The Ordinal Recursive Complexity of Lossy Channel Systems*

P. Chambart and Ph. Schnoebelen
LSV, ENS Cachan & CNRS
61 av Pdt. Wilson, F-94230 Cachan, France

Abstract

We show that reachability and termination for lossy channel systems is exactly at level $\mathfrak{F}_{\omega^\omega}$ in the Fast-Growing Hierarchy of recursive functions, the first level that dominates all multiply-recursive functions.

1 Introduction

Lossy channel systems, or shortly LCS's, are systems of finite-state automata communicating asynchronously via unbounded FIFO channels that are unreliable ("lossy") in the sense that they can nondeterministically lose messages. While systems with reliable ("perfect") channels are Turing-powerful, systems with lossy channels can be algorithmically verified, as was first observed by Abdulla and Jonsson [4] and Finkel et al. [16, 8]. The two main positive results for LCS's concern their reachability and termination properties, for which decidability relies on arguments from the theory of well-quasi-orderings [17, 5], and more precisely Higman's lemma.

The complexity of LCS's. Ever since these early positive results, the main open question has been to assess the algorithmic complexity of LCS's since no upper bounds had been extracted from the decidability proofs, and no interesting lower bounds were available for comparison. A partial answer was given in [26] where it is shown that LCS's can simulate perfect systems whose space is bounded by Ackermann's function. Hence the complexity of their verification problems cannot be primitive-recursive. This lower bound directly applies to problems that are equivalent to LCS verification, e.g., the "Regular Post Embedding Problem" introduced in [9, 10], or that can simulate LCS's, see [21, 19, 3, 14, 18, 22] among others.

Our contribution. We show that lossy channel systems can compute in a weak sense the functions F_α from the Fast-Growing Hierarchy for all $\alpha < \omega^\omega$, and use this to simulate perfect channels whose size is bounded by $F_{\omega^\omega}(n)$. This relies on an original "*differential encoding*" for lists of increasing ordinals that is simple enough to allow correct manipulation by finite-state transductions in the style of channel systems while, at the same time, being differential makes it robust in presence of message losses.

The consequence is that LCS verification cannot belong to $\mathfrak{F}_{<\omega^\omega}$, i.e., the class of all multiply-recursive functions [28, 15]. However, it belongs to $\mathfrak{F}_{\omega^\omega}$ as we explain in section 6, heavily relying on Cichon and Tahhan Bittar's analysis of Higman's lemma [12]. These results give a tight measure of the worst-case complexity and further show that the key parameter is the size of the message alphabet (and not the number of channels, or size of the control part). They also directly apply to all the problems with which LCS verification has been connected (see above). Beyond these applications, we hope that this work will help disseminate some fundamental proof-theoretical techniques and results on subrecursive hierarchies that are relevant to verification techniques based on well-quasi-orderings.

Related work. The study of the complexity of Higman's Lemma was initiated by de Jongh and Parikh [13] who measured the maximum order-type compatible with the subword ordering. Constructive proofs of Higman's Lemma provide recursive upper bounds that are inherited from the computational power of the underlying logical framework, and are thus exaggeratedly high. Using clever combinatorial reasonings, Cichon and Tahhan Bittar [12] were the first to provide tight upper bounds for the length of bad sequences w.r.t. the subword ordering.[1] Our construction does not only show that long bad sequences can be produced by something as weak as lossy channel systems: it shows how such systems can check that the long computations did actually not lose any message. Such reductions are scarce in the field of infinite-state systems verification (but they exist in the neighbouring field of automated deduction).

*Work supported by the Agence Nationale de la Recherche, grant ANR-06-SETIN-001.

[1] An earlier $\mathfrak{F}_\omega$ upper bound for bad sequences in $\mathbb{N}^k$ (Dickson's Lemma) was provided by McAloon [24].

2 The Fast-Growing Hierarchy

The *Fast-Growing Hierarchy*, also called the *Extended Grzegorczyk Hierarchy*, is a class $(F_\alpha)_\alpha$ of number-theoretic functions indexed by (an initial segment of the) ordinals and called *ordinal recursive functions* [28]. Our exposition is based on [25, 15, 12] where more details can be found.

