On flat lossy channel machines

Philippe Schnoebelen

On flat lossy channel machines

Ph. Schnoebelen

Abstract

We show that reachability, repeated reachability, nontermination and unboundedness are NP-complete for Lossy Channel Machines that are *flat*, i.e., with no nested cycles in the control graph. The upper complexity bound relies on a fine analysis of iterations of lossy channel actions and uses compressed word techniques for efficiently reasoning with paths of exponential lengths. The lower bounds already apply to acyclic or single-path machines.

1 Introduction

Lossy channel machines, aka LCMs, are FIFO automata, i.e., finite-state machines operating on buffers with FIFO read/write discipline, where the buffers are *unreliable*, or *lossy*, in the sense that letters (or "messages") in a buffer can be lost nondeterministically at any time.

LCMs were first introduced as a model for communication protocols designed to work properly in unreliable environments. They immediately attracted interest because, unlike FIFO automata with reliable buffers, they have decidable safety and termination problems [Fin94, AJ96, CFP96, ACBJ04]. It was later found that LCMs are a relevant computational model *per se*, useful for verifying timed automata [ADOW05, LW08], modal logics [GKWZ06], etc., and connected to other problems in computer science [KS15, CS10, Sch16].

Flat LCMs. In this paper we consider the case of *flat LCMs*, i.e., LCMs where the control graph has no nested cycles. In the area of infinite-state systems verification, flat systems were first considered in [FO97, CJ98] for counter systems[1]. In addition, some earlier "loop acceleration" results, e.g. [BW94], where one can compute reachability sets along a cycle, can often be generalised to flat systems. Positive results on flat counter systems can be found in [LS05, BIL09, DFGvD10, BIK14, LPS14, DDS15], and in [GI15] for counter systems with recursive calls. Regarding flat FIFO automata, verification was shown decidable by Bouajjani and Habermehl [BH99] who improved on earlier results by Boigelot [BG99], and the main verification problems were only recently proven to be NP-complete [EGM12, FP19]. These results have applications beyond flat systems in the context of *bounded verification* techniques, where one analyses a bounded subset of the runs of a general system [EGM12].

[1] Flatness remains relevant with *finite-state systems*, see e.g., [KF11]. This is especially true when one is considering the verification of properties expressed in a rich logic as in, e.g., [DHL+17]. In language theory, flat finite-state automata correspond to regular languages of polynomial density, sometimes called *sparse languages*, or also *bounded languages*.

Flat LCMs have not been explicitly considered in the literature. They are implicit in forward analysis methods based on loop acceleration, starting with [ACBJ04], but these works do not address the overall complexity of the verification problem, only the complexity of elementary operations.

It is not clear whether one should expect flat LCMs to be simpler than flat FIFO automata (on account of unrestricted LCMs being simpler than the Turing powerful, unrestricted FIFO automata), or if they could be more complex since message losses introduce some nondeterminism that does not occur when one follows a fixed cycle in a FIFO system. Indeed, message losses can be seen as hidden implicit loops that disrupt the apparent flatness of the LCM.

Our contribution. We analyse the behaviour of the backward-reachability algorithm on cycles of lossy channel actions and establish a bilinear upper bound on its complexity. As a consequence, reachability along runs of the form $\rho_1\sigma_1^*\rho_2\sigma_2^*\dots\rho_m\sigma_m^*$ where the ρ_i, σ_i are sequences of channel actions, can be decided in time $n^{O(m)}$. While shortest reachability witnesses can be exponentially long when the number m of cycles is not bounded, techniques based on SLP-compressed words allow handling and checking these witnesses in polynomial time, leading to an NP algorithm for flat LCMs. This easily translates into NP algorithms for nontermination, repeated reachability, and unboundedness, and in fact all four problems are NP-complete. Thus the restriction to flat systems really brings some simplification when compared to the very high complexity —sometimes undecidability as is the case for unboundedness— of verification for unrestricted LCMs [CS08, SS11, Sch16].

Remark 1.1 (Lossy channel machines vs. lossy channel systems). In line with most works on loop acceleration and verification of flat systems, we consider lossy channel "machines" instead of the more usual lossy channel "systems", i.e., systems where several independent concurrent machines communicate via shared channels. This is because a combination of individually flat machines does not lead to a "flat" system. Additionally, finite-state concurrent systems typically have PSPACE-hard verification problems already when they have no channels and run synchronously, or when they only synchronise via bounded channels that can hold at most one message [DLS06]. □

Outline. After some technical preliminaries (Section 2), we present our main technical contribution (Section 3): we analyse the computation of predecessors (of some given configuration) through a cycle iterated arbitrarily many times. In particular we show that the backward-reachability analysis of a single cycle reaches its fixpoint after a bilinear number of iterations. This leads to an effective bound on the length of the shortest runs between two configurations. In Section 4 we show how the previous analysis can be turned into a nondeterministic polynomial-time algorithmic via the use of SLP-compressed words for efficiently computing intermediary channel contents along a run. In Section 5 we show how our main results also apply to termination, repeated reachability, and boundedness. Finally Appendix C presents reductions showing how the problems we considered are NP-hard, even for acyclic LCMs or single-path LCMs.

Related work. After we circulated our draft proof, we became aware that a related NP-membership result will be found in [FP20]. There the authors adapt the powerful technique from [EGM12] and encode front-lossy channel systems into multi-head pushdown automata, from which an NP-algorithm for control-state reachability in flat machines ensue. Our approach is lower level, providing a tight bilinear bound on the number of times a cycle

must be visited in the backward-reachability algorithm. Once these bounds are established, our NP algorithm only needs to guess the number of times each cycle is visited.

2 Preliminaries

We consider words $u, v, w, x, y, z, \ldots$ over a finite alphabet $\Sigma = \{a, b, \ldots\}$. We write $|u|$ for the length of a word and ε for the empty word. The set of letters that occur in u is written $\mathrm{alph}(u)$. For a n-letter word $u = a_1 \cdots a_n$ and some index $\ell \in \{0, 1, \ldots, n\}$, we write $u_{\leq \ell} \stackrel{\text{def}}{=} a_1 \cdots a_\ell$ and $u_{>\ell} \stackrel{\text{def}}{=} a_{\ell+1} \cdots a_n$ for the ℓ-th prefix and the ℓ-th suffix of u. We write $u_{(\ell)} \stackrel{\text{def}}{=} u_{>\ell} \cdot u_{\leq \ell}$ for the ℓ-th *cyclic shift* of u.

Exponents are used to denote the concatenation of multiple copies of a same word, i.e., u^3 denotes uuu. A *fractional exponent* $p \in \mathbb{Q}$ can be used for u^p if $p \cdot |u|$ is a natural number. E.g., when a, b, c are letters, $(abc)^{\frac{11}{3}}$, or equivalently $(abc)^{3+\frac{2}{3}}$, denotes $abc\,abc\,abc\,ab$.

We write $u \preccurlyeq v$ to denote that u is a (scattered) subword of v, i.e., there exist $2m+1$ words $u_1, \ldots, u_m, x_0, x_1, \ldots, x_m$ such that $u = u_1 \cdots u_m$ and $v = x_0 u_1 x_1 \ldots u_m x_m$. It is well-known that $\preccurlyeq$ is a well-founded partial ordering. For a word $x \in \Sigma^*$, we write $\uparrow x \stackrel{\text{def}}{=} \{y \in \Sigma^* \mid x \preccurlyeq y\}$ to denote the *upward-closure* of x, i.e., the set of all words that contain x as a (scattered) subword.

LCMs. In this paper we consider channel machines with a single communication channel[2]. A *lossy channel machine* (LCM) is a tuple $S = \langle Q, \Sigma, \Delta \rangle$ where $Q = \{p, q, \ldots\}$ is a finite set of *control locations*, or just "locations", $\Sigma = \{a, b, \ldots\}$ is the finite *message alphabet*, and $\Delta \subseteq Q \times (\{!, ?\} \times \Sigma^*) \times Q$ is a finite set of *transition rules*. A rule $\delta = \langle q, (d, w), q' \rangle$ has a start location q, an end location q' and a *channel action* (d, w). We write $Act_\Sigma \stackrel{\text{def}}{=} \{!, ?\} \times \Sigma^*$ for the set of channel actions over Σ, and often omit the Σ subscript when it can be inferred from the context. We use $\theta, \theta', \ldots$ to denote actions and $\sigma, \rho, \ldots$ to denote sequences of channel actions.

We'll constantly refer to the *written part* and the *read part* of some channel action (or sequence of such). These are formally defined via

$$\begin{array}{lll} \mathtt{wri}(!w) \stackrel{\text{def}}{=} w\,, & \mathtt{wri}(?w) \stackrel{\text{def}}{=} \varepsilon\,, & \mathtt{wri}(\theta_1 \cdots \theta_m) \stackrel{\text{def}}{=} \mathtt{wri}(\theta_1) \cdots \mathtt{wri}(\theta_m)\,, \\ \mathtt{rea}(!w) \stackrel{\text{def}}{=} \varepsilon\,, & \mathtt{rea}(?w) \stackrel{\text{def}}{=} w\,, & \mathtt{rea}(\theta_1 \cdots \theta_m) \stackrel{\text{def}}{=} \mathtt{rea}(\theta_1) \cdots \mathtt{rea}(\theta_m)\,. \end{array} \tag{1}$$

Semantics. The operational semantics of LCMs is given via transition systems. Fix some LCM $S = \langle Q, \Sigma, \Delta \rangle$. Actions in Act_Σ induce a ternary relation $\rightarrow \subseteq \Sigma^* \times Act_\Sigma \times \Sigma^*$ on channel contents:

$$x \xrightarrow{!\,w} y \iff y \preccurlyeq x\,w\,, \qquad\qquad x \xrightarrow{?\,w} y \iff w\,y \preccurlyeq x\,. \tag{2}$$

Observe how Equation (2) includes the subword relation in the definition of the operational semantics. This models the fact that messages in the channel can be lost nondeterministically during any single computation step. A consequence is the following monotonicity property: if $x' \succcurlyeq x$ and $y \succcurlyeq y'$ then $x \xrightarrow{\theta} y$ implies $x' \xrightarrow{\theta} y'$.

[2]See Appendix D for a generalisation of our results to multi-channel machines.

A *configuration* of S is a pair $c = (q, x) \in Q \times \Sigma^*$ that denotes a current situation where the control of S is set at q while the contents of the channel is x. We let $Conf_S \stackrel{\text{def}}{=} Q \times \Sigma^*$ denote the set of configurations. The set of rules Δ induces a labelled transition relation $\to \subseteq Conf_S \times \Delta \times Conf_s$ between configurations defined by

$$(q, x) \xrightarrow{\delta} (q', y) \iff \text{def} \iff \delta \in \Delta \text{ has the form } \langle q, \theta, q' \rangle \text{ and } x \xrightarrow{\theta} y\,. \tag{3}$$

Several convenient notations are derived from the main transition relation: we write $c \xrightarrow{\theta} c'$ when $c \xrightarrow{\delta} c'$ for a rule δ that carries action θ. When $\sigma = \theta_1\theta_2 \cdots \theta_m$ is a sequence of actions, we write $c \xrightarrow{\sigma} c'$ when there is a sequence of steps $c_0 \xrightarrow{\theta_1} c_1 \xrightarrow{\theta_2} c_2 \cdots \xrightarrow{\theta_m} c_m$ with $c_0 = c$ and $c_m = c'$. Then $c \xrightarrow{*} c'$ means that $c \xrightarrow{\sigma} c'$ for some sequence σ. Similar notations, e.g., "$x \xrightarrow{\theta_1\,\theta_2} y$" or "$x \xrightarrow{\theta^*} y$", are used for channel contents. In fact, since we shall mostly consider fixed paths, or paths of a fixed shape, we will usually concentrate on the channel contents and leave the visited locations implicit.

