_1}$, $\neg\mathtt{b_i}$ and $\neg\mathtt{c_i}$ cannot be written to $\mathsf{rg}[k]$ hence (q_{no}, k) is not coverable.

If $\nu \models \neg\psi$, there exists $i \in [1,p]$ such that ν sets to false all three literals a_i, b_i and c_i. We consider the minimal i with this property. By hypothesis, none of the symbols among $\mathtt{a_i}$, $\mathtt{b_i}$ and $\mathtt{c_i}$ can be written to $\mathsf{rg}[k]$, and (q_{yes}, k) is not coverable. Moreover, by minimality of i, (q_{i-1}, k) is coverable and one symbol among $\neg\mathtt{a_1}$, $\neg\mathtt{b_i}$ and $\neg\mathtt{c_i}$ can be written to $\mathsf{rg}[k]$, hence (q_{no}, k) is coverable. ◀

▶ **Lemma 27.** *Let $k \in \mathbb{N}$ and $\nu_k := \mathsf{next}^k(\nu_0)$, the valuation obtained by applying* next *k times from $\nu_0 := 0^{2m}$. For all $i \in [0, 2m{-}1]$:*

- *$(q_{\mathsf{false},i}, k)$ is coverable if and only if $\nu_k(x_i) = 0$,*
- *$(q_{\mathsf{true},i}, k)$ is coverable if and only if $\nu_k(x_i) = 1$,*
- *$\neg\mathtt{x_i}$ can be written to $\mathsf{rg}[k]$ if and only if $\nu_k(x_i) = 0$,*
- *$\mathtt{x_i}$ can be written to $\mathsf{rg}[k]$ if and only if $\nu_k(x_i) = 1$.*

Moreover, if $k > 0$, then for all $j \in [0, 2m]$:

- *yes_j can be written to $\mathsf{rg}[k]$ if and only if computation $\nu_k = \mathsf{next}(\nu_{k-1})$ sets b_j to* yes*,*
- *no_j can be written to $\mathsf{rg}[k]$ if and only if computation $\nu_k = \mathsf{next}(\nu_{k-1})$ sets b_j to* no*,*
- *wait_j can be written to $\mathsf{rg}[k]$ if and only if computation $\nu_k = \mathsf{next}(\nu_{k-1})$ sets b_j to* wait*.*

Proof of Lemma 27. Write $P_{k,i}$ for the property corresponding to the first four items, and $Q_{k,j}$ for the property corresponding to the last three items in the lemma statement. We prove by induction on k the following property: for all $i \in [0, 2m{-}1]$, $P_{k,i}$, and if $k > 0$, for all $j \in [0, 2m]$, $Q_{k,j}$.

First, for all $i \in [0, 2m{-}1]$, $(q_{\mathsf{false},i}, 0)$ is coverable and $(q_{\mathsf{true},i}, 0)$ is not; also, $\neg\mathtt{x_i}$ can be written to $\mathsf{rg}[0]$ and $\mathtt{x_i}$ cannot, which proves the case $k = 0$.

Suppose that $k > 0$ and that the property is true for $k{-}1$. Write $(b_j)_{j \in [0,2m]}$ for the values set by computation $\nu_k = \mathsf{next}(\nu_{k-1})$.
We prove $Q_{k,j}$, $j \in [0, 2m]$, by induction on j. Thanks to Lemma 24 and to the induction hypothesis on $k{-}1$, yes_0 can be written to $\mathsf{rg}[k]$ if and only if $\nu_{k-1} \models \psi$, *i.e.*, if and only if $b_0 = \mathsf{yes}$; a similar property holds for no_0. Also, wait_0 cannot be written to $\mathsf{rg}[k]$, and $b_0 \neq \mathsf{wait}$, which proves $Q_{k,0}$.

Suppose that the property is true for $j \in [0, 2m{-}1]$ in order to prove it for $j{+}1$. By induction hypothesis on k, we have that $(q_{\mathsf{true},i}, k{-}1)$ is coverable if and only if $\nu_{k-1}(x_i) = 1$ (and similarly for $q_{\mathsf{false},i}$). Moreover, by the induction hypothesis applied to $j{-}1$, exactly one symbol among $\{\mathsf{yes}_{j-1}, \mathsf{no}_{j-1}, \mathsf{wait}_{j-1}\}$ can be written to $\mathsf{rg}[k]$ and it matches b_{j-1}. Therefore, by looking at every case in the computation of $\mathsf{next}(\nu)(x_{j-1})$, exactly one symbol among $\{\mathsf{yes}_j, \mathsf{no}_j, \mathsf{wait}_j\}$ can be written to $\mathsf{rg}[k]$ and it matches b_j. This also proves that exactly one of $\{(q_{\mathsf{true},j-1}, k), (q_{\mathsf{false},j-1}, k)\}$ is coverable and that it matches $\nu_k(x_{j-1})$. ◀