Each function $F_\alpha : \mathbb{N} \to \mathbb{N}$ eventually dominates the lower functions F_β for $\beta < \alpha$. Furthermore, writing $\mathfrak{F}_\alpha$ for the smallest class of functions containing F_α, addition, zero, projections, and closed under compositions and limited recursion, one obtains a strict hierarchy of subrecursive functions. Write $\mathfrak{F}_{<\alpha}$ for $\bigcup_{\beta<\alpha}\mathfrak{F}_\beta$: it is known that $\mathfrak{F}_{<\omega}$ is exactly the set of primitive-recursive functions, that $\mathfrak{F}_{<\omega^k}$ is the set of Péter's k-recursive functions for any $k = 1, 2, \ldots$, that $\mathfrak{F}_{<\omega^\omega}$ is the set of multiply-recursive functions, and that $\mathfrak{F}_{<\varepsilon_0}$ is the set of functions provably total in first-order Peano arithmetic.

Ordinals below ω^ω. In this paper we use Ω to denote the ordinal ω^ω. We shall work with set-theoretical ordinals less than Ω, written in Cantor's Normal Form.

We say that a given ordinal $0 < \alpha < \Omega$ has *degree* $d \in \mathbb{N}$, written $deg(\alpha) = d$, if $\omega^{d+1} > \alpha \geq \omega^d$. In that case, α can be decomposed in a unique way under the form $\alpha = \omega^d.a + \alpha'$ with $0 < a \in \mathbb{N}$ and $\alpha' < \omega^d$. (We further let $deg(0) = 0$.) For any $p \geq deg(\alpha)$, $\alpha < \Omega$ can be written in a unique way under the form $\alpha = \omega^p.a_p + \omega^{p-1}.a_{p-1} + \cdots + \omega^1.a_1 + \omega^0.a_0$, shortly written $\sum_{i\leq p}\omega^i.a_i$, with $a_0, \ldots, a_p \in \mathbb{N}$. The size $|\alpha|$ of $\alpha = \sum_{i\leq p}\omega^i.a_i$ is $\sum_{i\leq p} a_i$.

Assume $\alpha = \sum_{i\leq p}\omega^i.a_i$ and $\beta = \sum_{i\leq p}\omega^i.b_i$ are two ordinals below Ω. We say that α *embeds in* β, written $\alpha \sqsubseteq \beta$, when $a_i \leq b_i$ for all $i = 0, \ldots, p$. Observe that embedding between ordinals is only a partial order (in which, e.g., ω and 1 are incomparable), compatible with the usual linear ordering of ordinals.

The set of limit ordinals $\leq \Omega$ is denoted *Lim*. Each $\lambda \in Lim$ comes with its canonical *fundamental sequence* $(\lambda_n)_{n\in\mathbb{N}}$ satisfying $\lambda_0 < \lambda_1 < \cdots < \lambda_n < \lambda_{n+1} < \cdots$ and $\lambda = \sup_n \lambda_n$. For limit ordinals below Ω, the fundamental sequence is given by

$$\left(\sum_{i\leq p}\omega^i.a_i\right)_n \stackrel{\text{def}}{=} \omega^p.a_p + \cdots + \omega^{r+1}.a_{r+1} + \omega^r(a_r - 1) + \omega^{r-1}.n$$

assuming a_r is the last nonzero coefficient, i.e., $0 = a_0 = a_1 = \ldots = a_{r-1} < a_r$. Equivalently,

$$\left((\alpha+1).\omega^{i+1}\right)_n = \alpha.\omega^{i+1} + \omega^i.n \quad \text{for all } \alpha < \Omega \text{ and } i \in \mathbb{N}.$$

For example, if $\lambda = \omega^9.2 + \omega^3.6$, then $\lambda_n = \omega^9.2 + \omega^3.5 + \omega^2.n$. Observe that, for all $\lambda \in Lim$, $\lambda_n \sqsubseteq \lambda_{n+1}$ and $|\lambda_n| = |\lambda| + n - 1$. This scheme extends canonically up to ε_0 (and beyond) with $(\omega^\lambda)_n \stackrel{\text{def}}{=} \omega^{\lambda_n}$ etc. [25, 15].