Flat LCMs. An elementary cycle of length m in a LCM S is a non-empty set $C = \{\langle p_i, \theta_i, q_i \rangle \mid i = 1, \ldots, m\}$ of rules from Δ such that $q_m = p_1$ and $p_i = q_{i-1}$ when $2 \leq i \leq m$, and such that the p_i's are all distinct. A cycle of length 1 is a *self-loop*. The set $\{p_1, \ldots, p_m\}$ is the set of locations *visited* by C. Note that two distinct cycles may have the same visited set if they use different transition rules.

We say that S is *flat* if no control location is visited by two different elementary cycles. An extreme case of flat machines are the machines having no cycles whatsoever, called *acyclic* machines.[3]

When S is flat, there is (at most) one cycle around any location q and we write σ_q for the sequence of actions along this cycle, making sure that σ_q starts with the action leaving q (so that if q, q' are two locations visited by the same cycle, $\sigma_{q'}$ will be a cyclic shift of σ_q). When there is no cycle visiting q we let $\sigma_q = \varepsilon$ by convention.

NP-*hardness.* It is known that reachability and other verification problems are NP-hard for (reliable) FIFO automata: see [EGM12, App. C] and [FP19]. We strengthen these results in Appendix C with the following theorems that cover reliable and unreliable channels indifferently.

Theorem 2.1 (Hardness for acyclic channel machines)**.** *Reachability, nontermination and unboundedness are* NP*-hard for* acyclic *channel machines, with reliable or with unreliable channels. Hardness already holds for a single channel and a binary alphabet. It also holds for a unary alphabet (i.e., for acyclic VASSes, reliable or lossy) provided one allows several channels (or counters).*

NP-hardness for acyclic machines uses the nondeterminism allowed in channel machines. It is thus interesting to consider *single-path* machines where the control graph is a single line possibly carrying cycles on some locations, as is done in [KF11] or [DDS15]. In such a machine, nondeterminism only occurs in choosing how many times a cycle is visited (and what messages are lost in unreliable systems). This is equivalent to considering reachability (or nontermination or unboundedness) *along a given bounded path scheme of the form* $q_1C_1^*q_2C_2^* \cdots q_mC_m^*$.

[3] In the finite-automata literature, "acyclic automata" sometimes allow self-loops.

Theorem 2.2 (Hardness for single-path channel machines)**.** *Reachability, nontermination and unboundedness are* NP*-hard for* single-path *channel machines, with reliable or with unreliable channels. Hardness already holds for a single channel. It also holds for single-path VASSes, reliable or lossy, provided one allows several counters.*

The above NP-hardness does not apply to bounded path schemes *with a fixed number of cycles* and indeed we show in Section 3 that reachability along path schemes with m cycles can be verified in polynomial-time $n^{O(m)}$.

3 Backward reachability in flat LCMs

In this section we consider a generic flat single-channel LCM S with channel alphabet Σ and investigate the complexity of backward-reachability analysis.

3.1 Computing predecessors

The classical approach to deciding reachability in LCMs is the backward-reachability algorithm proposed by Abdulla and Jonsson. They first developed it for lossy channel systems [AJ96] before generalising it to the larger class of Well-Structured Systems [AČJT00, FS01].

For backward reachability, we write $\mathtt{Pre}[\sigma](x)$ for $\{y \in \Sigma^* \mid y \xrightarrow{\sigma} x\}$, the set of σ-predecessors of x, and $\mathtt{Pre}[\sigma](\uparrow x)$ for $\{y \in \Sigma^* \mid \exists x' \in \uparrow x : y \xrightarrow{\sigma} x'\}$, the set of σ-predecessors of x "and larger contents". A consequence of the monotonicity of steps is that $\mathtt{Pre}[\sigma](\uparrow x)$ is upward-closed set and, unless σ is the empty sequence, coincides with $\mathtt{Pre}[\sigma](x)$.

Definition 3.1 ($\mathtt{pr}[\sigma](x)$)**.** For a channel contents $x \in \Sigma^*$ and a sequence σ of channel actions, we write $\mathtt{pr}[\sigma](x) = y$ when $\mathtt{Pre}[\sigma](\uparrow x) = \uparrow y$.

In the case of lossy channels, $\mathtt{Pre}[\sigma](\uparrow x)$ always has a single minimal element, hence $\mathtt{pr}[\sigma](x)$ is always defined. We now explain how to compute it.

For two words x, v we define x/v as the prefix of x that remains when we remove from x its longest suffix that is a subword of v. This operation is always defined and can be computed using the following rules where a, b are letters:

$$x/\varepsilon = x\,, \qquad \varepsilon/v = \varepsilon\,, \qquad (x\,a)/(v\,b) = \begin{cases} x/v & \text{if } a = b, \\ (x\,a)/v & \text{if } a \neq b. \end{cases} \tag{4}$$

This immediately entails $(x/v)/v' = x/(v'\,v)$. We'll also use the following properties:

$$\text{if } x'/v \neq \varepsilon \text{ then } x(x'/v) = (x\,x')/v\,, \qquad \text{if } |x'| > |v| \text{ then } x'/v \neq \varepsilon\,. \tag{5}$$

We may now compute $\mathtt{pr}[\sigma](x)$ with:

$$\begin{aligned} \mathtt{pr}[?u](x) &= u \cdot x\,, & \mathtt{pr}[!v](x) &= x/v\,, \\ \mathtt{pr}[\varepsilon](x) &= x\,, & \mathtt{pr}[\sigma_1 \cdot \sigma_2](x) &= \mathtt{pr}[\sigma_1]\big(\mathtt{pr}[\sigma_2](x)\big)\,. \end{aligned} \tag{6}$$

W.r.t. subword ordering, the / operation is monotonic in its first argument and contramonotonic in the second : $u \preccurlyeq u'$ implies $u/v \preccurlyeq u'/v$ and $x/u \succcurlyeq x/u'$. Concatenation too is monotonic. This generalises to the following useful lemma:

Lemma 3.2. *Assume* $\mathtt{pr}[\sigma](x) = y$ *and* $\mathtt{pr}[\sigma](x') = y'$ *where* σ *is some sequence of actions. Then* $x \preccurlyeq x'$ *implies* $y \preccurlyeq y'$.

Proof. By induction on σ, using eqs. (4) and (6). □

3.2 Cycles: repeating a given sequence of actions

We now focus on computing $\mathtt{pr}[\sigma^k](x)$ for σ a sequence of actions and some $k \in \mathbb{N}$.

Without any loss of generality, σ can be written in the general form $?a_1!b_1?a_2!b_2\cdots?a_r!b_r$ where each a_i and b_i is a letter or the empty word ε. Then $\mathtt{rea}(\sigma) = a_1a_2\cdots a_r$ and $\mathtt{wri}(\sigma) = b_1b_2\cdots b_r$.

To fix notation, we define "*the small-step sequence for* $\mathtt{pr}[\sigma](x)$", or just "*the SSS*", as the sequence y_r, y'_r, y_{r-1}, y'_{r-1}, …, y_1, y'_1, y_0 of $2r+1$ words given by

$$y_r = x\,, \qquad\qquad y'_i = y_i/b_i\,, \qquad\qquad y_{i-1} = a_i\, y'_i\,. \tag{7}$$

Clearly, the SSS lists all the intermediary steps in the computation of $\mathtt{pr}[\sigma](x)$ as dictated by eq. (6), and thus it yields $y_0 = \mathtt{pr}[\sigma](x)$.

Our first lemma handles the special case where x is made of copies of $\mathtt{rea}(\sigma)$.

Lemma 3.3. *Let* $u = \mathtt{rea}(\sigma)$.
(i) If x *is a fractional power* u^p *of* u, *then* $y = \mathtt{pr}[\sigma](x)$ *is also a fractional power of* u, *written* $y = u^m$.
(ii) Furthermore, if $m > 1$, *then* $\mathtt{pr}[\sigma](u^{p+n}) = u^{m+n}$ *for all* $n \in \mathbb{N}$.
(iii) Finally, for all $n \in \mathbb{N}$, *if* $m > n+1$, *then* $\mathtt{pr}[\sigma](u^{p-n}) = u^{m-n}$.

Proof. The lemma holds spuriously if $u = \varepsilon$, so we assume $|u| > 0$. Let us write σ in the general form $?a_1\,!b_1\,?a_2\,!b_2\cdots?a_r\,!b_r$, so that $u = a_1a_2\cdots a_r$. To simplify notation we will write $u_{(i)}$ for the shift $a_{i+1}\cdots a_r\cdot a_1\ldots a_i$ that really should be written $u_{(|a_1\cdots a_i|)}$ (remember that $a_j = \varepsilon$ is possible).

We now claim that, in the SSS $(y_i, y'_i)_i$ for σ and x, each y_i and y'_i is a fractional power of $u_{(i)}$, written $y_i = u_{(i)}^{p_i}$ and $y'_i = u_{(i)}^{p'_i}$.

The proof is by induction on $r - i$. For y_i, there are two cases: (1) $y_r = x$ is a power of u by assumption, hence of $u_{(r)}$, with $p_r = p$; (2) y_{i-1} is $a_i\, y'_i$, i.e., $a_i\, u_{(i)}^{p'_i}$ by ind. hyp., hence a power of $u_{(i-1)}$ with $p_{i-1} = p'_i + \frac{|a_i|}{|u|}$. For y'_i the proof is simpler: by ind. hyp. it is $u_{(i)}^{p_i}/b_i$ and, as a prefix of a power of $u_{(i)}$, is itself a power of $u_{(i)}$, albeit with a perhaps smaller exponent, i.e., $p_i - \frac{|b_i|}{|u|} \le p'_i \le p_i$.

(i) Since y coincide with y_0, we obtain $y = u^m$ as required by letting $m = p_0$.

(ii) Equation (8) gathers the (in)equalities we just established:

$$p_r = p\,, \qquad p_{i-1} = p'_i + \frac{|a_i|}{|u|}\,, \qquad \max\Big(0, p_i - \frac{|b_i|}{|u|}\Big) \le p'_i \le p_i\,, \qquad p_0 = m\,. \tag{8}$$

Thus the assumption $m > 1$ entails $p'_i > 0$, i.e. $y'_i \neq \varepsilon$, for all $i = r, r-1, \ldots, 2, 1$. Let us now consider the SSS $(z_i, z'_i)_i$ for $\mathtt{pr}[\sigma](u^{p+1})$. We claim that for all i, $z_i = u_{(i)}y_i$ and

$z'_i = u_{(i)}y'_i$, as is easily proven by induction on $r-i$. The crucial case is z'_i, defined as z_i/b_i and equal to $(u_{(i)}y_i)/b_i$ by ind. hyp. Since $y_i/b_i = y'_i \neq \varepsilon$ as just observed, we deduce $(u_{(i)}y_i)/b_i = u_{(i)}(y_i/b_i)$ from eq. (5). This is $u_{(i)}y'_i$ as required. Finally we end up with $\mathtt{pr}[\sigma](u^{p+1}) = z_0 = u_{(0)}y_0 = u\,u^m = u^{m+1}$, and this generalises to $\mathtt{pr}[\sigma](u^{p+n}) = u^{m+n}$.