Fast-growing functions and monotonicity. The functions $F_\alpha : \mathbb{N} \to \mathbb{N}$ are defined by induction over α:

$$F_0(n) \stackrel{\text{def}}{=} n + 1, \tag{D1}$$

$$F_{\alpha+1}(n) \stackrel{\text{def}}{=} F_\alpha^{n+1}(n) = \overbrace{F_\alpha(F_\alpha(\ldots F_\alpha}^{n+1 \text{ times}}(n)\ldots)), \tag{D2}$$

$$F_\lambda(n) \stackrel{\text{def}}{=} F_{\lambda_n}(n) \qquad \text{if } \lambda \in Lim. \tag{D3}$$

This induces $F_1(n) = 2n + 1$ and $F_2(n) = (n+1)2^{n+1} - 1$. Expanding $F_3(n)$ needs a tower of n exponents. $F_\omega(n) = F_n(n)$, so that F_ω is a variant of Ackermann's function and is the first F_α that is not primitive-recursive. With $F_{\omega^\omega}(n) \stackrel{\text{def}}{=} F_{\omega^n}(n)$, F_{ω^ω} is a kind of "multiply-Ackermann" function, the first F_α that is not k-recursive for some $k \in \mathbb{N}$.

Since we later construct a channel system that evaluates the F_α functions for $\alpha < \Omega$, it is a good exercise for the reader to try and get some intuition of what would $F_{\omega+1}(n)$, $F_{\omega+2}(n)$, $F_{\omega.2}(n)$ and $F_{\omega^2}(n)$ look like. For example

$$\begin{aligned} F_{\omega^2.3}(5) &= F_{\omega^2.2+\omega.5}(5) \\ &= F_{\omega^2.2+\omega.4+5}(5) \\ &= \underbrace{F_{\omega^2.2+\omega.4+4}(\ldots(F_{\omega^2.2+\omega.4+4}}_{6 \text{ times}}(5))\ldots). \end{aligned}$$

We now state some standard monotonicity properties in the form that will be convenient for our later developments.

Lemma 2.1 (Monotonicity) *For every $\alpha < \Omega$ and $n \in \mathbb{N}$:*

$$n < F_\alpha(n), \tag{2.1.a}$$

$$F_\alpha(n) \leq F_\alpha(n+1), \tag{2.1.b}$$

$$|\alpha| < F_\alpha(n) \qquad \text{if } n > 0. \tag{2.1.c}$$

In general, $\beta < \alpha$ does not entail $F_\beta(n) \leq F_\alpha(n)$, e.g., $F_m(n) > F_\omega(n)$ when $0 < n < m < \omega$. What is true is that, for all $\beta < \alpha$, F_β is *eventually* dominated by F_α, i.e., $F_\beta(n) < F_\alpha(n)$ for n large enough.

The next lemma provides more precise information on this issue.

Lemma 2.2 (Monotonicity w.r.t. α) *For every $\alpha, \beta, \gamma < \Omega$ and $n, p \in \mathbb{N}$:*

$$F_\beta(n) \leq F_\alpha(n) \qquad \text{if } \beta \sqsubseteq \alpha, \tag{2.2.a}$$

$$F_{\gamma+\alpha}(n) \leq F_{\gamma+\omega^p+\alpha}(n) \qquad \text{if } n > |\gamma|. \tag{2.2.b}$$

Observe that (2.2.b) is not a special case of (2.2.a) since $\gamma + \alpha \sqsubseteq \gamma + \omega^p + \alpha$ does not hold in general.

We now prove Lemmas 2.1 and 2.2. The first four inequalities are proved by induction over α. We sometimes use simultaneous induction as when proving (2.1.b) and (2.2.a). Proving (2.2.b) requires the introduction of extra

notations and tools, and is done in a later step.

2.1.a. $F_\alpha(n) > n$:

An easy induction over α. This directly entails

$$F_\alpha^i(n) \geq n+i. \qquad (2.1.a')$$

2.1.b. We actually prove $F_\alpha(n+i) \geq F_\alpha(n)$ for all $i \in \mathbb{N}$:

If $\alpha = 0$, we are done with $n+i+1 \geq n+1$.

If $\alpha = \alpha'+1$ is a successor ordinal, then $F_\alpha(n+i) = F_{\alpha'}^{n+i+1}(n+i)$ (by D2) $\geq F_{\alpha'}^{n+1}(n+i)$ (by 2.1.a) $\geq F_{\alpha'}^{n+1}(n)$ (by ind. hyp.) $= F_\alpha(n)$.

If $\alpha \in \mathit{Lim}$, we rely on $\alpha_n \sqsubseteq \alpha_{n+i}$: $F_\alpha(n+i) = F_{\alpha_{n+i}}(n+i)$ (by D3) $\geq F_{\alpha_{n+i}}(n)$ (by ind. hyp.) $\geq F_{\alpha_n}(n)$ (by 2.2.a and ind. hyp.) $= F_\alpha(n)$.

2.1.c. $F_\alpha(n) > |\alpha|$ if $n > 0$:

If $\alpha = 0$, then $F_\alpha(n) = n+1 > 0 = |\alpha|$.

If $\alpha = \alpha'+1$ is a successor ordinal, then $F_\alpha(n) = F_{\alpha'}^{n+1}(n)$ (by D2) $> |\alpha'|+n$ (by ind. hyp. and using 2.1.a') $\geq |\alpha|$ since $|\alpha| = |\alpha'|+1$ and $n > 0$.

If $\alpha \in \mathit{Lim}$, we rely on $F_\alpha(n) = F_{\alpha_n}(n) > |\alpha_n|$ (by ind. hyp.) $= |\alpha| - 1 + n \geq |\alpha|$ since $n > 0$.

2.2.a. $F_\beta(n) \leq F_\alpha(n)$ if $\beta \sqsubseteq \alpha$:

If $\alpha = 0$, then necessarily $\beta = \alpha$ and we are done.

If $\alpha = \alpha'+1$ is a successor ordinal, we consider two cases. If $\beta = \beta'+1$ is a successor, then $\beta' \sqsubseteq \alpha'$ so that $F_{\beta'}(n) \leq F_{\alpha'}(n)$ by ind. hyp. Now, using 2.1.b we deduce $F_{\beta'}^{n+1}(n) \leq F_{\alpha'}^{n+1}(n)$, i.e., $F_\beta(n) \leq F_\alpha(n)$ as required. If β is a limit, then $\beta \sqsubseteq \alpha'$ and $F_\beta(n) \leq F_{\alpha'}(n)$ (by ind. hyp) $\leq F_{\alpha'}^{n+1}(n)$ (by 2.1.a) $= F_\alpha(n)$.

If $\alpha \in \mathit{Lim}$, then $\beta \in \mathit{Lim}$ too and there are two cases: either $\beta \sqsubseteq \alpha_n$ or $\beta_n \sqsubseteq \alpha_n$. In both cases the induction hypothesis concludes immediately.

Proof of (2.2.b)**.** Recall that, for any $p \in \mathbb{N}$, an ordinal α can be decomposed in a unique way under the form $\alpha = \alpha_1.\omega^p + \alpha_2$ such that $\alpha_2 < \omega^p$. This decomposition satisfies both $\alpha_1.\omega^p \sqsubseteq \alpha$ and $\alpha_2 \sqsubseteq \alpha$. Also note that $\alpha + \omega^p = \alpha_1.\omega^p + \omega^p = (\alpha_1+1).\omega^p$.

2.2.b. $F_{\gamma+\alpha}(n) \leq F_{\gamma+\omega^p+\alpha}(n)$ if $n > |\gamma|$: The proof is by induction over α. There are three cases.

1. $\alpha = 0$**:** we must prove that $F_\gamma(n) \leq F_{\gamma+\omega^p}(n)$. When $p = 0$, i.e., $\omega^p = 1$, we note that $\gamma \sqsubseteq \gamma + \omega^p$ so that (2.2.a) concludes. When $p > 0$, $\gamma + \omega^p \in \mathit{Lim}$. Decomposing γ as $\gamma_1.\omega^p + \gamma_2$ we obtain

$$\begin{aligned}
F_{\gamma+\omega^p}(n) &= F_{(\gamma_1+1).\omega^p}(n) \\
&= F_{\gamma_1.\omega^p+\omega^{p-1}.n}(n) \quad \text{(by D3)} \\
&= F_{\gamma_1.\omega^p+\omega^{p-1}.(n-1)+\omega^{p-2}.n}(n) \quad \text{(D3 again)} \\
&= F_{\gamma_1.\omega^p+\omega^{p-1}.(n-1)+\omega^{p-2}.(n-1)+\omega^{p-3}.n}(n) \\
&\cdots \\
&= F_{\gamma_1.\omega^p+[\sum_{i<p}\omega^i.(n-1)]+1}(n) \text{ (written } F_{\gamma_1.\omega^p+\gamma'}(n)).
\end{aligned}$$

Now $\gamma_2 \sqsubseteq \gamma'$ since $n > |\gamma|$. Hence $\gamma \sqsubseteq \gamma_1.\omega^p + \gamma'$ and (2.2.a) concludes.