(iii) With eq. (8), the assumption $m > n+1$ now entails $p_i, p'_i \geq n + \frac{1}{|u|}$ for all i. We claim that the SSS $(z_i, z'_i)_i$ for $\mathtt{pr}[\sigma](u^{p-n})$ satisfies $u^n_{(i)}z_i = y_i$ and $u^n_{(i)}z'_i = y'_i$ for all i, as can be proved by induction on $r-i$. The base case $u^n z_r = u^n u^{p-n} = u^p = y_r$ is clear. Let us now consider $u^n_{(i)}z'_i$. It is $u^n_{(i)}(z_i/b_i)$, that is $u^n_{(i)}(u^{p_i-n}_{(i)}/b_i)$ since $u^n_{(i)}z_i = y_i$ by ind. hyp. and $y_i = u^{p_i}_{(i)}$ by (i). Now $|u^{p_i-n}_{(i)}| = |u|(p_i - n) \geq 1 \geq |b_i|$, so eq. (5) applies and we deduce $u^n_{(i)}(z_i/b_i) = (u^n_{(i)}z_i)/b_i = y_i/b_i$ (by ind. hyp.) $= y'_i$. We have proved $u^n_{(i)}z'_i = y'_i$ as required. Finally, proving $u^n_{(i)}z_i = y_i$ is handled in a similar way.

□

Note that $m > 1$ is required for part (ii) of the Lemma. For example, with $\sigma = ?a!b?c!c!a$ one has $u = \mathtt{rea}(\sigma) = a\,c$ and $\mathtt{pr}[\sigma](u^{\frac{1}{2}}) = u^1$. However one can check that $\mathtt{pr}[\sigma](u^{\frac{3}{2}}) = a\,c\,a = u^{\frac{3}{2}}$.

Equipped with Lemma 3.3, we turn to the general case for $\mathtt{pr}[\sigma^k](x)$.

Theorem 3.4. *Let $\sigma \in Act^*_\Sigma$ be a sequence of actions and write u for $\mathtt{rea}(\sigma)$. Let $x \in \Sigma^*$ be some channel contents and write y_k for $\mathtt{pr}[\sigma^k](x)$.*
(i) For every $k \in \mathbb{N}$, y_k has the form $u^{p_k} \cdot x_{<\ell_k}$ for some fractional power p_k and some length $\ell_k \in \{0, 1, \ldots, |x|\}$.
(ii) Furthermore, computing p_k and ℓ_k can be done in time $\mathsf{poly}(|\sigma| + |x| + \log k)$.

Proof. (i) Write v for $\mathtt{wri}(\sigma)$ and consider the sequence $(x_k)_{k\in\mathbb{N}}$ given by $x_0 \stackrel{\text{def}}{=} x$ and $x_{k+1} \stackrel{\text{def}}{=} x_k/v$. Note that $|x_{k+1}| \leq |x_k|$ for all k and write $\boldsymbol{\kappa}$ for the largest index with $x_{\boldsymbol{\kappa}} \neq \varepsilon$. We let $\boldsymbol{\kappa} = -1$ if already we started with $x = \varepsilon$, and $\boldsymbol{\kappa} = \omega$ if all x_k's are non-empty, which happens iff $\mathrm{alph}(x) \not\subseteq \mathrm{alph}(v)$.

If $k \leq \boldsymbol{\kappa}$, $\mathtt{pr}[\sigma^k](x) = u^k \cdot x_k$ and x_k is a prefix of x, so taking $p_k = k$ and $\ell_k = |x_k|$ works.

If $k = \boldsymbol{\kappa} + 1$, y_k is $\mathtt{pr}[\sigma](u^{\boldsymbol{\kappa}} \cdot x_{<\ell_{\boldsymbol{\kappa}}})$. Since $x_{<\ell_{\boldsymbol{\kappa}}}/v = \varepsilon$, the result is a prefix of $u^{\boldsymbol{\kappa}+1}$, so has the form $u^{p_{\boldsymbol{\kappa}+1}}$ for some $p_{\boldsymbol{\kappa}+1}$. One also lets $l_{\boldsymbol{\kappa}+1} = 0$.

Finally, if $k > \boldsymbol{\kappa} + 1$, we have $y_k = \mathtt{pr}[\sigma^{k-\boldsymbol{\kappa}-1}](y_{\boldsymbol{\kappa}+1}) = \mathtt{pr}[\sigma^{k-\boldsymbol{\kappa}-1}](u^{p_{\boldsymbol{\kappa}+1}})$ and we just have to invoke Lemma 3.3 (and set $l_k = 0$).

(ii) Computing $\boldsymbol{\kappa}$ takes time $O(|x| + |\sigma|)$.

If $k \leq \boldsymbol{\kappa}$, comparing k with $\boldsymbol{\kappa}$ and computing p_k and ℓ_k takes additional time $O(|x| + |\sigma| + \log k)$.

If $k = \boldsymbol{\kappa} + 1$, we need to compute $\mathtt{pr}[\sigma](u^{\boldsymbol{\kappa}}\, x_{<\boldsymbol{\kappa}})$ in order to extract $p_{\boldsymbol{\kappa}+1}$. This uses eq. (6) for $O(|\sigma|)$ small steps. Note that we do not build $u^{\boldsymbol{\kappa}}$ explicitly: once x has been consumed, we work on some $u^p_{(i)}$ and just update p and i when applying some $\mathtt{pr}[?a]$, or only update p when applying some $\mathtt{pr}[!b]$, for which we only need to know where are the occurrences of b in u. For each small step, the updates can be computed in time $O(|u|+|x|)$, hence $p_{\boldsymbol{\kappa}+1}$ is computable in quadratic time.

If $k > \boldsymbol{\kappa}+1$, we set $q_0 = p_{\boldsymbol{\kappa}+1}$, $k' = k-\boldsymbol{\kappa}-1$ and aim for $\mathtt{pr}\big[\sigma^{k'}\big](x')$, starting from $x' = u^{q_0}$. We need to compute $u^{q_{k'}}$ in the sequence $u^{q_0}, u^{q_1}, \ldots$, defined by $u^{q_{i+1}} \stackrel{\text{def}}{=} \mathtt{pr}[\sigma](u^{q_i})$. Let us first compute q_1 and consider the three possibilities:
(1) If $q_0 = q_1$, y_p is a fixpoint for $\mathtt{pr}[\sigma]$ and we know $p_k = p$.
(2) If $q_0 < q_1$, the exponents increase under $\mathtt{pr}[\sigma]$ and after computing at most $|u|+1$ consecutive values, we'll find two indexes $1 \leq i < j \leq |u|+1$ such that q_i and q_j have the same fractional parts, i.e., differ by some natural number. We can then use Lemma 3.3.(ii) and compute $q_{j+\left\lfloor \frac{k'-j}{j-i} \right\rfloor} = q_j + \left\lfloor \frac{k'-j}{j-i} \right\rfloor$. From there, we're just at most $|u|$ steps from $q_{k'}$, i.e., p_k.
(3) Finally, if $q_0 > q_1$ a similar technique, now relying on Lemma 3.3.(iii), will let us compute $q_{k'}$ in polynomial time. □

The next step is to compute $\mathtt{Pre}[\sigma^*](\uparrow x)$, that is, $\uparrow x \cup \mathtt{Pre}[\sigma](\uparrow x) \cup \mathtt{Pre}[\sigma^2](\uparrow x) \cup \cdots$. Like $\mathtt{Pre}[\sigma](\uparrow x)$, this set is upward-closed. However it may have several minimal elements and one needs to collect all of them in order to represent the set faithfully.

Definition 3.5 (Iteration number)**.** The *iteration number* $L(\sigma, x)$ associated with a sequence of actions σ and a channel contents x is the smallest integer such that there exists $\ell \leq L(\sigma, x)$ with $\mathtt{pr}[\sigma^\ell][x] \preccurlyeq \mathtt{pr}[\sigma^{L(\sigma,x)+1}](x)$. Note that, by Higman's Lemma, such an integer always exists.

The point of Definition 3.5 is that it captures the number of iterations that are sufficient to compute $\mathtt{Pre}[\sigma^*](\uparrow x)$.

Lemma 3.6. $\mathtt{Pre}[\sigma^*](\uparrow x) = \bigcup_{i=0}^{L(\sigma,x)} \uparrow \mathtt{pr}[\sigma^i](x)$.

Proof. Write y_k for $\mathtt{pr}[\sigma^k](x)$ and L for $L(\sigma, x)$. By definition there is some $\ell \leq L$ with $y_\ell \preccurlyeq y_{L+1}$. By Lemma 3.2, this continues into $y_{\ell+1} \preccurlyeq y_{L+2}$, $y_{\ell+2} \preccurlyeq y_{L+3}$, etc., implying $\uparrow y_\ell \supseteq \uparrow y_{L+1}$, $\uparrow y_{\ell+1} \supseteq \uparrow y_{L+2}$, $\uparrow y_{\ell+2} \supseteq \uparrow y_{L+3}$, … Finally $\mathtt{Pre}[\sigma^*](\uparrow x)$, which is $\bigcup_{i\in\mathbb{N}} \uparrow y_i$ coincides with the finite union $\bigcup_{i=0}^{L(\sigma,x)} \uparrow y_i$. □

Theorem 3.7 (Bounding iteration numbers)**.** $L(\sigma, x) \leq |x|(|\,\mathtt{rea}(\sigma)| + 1)$ *for any action sequence* σ *and channel contents* x*.*

Proof. We write u for $\mathtt{rea}(\sigma)$. Using Theorem 3.4, we write $y_k = \mathtt{pr}[\sigma^k](x) = u^{p_k} \cdot x_{<\ell_k}$ and observe that $p_i \leq p_j$ and $\ell_i \leq \ell_j$ imply $y_i \preccurlyeq y_j$. Recall from the proof of Theorem 3.4 that $|x| = \ell_0 \geq \ell_1 \geq \cdots \geq \ell_i \geq \cdots$ is a decreasing sequence and that $p_k = k$ when $\ell_k > 0$.

There are two cases:
(1) If $(\ell_k)_k$ stabilises with some limit value ℓ_∞ that is strictly positive, then $\ell_{|x|-1} = \ell_{|x|}$ and we deduce $y_{|x|-1} \preccurlyeq y_{|x|}$, entailing $L(\sigma, x) < |x|$.
(2) If $\ell_\infty = 0$ then, writing k_0 for the first index with $\ell_{k_0} = 0$, we know that $k_0 \leq |x|$ and $p_{k_0} = k_0$. If $p_{k_0+1} \geq p_{k_0}$ then $y_{k_0} \preccurlyeq y_{k_0+1}$. Otherwise $p_{k_0} > p_{k_0+1}$ and as a consequence of Lemma 3.2 the suffix sequence $p_{k_0} > p_{k_0+1} \geq p_{k_o+2} \geq p_{k_0+3} \geq \cdots$ is decreasing. Since the p_k fractions are multiples of $\frac{1}{|u|}$, the sequence $(p_k)_{k\geq k_0}$ can only take $1 + |u|k_0$ different values and eventually yield $p_k = p_{k+1}$ for some $k \leq k_0 + k_0|u| \leq |x|(|u|+1)$, entailing $L(\sigma, x) \leq |x|(|u|+1)$ as claimed. □

The bound given by Theorem 3.7 is tight as the next simple example shows.