2. $\alpha = \alpha'+1$**:** then $F_{\gamma+\alpha'}(n) \leq F_{\gamma+\omega^p+\alpha'}(n)$ by ind. hyp. One deduces that $F_{\gamma+\alpha'}^k(n) \leq F_{\gamma+\omega^p+\alpha'}^k(n)$ for all $k \in \mathbb{N}$ using 2.1.b (and also 2.1.a to guarantee that all arguments are $> |\gamma|$). Putting $k = n+1$, one obtains $F_{\gamma+\alpha}(n) \leq F_{\gamma+\omega^p+\alpha}(n)$ as required.

3. $\alpha \in \mathit{Lim}$**:** Let $d = \mathit{deg}(\alpha)$. If $d = p$ then $\gamma+\alpha \sqsubseteq \gamma+\omega^p+\alpha$ so that (2.2.a) concludes. If $d > p$, then $\gamma+\alpha = \gamma+\omega^p+\alpha$ which is even more direct.

If now $d < p$ then $(\omega^p+\alpha)_n$ is $\omega^p+\alpha_n$. Decompose γ both as $\gamma_1.\omega^p+\gamma_2$ and as $\gamma'_1.\omega^d+\gamma'_2$. Note that $\gamma_1+\omega^p = \gamma'_1+\omega^p$ since $d < p$. Finally

$$\begin{aligned}
F_{\gamma+\omega^p+\alpha}(n) &= F_{(\gamma+\omega^p+\alpha)_n}(n) \quad \text{by D3} \\
&= F_{\gamma_1+\omega^p+\alpha_n}(n) \\
&= F_{\gamma'_1+\omega^p+\alpha_n}(n) \\
&\leq F_{\gamma'_1+\alpha_n}(n) \quad \text{by ind. hyp., noting that } |\gamma'_1| \leq |\gamma| \\
&= F_{(\gamma+\alpha)_n}(n) \\
&= F_{\gamma+\alpha}(n) \quad \text{by D3.}
\end{aligned}$$

3 Stacking ordinals

We use stacks to define a small-steps semantics for the F_α's that will be easier to simulate in channel systems.

Definition 3.1 *A* stack *(of length $k \in \mathbb{N}$) is a finite sequence $\pi = \alpha_1, \alpha_2, \ldots, \alpha_k$ of increasing ordinals $< \Omega$, i.e., $\alpha_1 \leq \alpha_2 \leq \cdots \leq \alpha_k < \Omega$.*

Since a stack must list its elements in increasing order, there is a natural bijection between stacks and finite multisets over Ω . Hence we let $\mathcal{M}_f(\Omega)$ denote the set of stacks, and write $\pi <_{\text{ms}} \pi'$ when π is strictly smaller than π' in the multiset ordering inherited from the ordering of ordinals below Ω. This is a well-founded linear ordering with ε (the empty stack) as minimal element.

We now extend the $(F_\alpha)_\alpha$ family with fast-growing functions indexed by stacks, denoted $F_\pi : \mathbb{N} \to \mathbb{N}$, and defined with:

$$F_\varepsilon(n) \stackrel{\text{def}}{=} n, \qquad F_{\alpha,\pi}(n) \stackrel{\text{def}}{=} F_\pi(F_\alpha(n)).$$

Note that F_α is the same when we see α as an ordinal or as a stack of length one, hence we will not disambiguate.

The evaluation of some $F_\pi(n)$ can be expressed as a transformation system, where the manipulated objects are pairs $\langle\langle \pi\,;\, n\rangle\rangle$ of a stack π and a natural number n. Formally, we define a relation over $\mathcal{M}_f(\Omega)\times\mathbb{N}$, denoted $\to_R$, and defined by the three following "rewrite" rules:

$$\langle\langle 0,\pi\,;\, n\rangle\rangle \;\to_R\; \langle\langle \pi\,;\, n+1\rangle\rangle \tag{R1}$$

$$\langle\langle \alpha+1,\pi\,;\, n\rangle\rangle \;\to_R\; \langle\langle \overbrace{\alpha,\alpha,\ldots,\alpha}^{n+1\text{ times}},\pi\,;\, n\rangle\rangle \tag{R2}$$

$$\langle\langle \lambda,\pi\,;\, n\rangle\rangle \;\to_R\; \langle\langle \lambda_n,\pi\,;\, n\rangle\rangle \qquad \text{if } \lambda\in \mathit{Lim}. \tag{R3}$$

Observe that if π is a stack and $\langle\langle \pi\,;\, n\rangle\rangle \to_R \langle\langle \pi'\,;\, n'\rangle\rangle$ then π' is indeed a stack, $\pi' <_{\text{ms}} \pi$ and $n'\geq n$. Note that $\to_R$ is deterministic.