Example 3.8 (Bounds for $L(\sigma, x)$ are tight.). For $\sigma = !ab^5\,?b^4$ and $x = a^4$, the $(y_k)_k$ sequence with $y_k \stackrel{\text{def}}{=} \mathtt{pr}[\sigma^k](x)$ is:

$$\overbrace{a^4,\ b^4a^3,\ b^8a^2,\ b^{12}a,\ b^{16},}^{|x|=l_0>l_1>\cdots>l_4=0}\ \overbrace{b^{15},\ b^{14},\ b^{13},\ \ldots\ b,\ \varepsilon,\ \varepsilon,\ \ldots}^{k_0=4=p_{k_0}\wedge p_4\geq p_5\geq p_6\cdots}$$
$$y_0\quad y_1\quad y_2\quad y_3\quad y_4\quad y_5\quad y_6\quad y_7\qquad\qquad y_{19}\quad y_{20}\quad y_{21}$$

Since $y_{20} \preccurlyeq y_{21}$ is the earliest increasing pair, Definition 3.5 gives $L(!ab^5\,?b^4, a^4) = 20$.

This generalises to $L(!ab^{n+1}\,?b^n, a^m) = m(n+1)$ for any $n, m \in \mathbb{N}$, which is exactly the $|x|(|\,\mathtt{rea}(\sigma)| + 1)$ bound given by Theorem 3.7. □

3.3 Bounding runs

Assume that a flat LCM S is such that $(q', y) \in \mathtt{Post}^*(q, x)$. Since S is flat, the run $(q, x) \xrightarrow{*} (q', y)$ has the following shape:

$$\begin{aligned}(q,x) = (q_0,z_0) \xrightarrow{\sigma_0^{n_0}} (q_0,z_0') \xrightarrow{\theta_1} (q_1,z_1) \xrightarrow{\sigma_1^{n_1}} (q_1,z_1') \xrightarrow{\theta_2} (q_2,z_2) \xrightarrow{\sigma_2^{n_2}} \cdots \\ \cdots\ (q_{m-1},z_{m-1}') \xrightarrow{\theta_m} (q_m,z_m) \xrightarrow{\sigma_m^{n_m}} (q_m,z_m') = (q',y)\,.\end{aligned} \tag{9}$$

In eq. (9), the control locations $(q =)q_0, q_1, \ldots, q_m (= q')$ are all distinct, σ_i is the sequence of actions performed along the (unique) cycle on q_i, and n_i is the number of times this cycle has been traversed along the run. We use $\sigma_i = \varepsilon$ when there is no cycle on q_i, and we use $n_i = 0$ when the cycle is not traversed at all. For $i = 1, \ldots, m$, θ_i is the sequence of actions that labels the transition from q_{i-1} to q_i.

We say that the run in eq. (9) is *minimal* if for all $i = 1, \ldots, m$, z_i is a minimal element in $\mathtt{Pre}[\sigma_i^*](\uparrow z_i')$ and n_i is the smallest such $z_i = \mathtt{pr}[\sigma_i^{n_i}](z_i')$, and if $z_i = \mathtt{pr}[\sigma_i^{n_i}](z_i')$ for $i = 0, \ldots, m$. By allowing $z_0 \preccurlyeq x$, it is always possible to associate a minimal run with some reachability statement "$(q, x) \xrightarrow{*} (q', y)$" and use the tuple

$$\langle q_0, z_0, n_0, z_0', q_1, z_1, n_1, z_1', \ldots, q_m, z_m, n_m, y\rangle \tag{10}$$

as a witness of reachability.

We now try to bound the size of such a witness. One has

$$|z_m'| = |y|\,, \qquad |z_i| \leq |z_i'| + n_i|\,\mathtt{rea}(\sigma_i)|\,, \qquad |z_{i-1}'| \leq |z_i| + |\,\mathtt{rea}(\theta_i)|\,, \tag{11}$$

for all i. We further know from Theorem 3.7, that $n_i \leq |z_i'|(1 + |\,\mathtt{rea}(\sigma_i)|)$ for $i = 0, \ldots, m$.

Thus, writing n for the size $|S| + |x| + |y|$ of the instance (so that $m \leq n$, and $|\,\mathtt{rea}(\sigma_i)|, |\,\mathtt{rea}(\theta_i)| \leq n$ for all i), we have quadratic bounds $O(n^2)$ for n_m and $|z_m|$, cubic bounds $O(n^3)$ for n_{m-1} and $|z_{m-1}|$, …, etc., so that the witness has size $O(n^m)$, hence $2^{O(n)}$.

Unfortunately, as Example 3.9 shows, these bounds cannot be much improved upon.

Example 3.9. Consider the flat LCM S depicted in fig. 1. In S, $(q_0, \varepsilon) \xrightarrow{*} (q_0', \varepsilon)$ is witnessed by the following run schema

$$\begin{aligned}(q_0,\varepsilon) \to (q_1,ab) \xrightarrow{*} (q_1,ba^2) \to (q_2,a^2b) \xrightarrow{*} (q_2,ba^4) \to (q_3,a^4b) \xrightarrow{*} \cdots \xrightarrow{*} (q_n,ba^{2^n}) \\ \to (q_n',a^{2^n}b) \xrightarrow{*} (q_n',ba^{2^{n-1}}) \to (q_{n-1}',a^{2^{n-1}}b) \xrightarrow{*} \cdots \to (q_1',a^2b) \xrightarrow{*} (q_1',ba) \to (q_0',\varepsilon)\,.\end{aligned}$$

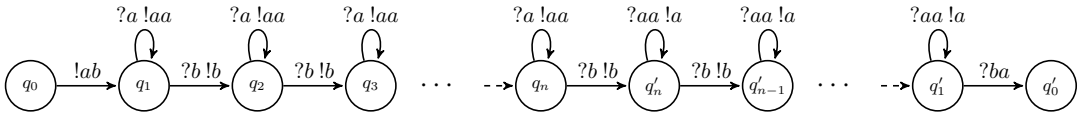

Figure 1: A flat LCM where $(q_0, \varepsilon) \xrightarrow{*} (q'_0, \varepsilon)$ requires exponential-sized configurations.

In fact, there is only one run witnessing $(q_0, \varepsilon) \xrightarrow{*} (q'_0, \varepsilon)$ and this run necessarily visits (q_n, ba^{2^n}), a configuration of exponential size, iterating 2^{n-1} times the cycle on q_n. Observe that, starting from (q_0, ε), any message loss will prevent ever reaching q'_0. □

4 SLP-compressed words and an NP algorithm for reachability

In this section we explain how the exponentially long minimal runs analysed in Section 3.3 can be handled efficiently using SLP-compressed words. This provides witnesses of polynomial size that can be validated in polynomial time, thus showing that reachability in flat LCMs is in NP.

4.1 SLP-compressed words

Compressed words are data structures used to represent long words via succinct encodings. If a long word is rather repetitive, it can have a succinct encoding of logarithmic size. Since several operations on long words or decision tests about them can be performed efficiently on the succinct representation, compressed words have been used to provide efficient solutions to algorithmic problems involving exponential-size (but rather repetitive) words, see [Loh12] for a survey.

The most studied encoding is the SLP, for Straight-Line Program, which is in effect an acyclic context-free grammar that generates a single word, called its *expansion*.

From now on, we always use small letters x, y, u, v for usual words, and capital letters X, Y, U, V for SLPs expanding to the corresponding words. Since SLPs are interpreted as plain words, we will use them freely in places where words can be used. It will always be clear when we consider the SLP as a data structure and then we use it to denote its expansion. The main situation where we want to distinguish between the two usages is when reasoning about size and algorithmic complexity: for this we write $|X|$ for the length $|x|$ of the expansion, while we write $\|X\|$ for the size of the SLP as a data structure. For example, if X expands to x then for any fractional power of the form x^p, there is an SLP X^p with $|X^p| = |x^p| = p|x|$ and $\|X^p\| = O(\|X\| + \log p)$.

In the rest of this section we will use well-known, or easy to prove, algorithmic results on SLP. In particular, all the following problems can be solved in polynomial time:

length: Given a SLP X, compute $|X|$.

factor: Given a SLP X and two positions $0 \leq i \leq j \leq |X|$, construct a SLP of size $O(\|X\|)$ for the factor $X[i : j]$.

concatenation: Given two SLPs X and Y, construct a SLP for $X \cdot Y$.

matching: Given two SLPs X and Y, decide if X is a factor (or a prefix, or a suffix) of Y.

To this list we add results tailored to our needs:

(scattered) subword with a power word: Given a SLP X, a plain word v and some power $k \in \mathbb{N}$, decide if $X \preccurlyeq v^k$. This special case of the fully compressed subsequence test can be done in time $\mathsf{poly}(\|X\| + |v| + \log k)$, see Proposition A.1 in the Appendix.

iterated LCM predecessor: Given a SLP X, a plain word v, and some power $k \in \mathbb{N}$, compute a SLP for X/v^k, i.e., for $\mathtt{pr}[(!v)^k](X)$. This can be done in time $\mathsf{poly}(\|X\| + |v| + \log k)$, see Proposition A.2 in the Appendix.

With the above results, we are ready to lift the computation of $\mathtt{pr}[\sigma^k](x)$ from plain words to SLPs:

Proposition 4.1. *Given an SLP X, a sequence of actions σ, and some $k \in \mathbb{N}$, it possible to compute an SLP Y for $\mathtt{pr}[\sigma^k](X)$ in time $\mathsf{poly}(\|X\| + |\sigma| + \log k)$.*

Proof (sketch). We follow the construction described in the proof of Theorem 3.4, now using SLPs. So again let us write u and v for $\mathtt{rea}(\sigma)$ and $\mathtt{wri}(\sigma)$.

The first step is to compute $\boldsymbol{\kappa}$. This is done by dichotomic search, since we can decide in polynomial time whether a candidate n leads to $X/v^n = \varepsilon$. We then build $X_{<\ell_{\boldsymbol{\kappa}}}$ as $X/v^{\boldsymbol{\kappa}}$.

If $k \leq \boldsymbol{\kappa}$, we build a SLP Y for $u^k \cdot (X/v^k)$ and we are done.

If $k \geq \boldsymbol{\kappa} + 1$, we compute a SLP for $y_{\boldsymbol{\kappa}+1} = u^{p_{\boldsymbol{\kappa}+1}}$ by applying $\mathtt{pr}[\sigma]$ on a SLP for $y_{\boldsymbol{\kappa}} = u^{\boldsymbol{\kappa}} \cdot x_{<\ell_{\boldsymbol{\kappa}}}$: this involves computing a SSS involving at most $2m$ operations like prefixing by a_i or computing Y/b_j. This is done in polynomial time and the exponent in $u^{p_{\boldsymbol{\kappa}+1}}$ can be computed by dividing the length of a SLP with the length of u. From there we continue as in the proof of Theorem 3.4. This involves performing a polynomial number of simple $\mathtt{pr}$ operations and some simple reasoning on the exponents. □

4.2 Reachability for flat LCMs is in NP

We now explain how eq. (10) can be replaced by an SLP-based witness of the form

$$\langle q_0, Z_0, n_0, Z'_0, q_1, Z_1, n_1, Z'_1, \ldots, q_m, Z_m, n_m, Y \rangle \,. \tag{10'}$$

Lemma 4.2. *If $\langle q_0, z_0, n_0, z'_0, q_1, n_1, z_1, z'_1, \ldots, q_m, z_m, n_m, y \rangle$ is a minimal witness for $(q, x) \xrightarrow{*} (q', y)$ in S, then there exist SLPs $Z_0, Z'_0, Z_1, \ldots, Z_m, Y$ representing $z_0, z'_0, z_1, \ldots, z_m, y$ that have size polynomial in $|S| + |y|$.*

Proof. By induction on $m - i$. We start with Y for y which does not need any compression (and let $Z'_m = Y$ for the inductive reasoning).

Then any Z_i has the shape $U_i^{p_i} \cdot (Z'_i)_{<\ell_i}$ for some p_i and ℓ_i. Now $\|(Z'_i)_{<\ell_i}\|$ is in $O(\|Z'_i\|)$ and since p_i is in $2^{O(|S|)}$ —as shown in Section 3.3—, the size of the SLP for $u_i^{p_i}$ is is $O(|u_i| + |S|)$, i.e., $O(|S|)$.

Now any Z'_{i-1} is $\mathtt{pr}[\theta_i](Z_i)$ and is easily obtained from Z_i and θ_i according to eq. (6). One can ensure that $\|Z'_i\|$ is in $O(\|Z_i\| + |S|)$.

Finally, and since each SLP has size linearly bounded in the size of the following one (the bounds propagate from right to left), we have a quadratic bound on the individual sizes for the Z_i and Z'_i, hence a cubic bound on the SLP witness overall (recall that the n_i, written in binary, have size $O(|S|)$). □

Theorem 4.3. *Deciding whether* $(q, x) \xrightarrow{*} (q', y)$ *in a flat LCM* S *is* NP*-complete.*

Proof. NP-hardness is proven in Appendix C and we just provide a NP decision algorithm.

As expected, the algorithm just guesses a SLP-based witness and checks that it is indeed a valid witness. For a positive instance of the problem, a witness exists and has polynomial size as shown in Lemma 4.2. Now checking that it is valid, i.e., that each Z_i is indeed $\mathtt{pr}[\sigma^{n_i}](Z_i')$ etc., can be done in polynomial time as shown with Proposition 4.1.[4] □

5 NP algorithms for liveness properties

We show in this section how, for flat LCMs, liveness properties like nontermination, unboundedness, and existence of a Büuchi run, effectively reduce to reachability. This only requires characterising and computing the set of configurations from which infinite runs are possible but Section 3 provides all the necessary tools.

With any sequence of channel actions σ we associate $I_\sigma \stackrel{\text{def}}{=} \bigcap_{k=0,1,2,\dots} \mathtt{Pre}[\sigma^k](\Sigma^*)$.

Lemma 5.1. $I_\sigma \subseteq \Sigma^*$ *is an upward-closed set of channel contents. It has a single minimal element or is empty.*

Proof. Write $(y_k)_{k\in\mathbb{N}}$ for the sequence $y_0 \stackrel{\text{def}}{=} \varepsilon$ and $y_{k+1} = \mathtt{pr}[\sigma](y_k)$. Then $\mathtt{Pre}[\sigma^k](\Sigma^*) = {\uparrow} y_k$ for all $k \in \mathbb{N}$ (Definition 3.1) and $I_\sigma = \bigcap_{k\in\mathbb{N}} \mathtt{Pre}[\sigma^k](\Sigma^*) = \bigcap_k {\uparrow} y_k$. From $y_0 \preccurlyeq y_1$ and monotonicity of $\mathtt{pr}$ (Lemma 3.2) we obtain $y_0 \preccurlyeq y_1 \preccurlyeq y_2 \preccurlyeq \cdots$ and ${\uparrow} y_0 \supseteq {\uparrow} y_1 \supseteq {\uparrow} y_2 \supseteq \cdots$. Thus we have

$$I_\sigma = \bigcap_{k\in\mathbb{N}} {\uparrow} y_k = \begin{cases} {\uparrow} y_K & \text{if } y_K = y_{K+1} \text{ for some } K, \\ \varnothing & \text{if the } (y_k)_{k\in\mathbb{N}} \text{ sequence is strictly increasing.} \end{cases}$$

□

We write $\mathtt{pr}[\sigma^\omega](\varepsilon) = y$ if $I_\sigma = {\uparrow} y$, and $\mathtt{pr}[\sigma^\omega](\varepsilon) = \bot$ if I_σ is empty.

Lemma 5.2. $\mathtt{pr}[\sigma^\omega](\varepsilon)$ *can be computed in time* $O(|\sigma|^3)$.

Proof (sketch). We start computing the elements $y_0, y_1, y_2, \dots$ of the $(y_k)_k$ sequence. If two consecutive values y_K and y_{K+1} coincide, we have found $\mathtt{pr}[\sigma^\omega](\varepsilon)$. Otherwise we continue while the sequence is strictly increasing until eventually $|y_k| > |\mathtt{rea}(\sigma)|$ for some k (indeed, some $k \leq 1 + |\sigma|$). In this case we can invoke Lemma 3.3.(ii) and conclude that the $(y_k)_k$ sequence will remain strictly increasing, hence $\mathtt{pr}[\sigma^\omega](\varepsilon) = \bot$.

For complexity, we note that each y_{k+1} is obtained in time $O(|\sigma| + |y_k|)$ and has length in $O(|\sigma|^2)$ since $|y_{k+1}| \leq |y_k| + |\mathtt{rea}(\sigma)|$ for all k. □

The set I_σ, represented via $\mathtt{pr}[\sigma^\omega](\varepsilon)$, is interesting because it characterises the configurations from which a σ-labelled cycle can be traversed infinitely many times, i.e., it characterises nontermination.

Indeed, the following lemma reduces nontermination to reachability:

[4] In fact, it is sufficient to guess the exponents $n_1, \dots, n_m$ for the σ_i's since the Z_i, Z_i''s can be computed from them.

Lemma 5.3 (Existence of infinite runs)**.** *(i) There exists an infinite sequence $x = x_0 \xrightarrow{\sigma} x_1 \xrightarrow{\sigma} x_2 \cdots$ starting from x if, and only if, $\mathtt{pr}[\sigma^\omega](\varepsilon) \preccurlyeq x$.*
(ii) There exists an infinite run in S that starts from (q, x) and visits a given $q' \in Q$ infinitely many times if, and only if, q' is on an elementary cycle of S and $(q, x) \xrightarrow{*} (q', \mathtt{pr}[\sigma^\omega_{q'}](\varepsilon))$.

Proof. (i) Write y for $\mathtt{pr}[\sigma^\omega](\varepsilon)$. The proof of Lemma 5.2 shows that, unless $y = \bot$, $y = \mathtt{pr}[\sigma](y)$ and thus $y \xrightarrow{\sigma} y$.
($\Longleftarrow$): Since $x \succcurlyeq y$, we have $x \xrightarrow{\sigma} y \xrightarrow{\sigma} y \xrightarrow{\sigma} \cdots$ if $\sigma \neq \varepsilon$, and $x \xrightarrow{\sigma} x \xrightarrow{\sigma} x \xrightarrow{\sigma} \cdots$ in the degenerate case where $\sigma = \varepsilon$.
($\Longrightarrow$): We assume $\sigma \neq \varepsilon$ since otherwise $x \succcurlyeq \varepsilon = \mathtt{pr}[\sigma^\omega](\varepsilon)$ holds trivially. The infinite sequence $x_0 \xrightarrow{\sigma} x_1 \xrightarrow{\sigma} x_2 \xrightarrow{\sigma} \cdots$ satisfies $x_0 \succcurlyeq \mathtt{pr}[\sigma^k](x_k) \succcurlyeq \mathtt{pr}[\sigma^k](\varepsilon)$ for all $k \in \mathbb{N}$. Thus $\mathtt{pr}[\sigma^\omega](\varepsilon) \neq \bot$ and $x = x_0 \succcurlyeq \mathtt{pr}[\sigma^\omega](\varepsilon)$.

(ii) is an immediate consequence of (i). □

By combining the above lemmas with Theorem 4.3 and the NP-hardness results proven in Appendix C, one now obtains:

Theorem 5.4. *Nontermination and existence of a Büchi run are* NP*-complete for flat LCMs.*

Remark 5.5 (Repeated coverability is NP-complete). Let us define more generally $I_\sigma(x)$ as $\bigcap_{k=0,1,2,\ldots} \mathtt{Pre}[\sigma^k](\uparrow x)$, so that I_σ really is shorthand for $I_\sigma(\varepsilon)$. For a location q on a σ_q-labelled cycle, $I_{\sigma_q}(x)$ characterises a form of *repeated coverability* since $y \in I_{\sigma_q}(x)$ iff there is an infinite run from (q, y) such that the channel contains a superword of x every time q is (re)visited. Using some temporal logic, this could be written under the form

$$y \in I_{\sigma_q}(x) \iff (q, y) \models_\exists \mathsf{GF}q \wedge \mathsf{G}(q \implies \mathit{chan} \geq x)\,.$$

The proof of Lemma 5.2 can be extended to the computation of $I_\sigma(x)$. One obtains $I_{\sigma_q}(x) = \uparrow y_0 \cap \uparrow y_1 \cap \cdots \cap \uparrow y_K$ for some K in $O(|\sigma| \cdot |x|)$. We deduce that the repeated coverability problem is in NP for flat LCMs, and is indeed NP-complete.

Note however that now the $(y_k)_k$ sequence does not necessarily satisfies $y_0 \preccurlyeq y_1$, so that $I_\sigma(x)$ will have in general several minimal elements, and possibly exponentially many. In fact already $\uparrow y_0 \cap \uparrow y_1$ may have exponentially many minimal elements (see [GLHK+20, § 6.3]). Thus the NP-algorithm for repeated coverability represents $I_\sigma(x)$ as a conjunction of subword constraints, not via a set of minimal elements, but this is sufficient for its purposes. □

Unboundedness reduces to reachability in a very similar way. We say that a sequence of actions σ is *increasing* if $u^{\ell_v} \preccurlyeq v^{\ell_v - 1}$ (and $\ell_v > 0$) for $u \stackrel{\text{def}}{=} \mathtt{rea}(\sigma)$, $v \stackrel{\text{def}}{=} \mathtt{wri}(\sigma)$ and $\ell_v \stackrel{\text{def}}{=} |v|$. Now $\mathtt{pr}[\sigma^\omega](\varepsilon)$ and increasingness of σ characterise unbounded reachability sets.

Lemma 5.6 (Proof in Appendix B.1)**.** *Let $x \in \Sigma^*$ be some channel contents and σ a sequence of channel actions. T.f.a.e.:*
(i) For all $k \in \mathbb{N}$ there exists x_k with $x \xrightarrow{\sigma^*} x_k$ and $|x_k| \geq k$.
(ii) There exists an infinite unbounded sequence $x \xrightarrow{\sigma^*} x_1 \xrightarrow{\sigma^*} x_2 \xrightarrow{\sigma^*} \cdots$ with $|x_1| < |x_2| < \cdots$.
(iii) σ is increasing and $x \succcurlyeq \mathtt{pr}[\sigma^\omega](\varepsilon)$.

Lemma 5.7 (Existence of unbounded runs)**.** *In a flat LCM, t.f.a.e.*
(i) The reachability set $\mathtt{Post}^*(q, x)$ *is infinite.*
(ii) There is an unbounded run starting from (q, x)*.*
(iii) $(q, x) \xrightarrow{*} (q', \mathtt{pr}[\sigma_{q'}^{\omega}](\varepsilon))$ *for some control location* q' *with an increasing* $\sigma_{q'}$*.*

Proof (sketch).
(*ii* $\implies$ *iii*): In an unbounded run, there must be a control location q' that is visited infinitely many times with associated channel contents that are unbounded. Since from q' one can only return to q' by running through the cycle around q', hence performing $\sigma_{q'}$ some number of times, the first visit of q' is some (q', x') satisfying case *(ii)* of Lemma 5.6. We deduce that σ'_q is increasing and that $x' \succcurlyeq \mathtt{pr}[\sigma_{q'}^{\omega}](\varepsilon)$ as in case *(iii)* of the Lemma.
(*iii* $\implies$ *ii*): by Lemma 5.6 there exists an unbounded run starting from $(q', \mathtt{pr}[\sigma_{q'}^{\omega}](\varepsilon))$. Hence there is one starting from (q, x).
(*i* $\iff$ *ii*): is an application of Kőnig's Lemma, not specific to LCMs, see e.g. [Sch10, §6]. □

We can thus reduce unboundedness to reachability of an increasing cycle. With the NP-hardness results proven in Appendix C, one now obtains:

Theorem 5.8. *Unboundedness for flat LCMs is* NP*-complete.*

6 Conclusion

We analysed the behaviour of the backward-reachability algorithm for lossy channel machines when a cycle of channel actions can be performed arbitrarily many times. This provides complexity bounds on the size of runs that follow a bounded path scheme of the form $\sigma_1^*\rho_1\sigma_2^*\rho_2 \dots \sigma_m^*\rho_m$, with applications in the verification of flat systems, or in bounded verification for general systems. The main result is an NP upper bound for reachability and, by reduction, several other verification problems like unboundedness or existence of a Büchi run.

Natural directions for future work include extending our approach to deal with richer verification problems, like temporal logic model checking. It would also be interesting to consider more expressive models, like the partially lossy channel systems from [Köc19] or the higher-order lossy channel systems and priority channel systems from [HSS14].

Acknowledgements

We thank A. Finkel who raised the issue of flatness in lossy channel systems. We also thank J. Leoux and S. Halfon for useful comments that helped improve this paper.

References

[ACBJ04] P. A. Abdulla, A. Collomb-Annichini, A. Bouajjani, and B. Jonsson. Using forward reachability analysis for verification of lossy channel systems. *Formal Methods in System Design*, 25(1):39–65, 2004.

[AČJT00] P. A. Abdulla, K. Čerāns, B. Jonsson, and Yih-Kuen Tsay. Algorithmic analysis of programs with well quasi-ordered domains. *Information and Computation*, 160(1/2):109–127, 2000.

[ADOW05] P. A. Abdulla, J. Deneux, J. Ouaknine, and J. Worrell. Decidability and complexity results for timed automata via channel machines. In *Proc. ICALP 2005*, volume 3580 of *Lecture Notes in Computer Science*, pages 1089–1101. Springer, 2005.

[AJ96] P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, 1996.

[BG99] B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using QDDs. *Formal Methods in System Design*, 14(3):237–255, 1999.

[BH99] A. Bouajjani and P. Habermehl. Symbolic reachability analysis of FIFO-channel systems with nonregular sets of configurations. *Theoretical Computer Science*, 221(1–2):211–250, 1999.

[BIK14] M. Bozga, R. Iosif, and F. Konecný. Safety problems are NP-complete for flat integer programs with octagonal loops. In *Proc. VMCAI 2014*, volume 8318 of *Lecture Notes in Computer Science*, pages 242–261. Springer, 2014.

[BIL09] M. Bozga, R. Iosif, and Y. Lakhnech. Flat parametric counter automata. *Fundamenta Informaticae*, 91(2):275–303, 2009.

[BW94] B. Boigelot and P. Wolper. Symbolic verification with periodic sets. In *Proc. CAV '94*, volume 818 of *Lecture Notes in Computer Science*, pages 55–67. Springer, 1994.

[BZ83] D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, 1983.

[CFP96] G. Cécé, A. Finkel, and S. Purushothaman Iyer. Unreliable channels are easier to verify than perfect channels. *Information and Computation*, 124(1):20–31, 1996.

[CGLM06] P. Cégielski, I. Guessarian, Y. Lifshits, and Y. V. Matiyasevich. Window subsequence problems for compressed texts. In *Proc. CSR 2006*, volume 3967 of *Lecture Notes in Computer Science*, pages 127–136. Springer, 2006.

[CJ98] H. Comon and Y. Jurski. Multiple counters automata, safety analysis, and Presburger arithmetic. In *Proc. CAV '98*, volume 1427 of *Lecture Notes in Computer Science*, pages 268–279. Springer, 1998.

[CS08] P. Chambart and Ph. Schnoebelen. The ordinal recursive complexity of lossy channel systems. In *Proc. LICS 2008*, pages 205–216. IEEE Comp. Soc. Press, 2008.

[CS10] P. Chambart and Ph. Schnoebelen. Toward a compositional theory of leftist grammars and transformations. In *Proc. FOSSACS 2010*, volume 6014 of *Lecture Notes in Computer Science*, pages 237–251. Springer, 2010.

[DDS15] S. Demri, A. K. Dhar, and A. Sangnier. Taming past LTL and flat counter systems. *Information and Computation*, 242:306–339, 2015.

[DFGvD10] S. Demri, A. Finkel, V. Goranko, and G. van Drimmelen. Model-checking CTL* over flat Presburger counter systems. *Journal of Applied Non-Classical Logics*, 20(4):313–344, 2010.

[DHL+17] N. Decker, P. Habermehl, M. Leucker, A. Sangnier, and D. Thoma. Model-checking counting temporal logics on flat structures. In *Proc. CONCUR 2017*, volume 85 of *Leibniz International Proceedings in Informatics*, pages 29:1–29:17. Leibniz-Zentrum für Informatik, 2017.

[DLS06] S. Demri, F. Laroussinie, and Ph. Schnoebelen. A parametric analysis of the state explosion problem in model checking. *Journal of Computer and System Sciences*, 72(4):547–575, 2006.

[EGM12] J. Esparza, P. Ganty, and R. Majumdar. A perfect model for bounded verification. In *Proc. LICS 2012*, pages 285–294. IEEE Comp. Soc. Press, 2012.

[Fin94] A. Finkel. Decidability of the termination problem for completely specificied protocols. *Distributed Computing*, 7(3):129–135, 1994.

[FO97] L. Fribourg and H. Olsén. A decompositional approach for computing least fixed-points of datalog programs with $\mathcal{Z}$-counters. *Constraints*, 2(3/4):305–335, 1997.

[FP19] A. Finkel and M. Praveen. Verification of flat FIFO systems. In *Proc. CONCUR 2019*, volume 140 of *Leibniz International Proceedings in Informatics*, pages 12:1–12:17. Leibniz-Zentrum für Informatik, 2019.

[FP20] A. Finkel and M. Praveen. Verification of flat FIFO systems. Long version of [FP19], submitted for publication, June 2020.

[FS01] A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, 2001.

[GI15] P. Ganty and R. Iosif. Interprocedural reachability for flat integer programs. In *Proc. FCT 2015*, volume 9210 of *Lecture Notes in Computer Science*, pages 133–145. Springer, 2015.

[GKWZ06] D. Gabelaia, A. Kurucz, F. Wolter, and M. Zakharyaschev. Non-primitive recursive decidability of products of modal logics with expanding domains. *Annals of Pure and Applied Logic*, 142(1–3):245–268, 2006.

[GLHK+20] J. Goubault-Larrecq, S. Halfon, P. Karandikar, K. Narayan Kumar, and Ph. Schnoebelen. The ideal approach to computing closed subsets in well-quasi-orderings. In *Well Quasi-Orders in Computation, Logic, Language and Reasoning*, volume 53 of *Trends in Logic*, chapter 3, pages 55–105. Springer, 2020.

[HSS14] Ch. Haase, S. Schmitz, and Ph. Schnoebelen. The power of priority channel systems. *Logical Methods in Comp. Science*, 10(4:4), 2014.

[KF11] L. Kuhtz and B. Finkbeiner. Weak Kripke structures and LTL. In *Proc. CONCUR 2011*, volume 6901 of *Lecture Notes in Computer Science*, pages 419–433. Springer, 2011.

[Köc19] Ch. Köcher. Reachability problems on partially lossy queue automata. In *Proc. RP 2019*, volume 11674 of *Lecture Notes in Computer Science*, pages 149–163. Springer, 2019.

[KS15] P. Karandikar and Ph. Schnoebelen. Generalized Post embedding problems. *Theory of Computing Systems*, 56(4):697–716, 2015.

[Loh12] M. Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012.

[LPS14] J. Leroux, V. Penelle, and G. Sutre. The context-freeness problem is coNP-complete for flat counter systems. In *Proc. ATVA 2014*, volume 8837 of *Lecture Notes in Computer Science*, pages 248–263. Springer, 2014.

[LS05] J. Leroux and G. Sutre. Flat counter automata almost everywhere! In *Proc. ATVA 2005*, volume 3707 of *Lecture Notes in Computer Science*, pages 489–503. Springer, 2005.

[LW08] S. Lasota and I. Walukiewicz. Alternating timed automata. *ACM Trans. Computational Logic*, 9(2), 2008.

[MS04] N. Markey and Ph. Schnoebelen. A PTIME-complete matching problem for SLP-compressed words. *Information Processing Letters*, 90(1):3–6, 2004.

[Sch10] Ph. Schnoebelen. Lossy counter machines decidability cheat sheet. In *Proc. RP 2010*, volume 6227 of *Lecture Notes in Computer Science*, pages 51–75. Springer, 2010.

[Sch16] S. Schmitz. Complexity hierarchies beyond Elementary. *ACM Trans. Computation Theory*, 8(1), 2016.

[SS11] S. Schmitz and Ph. Schnoebelen. Multiply-recursive upper bounds with Higman's lemma. In *Proc. ICALP 2011*, volume 6756 of *Lecture Notes in Computer Science*, pages 441–452. Springer, 2011.

[VF80] B. Vauquelin and P. Franchi-Zannettacci. Automates à file. *Theoretical Computer Science*, 11(2):221–225, 1980.

[YBIT11] T. Yamamoto, H. Bannai, S. Inenaga, and M. Takeda. Faster subsequence and don't-care pattern matching on compressed texts. In *Proc. CPM 2011*, volume 6661 of *Lecture Notes in Computer Science*, pages 309–322. Springer, 2011.

A Some SLP algorithms

We describe here some SLP algorithms that are not readily available in the literature (as far as we know). Formally, by "an SLP X" we mean a grammar (Σ, N, X, P) where $X \in N$ is the axiom (a non terminal), where Σ is the set of terminal letters, and where the production rules in P are either $A_i \to a$ or $A_i \to A_j A_k$ for some $a \in \Sigma$ and some nonterminals A_i, A_j, A_k with $i < j, k$. There is exactly one production rule for each $A_i \in N$, so that each A_i defines a unique word $L(A_i) \in \Sigma^*$.

A.1 Deciding $X \preccurlyeq v^n$

Deciding $X \preccurlyeq Y$ between SLPs is a difficult problem, PP-hard as show in [Loh12]. When x (or y) is a plain word, the problem has polynomial-time solutions [MS04, CGLM06, YBIT11].

Here we consider the special case where Y is some v^n.

Proposition A.1. *Deciding whether $X \preccurlyeq v^n$, where X is an SLP, v is a plain word, and n is a fractional exponent, can be done in time $O(\|X\| \cdot |v| + |v|^2 + \log n)$.*

Proof. For $v \neq \varepsilon$ and some word x such that $\mathrm{alph}(x) \subseteq \mathrm{alph}(v)$, let us define $p(x, v)$ as the smallest fractional power such that $x \preccurlyeq v^p$. Now $p(x, v)$ satisfies the following equalities:

$$\begin{aligned} p(\varepsilon, v) &= 0 \\ p(a, v) &= \frac{i}{|v|}, \text{ if the first occurrence of } a \text{ in } v \text{ is at position } i, \\ p(x\,y, v) &= p(x, v) + p(y, v_{(j)}), \text{ if } p(x, v) \text{ is some } q + \frac{j}{|v|} \text{ with } q \in \mathbb{N}. \end{aligned} \tag{12}$$

Using eq. (12) leads to a dynamic programming algorithm computing $p(X, v)$ for an SLP X. After checking that $\mathrm{alph}(X) \subseteq \mathrm{alph}(v)$, one computes the values of all $p(A, v_{(i)})$ for $i = 1, \ldots, |v|$ and A a nonterminal in SLP X. Each of these $O(\|X\| \cdot |u|)$ values is computed in time $O(1)$ if one precomputes the first occurrences of letters in the cyclic shifts of v, say in time $O(|v|^2)$. Finally, one only has to compare $p(X, v)$ with n. □

A.2 Computing X/v^k

Proposition A.2. *Building a SLP for X/v^k, where X is an SLP, v is a plain word, and $k \in \mathbb{N}$, can be done in time* $\mathsf{poly}(\|X\| + |v| + \log n)$.

Proof. For given ℓ, deciding whether X/v^k has length at least ℓ is easy: One just applies the definition, builds an SLP X' for the suffix of length $|X| - \ell$ of X, and checks that it is a subword of v^k with proposition A.1.

Thus one can computes $|X/v^k|$ by finding the length of the result via dichotomic search, repeating the previous process $\log |X|$, i.e., $O(\|X\|)$, times.[5] □

[5] A better, dynamic programming, algorithm exists but here we aim for the simplest feasability proof.

B Forward reachability techniques

We collect in this section some proofs relying on forward-reachability analysis.

Let us reuse notations from [ACBJ04] and define a partial function $x \ominus u$ between channel contents as follows:

$$x \ominus u \stackrel{\text{def}}{=} \begin{cases} x & \text{if } u = \varepsilon, \\ \text{undefined} & \text{if } x = \varepsilon \text{ and } u \neq \varepsilon, \\ x' \ominus u' & \text{if } x = ax' \text{ and } u = au' \text{ for some } a \in \Sigma, \\ x' \ominus u & \text{if } x = ax' \text{ and } u = bu' \text{ for some } a \neq b \in \Sigma. \end{cases} \tag{13}$$

Observe that $x \ominus u$ is defined if, and only if, $u \preccurlyeq x$. Note also that, when $x \ominus u$ is defined, we can use monotonicity and commutation with concatenation:

$$\text{if } u \preccurlyeq x \text{ then for all } x' : \begin{cases} x \preccurlyeq x' \text{ implies } x \ominus u \preccurlyeq x' \ominus u\,, \\ (x \ominus u) \cdot x' = (xx') \ominus u\,. \end{cases} \tag{14}$$

Now $x \ominus u$ captures the forward effects of $?u$ actions in LCMs:

Lemma B.1. *$x \xrightarrow{?u} y$ iff $y \preccurlyeq x \ominus u$.*

We can also use $\ominus$ to characterise the outcome of arbitrary sequences of actions.

Lemma B.2. *Let $\sigma \in Act_\Sigma^*$ be an arbitrary sequence of actions.*

$$x' \xrightarrow{\sigma} y \text{ for some } x' \preccurlyeq x \text{ iff } x \xrightarrow{\sigma} \text{ and } y \preccurlyeq \big(x \cdot \mathtt{wri}(\sigma)\big) \ominus \mathtt{rea}(\sigma)\,.$$

Proof. By induction on the length of σ. The existential quantification on some $x' \preccurlyeq x$ accounts for the case where $\sigma = \varepsilon$ is the empty sequence.

For the inductive step, we consider two cases:

1. $\sigma = !w \cdot \sigma'$: For the "$\Longrightarrow$" direction, $x \xrightarrow{\sigma} y$ implies $x' \xrightarrow{\sigma'} y$ for some $x' \preccurlyeq xw$, which implies

$$\begin{aligned} y &\preccurlyeq \big(x'w \cdot \mathtt{wri}(\sigma')\big) \ominus \mathtt{rea}(\sigma') && \text{by ind. hyp.,} \\ &= \big(x' \cdot \mathtt{wri}(\sigma)\big) \ominus \mathtt{rea}(\sigma) && \text{since } \sigma = !w \cdot \sigma', \\ &\preccurlyeq \big(x \cdot \mathtt{wri}(\sigma)\big) \ominus \mathtt{rea}(\sigma) && \text{by monotonicity.} \end{aligned}$$

   For the "$\Longleftarrow$" direction, we know that $y \preccurlyeq \big(x.\,\mathtt{wri}(\sigma)\big) \ominus \mathtt{rea}(\sigma) = \big(xw.\,\mathtt{wri}(\sigma')\big) \ominus \mathtt{rea}(\sigma')$, so the ind. hyp. tells us that $x' \xrightarrow{\sigma'} y$ for some $x' \preccurlyeq xw$. We deduce $x \xrightarrow{!w} x' \xrightarrow{\sigma'} y$.

2. $\sigma = ?w \cdot \sigma'$: For the "$\Longrightarrow$" direction, $x' \xrightarrow{\sigma} y$ implies $x' \xrightarrow{?w} x'' \xrightarrow{\sigma'} y$ for some $x'' \preccurlyeq x' \ominus w$. We have

$$\begin{aligned} y &\preccurlyeq \big(x''.\,\mathtt{wri}(\sigma')\big) \ominus \mathtt{rea}(\sigma') && \text{by ind. hyp.,} \\ &\preccurlyeq \big([x \ominus w] \cdot \mathtt{wri}(\sigma')\big) \ominus \mathtt{rea}(\sigma') && \text{by monotonicity,} \\ &= \big(x \cdot \mathtt{wri}(\sigma')\big) \ominus \big(w \cdot \mathtt{rea}(\sigma')\big) && \text{by (14),} \\ &= (x \cdot \mathtt{wri}(\sigma)) \ominus \mathtt{rea}(\sigma) && \text{since } \sigma = ?w \cdot \sigma'. \end{aligned}$$

For the "$\Longleftarrow$" direction, we know that $x \xrightarrow{\sigma}$ hence in particular $x \ominus w$ is defined. We also know that $y \preccurlyeq \big(x.\,\mathtt{wri}(\sigma)\big) \ominus \mathtt{rea}(\sigma) = \big(x.\,\mathtt{wri}(\sigma')\big) \ominus (w \cdot \mathtt{rea}(\sigma')) = \big((x \ominus w).\,\mathtt{wri}(\sigma')\big) \ominus \mathtt{rea}(\sigma')$, so by ind. hyp. there is some $x' \preccurlyeq x \ominus w$ with $x' \xrightarrow{\sigma'} y$ for some $x' \preccurlyeq xw$. We deduce $x \xrightarrow{?w} x' \xrightarrow{\sigma'} y$.

□

B.1 Proof of Lemma 5.6

Write u, v for $\mathtt{rea}(\sigma)$, $\mathtt{wri}(\sigma)$.

(*ii* $\implies$ *iii*): we only have to prove that σ_q is increasing since Lemma 5.3 entails $x \succcurlyeq \mathtt{pr}[\sigma^\omega](\varepsilon)$ already.

By assumption, there is a sequence $x_1, x_2, \ldots$ of channel contents of increasing length, and some numbers $n_1, n_2, \ldots$ in $\mathbb{N}$ such that $x \xrightarrow{\sigma^{n_i}} x_i$. W.l.o.g. we can assume $n_1 < n_2 < \cdots$.

With Lemma B.2 we deduce $x_i \preccurlyeq (x\,v^{n_i}) \ominus u^{n_i}$, hence $u^{n_i}\,x_i \preccurlyeq x\,v^{n_i}$, for all $i = 1, 2, \ldots$ If $u = \varepsilon$, σ is trivially increasing, so assume $|u| > 0$ and write $m = |x|$: we get $u^{n_i - m} x_i \preccurlyeq v^{n_i}$ for all i such that $n_i \geq m$. Now take i such that $|x_i| \geq (m+1)|v|$ (and such that $n_i > m$): we get $u^{n_i - m} \preccurlyeq v^{n_i - m - 1}$. We now applies Lemma 6.2 from [ACBJ04]: "if there is some $k \geq 1$ such that $w_1^k \preccurlyeq w_2^{k-1}$ (for two words w_1, w_2), then in particular one can choose $k = |w_2|$". This yields $u^{|v|} \preccurlyeq v^{|v|-1}$, i.e., σ is increasing.

(*iii* $\implies$ *i*): we assume that σ is increasing, i.e., $u^{|v|} \preccurlyeq v^{|v|-1}$, and that $x \succcurlyeq \mathtt{pr}[\sigma_q^\omega](\varepsilon)$. The second assumption entails that $x \xrightarrow{\sigma^n}$ for all n. The first assumption entails $v^k \preccurlyeq x v^{k|v|} \ominus u^{k|v|}$, hence $x \xrightarrow{\sigma^{k|v|}} v^k$ by Lemma B.2, for all $k \in \mathbb{N}$.

(*i* $\implies$ *ii*): is an application of Kőnig's Lemma, not specific to LCMs, see e.g. [Sch10, §6].

C NP-hardness for flat LCMs and flat FIFO machines

LCMs are derived from FIFO automata [VF80, BZ83] and our NP-hardness results apply to both models. *FIFO automata*, sometimes called *queue automata*, or *communicating finite state machines*, are reliable channel machines where messages are never lost. Their operational semantics is based on a reliable notion of steps, formally given by $x \xrightarrow{!\,w}_{\mathrm{rel}} y \iff^{\mathrm{def}} y = xw$ and $x \xrightarrow{?\,w}_{\mathrm{rel}} y \iff^{\mathrm{def}} wy = x$, to be compared with Equation (2). This is extended to $x \xrightarrow{\sigma}_{\mathrm{rel}} y$, $c \xrightarrow{*}_{\mathrm{rel}} c'$, etc., as for LCMs.

C.1 Proof of Theorem 2.1: NP-hardness for acyclic machines

We first show hardness for reachability and reduce from SAT. Let $\varphi = C_1 \wedge \cdots \wedge C_m$ be a 3CNF with Boolean variables among $V = \{v_1, \ldots, v_n\}$. With φ we associate a machine S_φ as illustrated below in fig. 2.

Let us explain informally how S_φ operates. Starting from I^{b} it first reaches I^{e} while writing in the channel a word of the form $w\,\$$ with $w \in \{0,1\}^n$. This word encodes a valuation of the Boolean variables and carries an end marker $\$$. Then S_φ crosses from C_1^{b} to C_1^{e}: this requires reading the valuation on the channel and checking that it satisfies C_1.

For this S_φ has to choose the line corresponding to one of the three literals in C_1, in fact choose one literal made true by the valuation. During this check, the valuation is written back on the channel. Then S_φ checks that the remaining clauses, C_2 to C_m, are satisfied by the valuation, each time reading the valuation and writing it back on the channel. Finally, the last leg from V^{b} to V^{e} checks that no message has been lost during all this run.

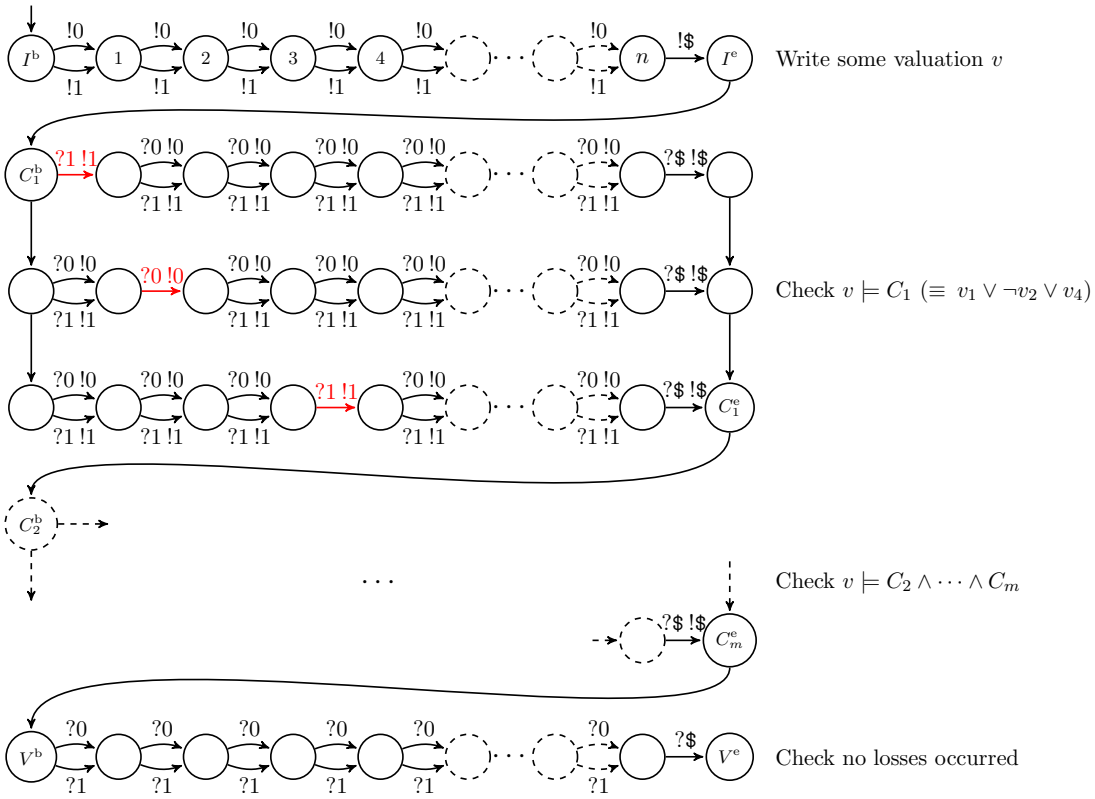

Figure 2: LCM S_φ for satisfiability of $\varphi = (v_1 \vee \neg v_2 \vee v_4) \wedge C_2 \cdots \wedge C_m$.

It is now clear that $(I^{\mathrm{b}}, \varepsilon) \xrightarrow{*} (V^{\mathrm{e}}, \varepsilon)$ in S_φ if, and only if, φ is satisfiable. The reasoning holds for lossy LCMs and for reliable FIFO automata. We have thus reduced SAT to the reachability problem for both types of acyclic machines.

Remark C.1. The construction of S_φ can be simplified at the cost of making the reduction perhaps less obviously correct: one can either omit the end-marker symbol \$ since in the end the machine checks that no message was lost (thus a binary alphabet suffices), or one can stop the machine at C^{e}_m, getting rid of the V^{b} to Ve part, since the markers ensure that the valuation read while checking a clause C_i is indeed the full valuation written at the previous stage. □

For hardness of nontermination and unboundedness we adapt the previous reduction by adding a single cycle $V^{\mathrm{e}} \xrightarrow{!\$} V^{\mathrm{e}}$ on the last control location. Starting from $(I^{\mathrm{b}}, \varepsilon)$, the modified S_φ has an infinite run iff it has an unbounded run iff φ is satisfiable.

The above reductions adapt to flat VASSes and lossy VASSes, i.e., channel machines with unary alphabet, provided that we allow $2n$ channels (or counters) for a valuation on

n Boolean variables.

C.2 Proof of Theorem 2.2: NP-hardness for single-path machines

We first show hardness for reachability. For this we reduce from SAT. So let us consider a 3CNF formula φ with Boolean variables among $V = \{v_1, \ldots, v_n\}$. Let us say $\varphi = (v_2 \vee \neg v_3 \vee \neg v_n) \wedge C_2 \wedge \cdots \wedge C_m$, with m clauses.

With φ we associate S_φ, the single-path flat LCM described in Figure 3. This LCM has $O(mn^2)$ control locations[6], and is organised as a series of distinct operations on the channel contents. The operations are grouped in lines and we describe them informally.

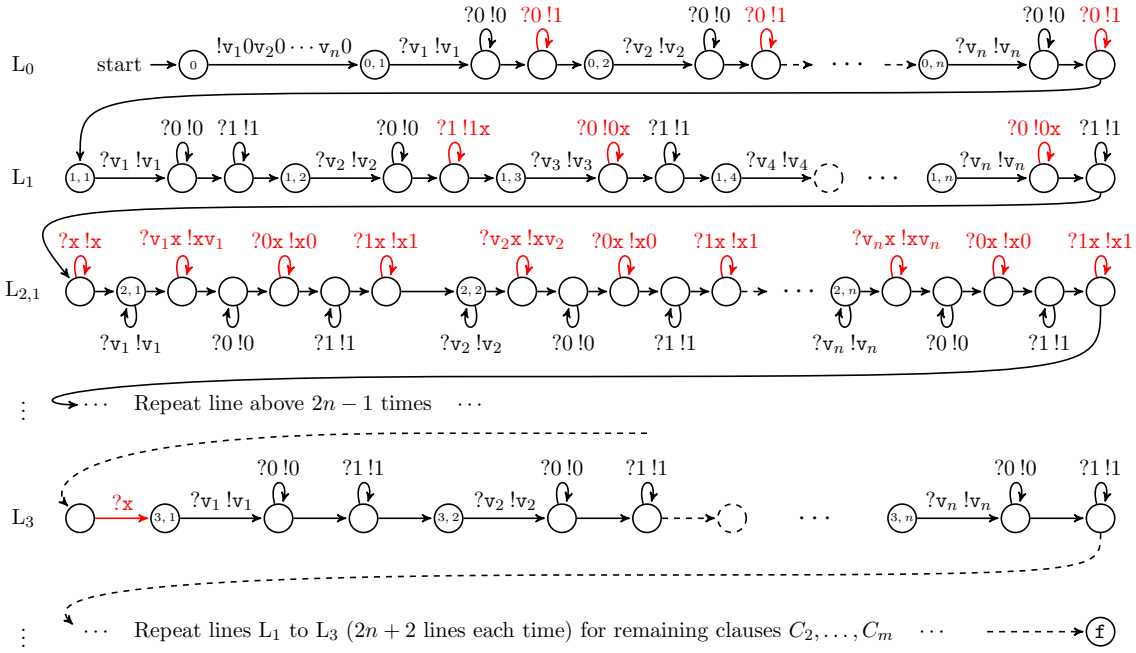

Figure 3: Single-path LCM for satisfiability of $\varphi = (v_2 \vee \neg v_3 \vee \neg v_n) \wedge C_2 \cdots \wedge C_m$.

L_0, choosing a valuation nondeterministically: S_φ first write $\mathtt{v}_1 0 \mathtt{v}_2 0 \ldots \mathtt{v}_n 0$ on the channel. This is our encoding for the valuation that is 0 for all variables. Then S_φ reads the valuation and write it back, possibly changing any 0 value with a 1 (this happens at the red-coloured actions), and thus picking an arbitrary valuation nondeterministically. Here we see how the $\mathtt{v}_1, \ldots, \mathtt{v}_n$ markers are used to check positions inside the valuation.

L_1, marking where clause C_1 is validated: S_φ now checks whether the valuation stored on the channel makes C_1 true. In this example, we assume that C_1 is $v_2 \vee \neg v_3 \vee \neg v_n$. Again S_φ reads the valuation and writes it back. However, if it reads $\mathtt{v}_2 1$ or $\mathtt{v}_3 0$ or $\mathtt{v}_n 0$, it writes it back followed by a special checkmark symbol $\mathtt{x}$ that "means C_1 has been validated" (see red actions). Note that as many as 3 occurrences of $\mathtt{x}$ can be inserted in the encoding of the valuation.

[6]Our reduction insists on using only one channel. With multiple channels the same idea would use $O(n + m)$ control locations.

$L_{2,1}$, pushing `x` to the head of the valuation encoding: S_φ now pushes any checkmark symbol to the left. This is done along the $L_{2,1}$ line. While the valuation is read and written back as usual (black actions), any symbol preceding a `x` can swap position with it (red actions).

$L_{2,2}$, ..., $L_{2,2n}$, more pushing `x` to the left: this behaviour is repeated $2n$ times in total, so that any `x` can be pushed completely to the left of the valuation. In case of multiple occurrences of `x`, we just need one of them to reach the head of the valuation so we assume that the other ones will just be lost.

L_3, checking that clause C_1 has been validated: Now S_φ knows where to expect `x`. The machine can only proceed if indeed a `x` is present in the channel, in front of the valuation, and thus if the valuation on the channel satisfies C_1. The rest of the line reads and writes back the valuation, clearing it of any remaining `x`'s.

Same treatment for the remaining clauses $C_2, \ldots, C_m$: S_φ now continues with similar locations and rules checking that the remaining clauses are validated.

Note that, once the valuation has been picked nondeterministically (in L_1), it cannot be modified. Also note that the machine will block if one of the $\mathtt{v}_i$ markers is lost before the last clause has been validated. If one of the 0/1 values of the valuation is lost, this value cannot be used any more for checkmarking a validated clause. Such message losses do not lead to any incorrect behaviour, they can only hinder the validation of a clause.

Finally, starting from $(0, \varepsilon)$, S_φ can reach its final location `f` iff φ is satisfiable.

Now the reduction extends to show prove NP-hardness of unboundedness for single-path LCMs with exactly the same adaptation as in the proof for acyclic LCMs. For hardness of nontermination a little more work is needed since every cycle where S_φ reads the valuation and writes it back could become a nonterminating cycle if all but one letter are lost. One possible trick to overcome this is to have two copies of the alphabet, say of two different colours, and to ensure that in all its phases the machine reads in one colour and writes back in the other, so that the valuation is always read and written in alternating colours. Once this is implemented, the system cannot have infinite runs as is. Adding a single loop on `f`, the final control location, as we did for acyclic LCMs, now provides a correct reduction from SAT to nontermination for single-path LCMs.

The idea behind this reduction can easily be adapted so that it applies to single-path VASSes and lossy VASSes, or equivalently, to channel machines with a unary alphabet. One uses $2n$ channels (or counters) for storing the valuation and m distinct counters for marking the clauses that have been validated.

Restricting to a binary alphabet on a single channel is equally easy for reliable FIFO automata, but more difficult when message losses have to be taken care of. Therefore we won't attempt it in this preliminary version.

D Multiple channels

The analysis we conducted in Section 3 carries over without any difficulty to systems with multiple channels. Lemma 3.3 and Theorem 3.4 remain valid since, once σ and k have been

fixed, computing $\mathtt{pr}[\sigma^k](\langle x_1, \ldots, x_c\rangle)$ for a system with c channels can be done independently for each of the c channels: one only needs to distribute the actions on σ to their corresponding channel, so that $\mathtt{rea}(\sigma)$ now is some tuple $\langle u_1, \ldots, u_c\rangle$. In particular the bound in Theorem 3.7 becomes

$$L\big(\sigma, \langle x_1, \ldots, x_c\rangle\big) \leq \max_{i=1}^{c} |x_i| \cdot (|u_i| + 1)\,, \quad \text{where } \mathtt{rea}(\sigma) = \langle u_1, \ldots, u_c\rangle\,.$$