Corollary 3.2 *$\to_R$ is terminating and convergent.*

The normal forms are the pairs $\langle\langle \pi\,;\, n\rangle\rangle$ with $\pi=\varepsilon$.

Since rules R1–3 merely reformulate definitions D1–3 in terms of stacks, it follows that $\langle\langle \pi\,;\, n\rangle\rangle \to_R \langle\langle \pi'\,;\, n'\rangle\rangle$ implies $F_\pi(n) = F_{\pi'}(n')$. With Cor. 3.2, one deduces $\langle\langle \pi\,;\, n\rangle\rangle \to_R^* \langle\langle \varepsilon\,;\, F_\pi(n)\rangle\rangle$.

Write $\leftrightarrow_R$ for $\to_R \cup \to_R^{-1}$. The previous observations entail

Lemma 3.3 *$\langle\langle \pi\,;\, n\rangle\rangle \leftrightarrow_R^* \langle\langle \pi'\,;\, n'\rangle\rangle$ iff $F_\pi(n) = F_{\pi'}(n')$.*

Notation 3.4 *When dealing with $\leftrightarrow_R$, it is convenient to decompose it as the union $\to_{R1}\cup\to_{R2}\cup\to_{R3}\cup\to_{S1}\cup\to_{S2}\cup\to_{S3}$ of the six relations defined by rules R1 to R3 and by* inverse rules *denoted S1 to S3, and defined such that $\to_{Si} = \to_{Ri}^{-1}$. (See Appendix B for the explicit definition.)*

4 A differential encoding of stacks

For $K\in\mathbb{N}$, we let $\Sigma_K \stackrel{\text{def}}{=} \{\omega^0,\omega^1,\omega^2,\ldots,\omega^{K-1}\}\cup\{\mathsf{I}\}$ be an alphabet with $K+1$ symbols, that we use to encode stacks (restricted to ordinals $<\omega^K$). The symbols "ω^p" denote the corresponding finite powers of the ordinal ω. In particular, "ω^0" and "ω^1" denote, respectively, the ordinals 1 and ω.

We first explain the encoding informally. Consider the following word $u\in\Sigma_K^*$:

$$u \;=\; \omega^0\omega^0\mathsf{I}\omega^3\omega^1\mathsf{I}\mathsf{I}\omega^1\omega^0\mathsf{I}.$$

One reads u from left to right. While reading u, all the encountered ordinal symbols are added up, giving rise to a notion of current sum, or height. A tally symbol "I" codes for an ordinal in the stack: *each* I *stands for one copy of the current sum*. In our example, the stack associated with u, is

$$\Pi(u) \;=\; 2,\ \omega^3+\omega,\ \omega^3+\omega,\ \omega^3+\omega.2+1.$$

(Indeed $\omega^0+\omega^0=2$ and $\omega^0+\omega^0+\omega^3+\omega^1=\omega^3+\omega$. Furthermore, $\Pi(u)$ contains two occurrences of $\omega^3+\omega$ because u contains two tally symbols immediately after the first occurrence of ω^1.)

Formally, the correspondence $\Pi:\Sigma_K^*\to\mathcal{M}_f(\Omega)$ and the height function $h:\Sigma_K^*\to\Omega$ are defined by induction over u:

$$h(\varepsilon)\stackrel{\text{def}}{=}0;\quad h(u\mathsf{I})\stackrel{\text{def}}{=}h(u);\qquad h(u\omega^i)\stackrel{\text{def}}{=}h(u)+\omega^i;$$
$$\Pi(\varepsilon)\stackrel{\text{def}}{=}\varepsilon;\quad \Pi(u\mathsf{I})\stackrel{\text{def}}{=}\Pi(u),h(u);\quad \Pi(u\omega^i)\stackrel{\text{def}}{=}\Pi(u).$$

Observe that $\Pi(u)$ is indeed a stack, i.e., $\Pi(u)$ lists increasing ordinals, since $h(u.v)\geq h(u)$ for all u,v.

Remark 4.1 *We call this encoding* differential *since the ω^p symbols in Σ_K are not used to directly represent an α_j in a stack $\pi=\alpha_1,\ldots,\alpha_k$. Rather they represent the "difference" $\alpha_j-\alpha_{j-1}$ that must be added to the previous ordinal in order to obtain α_j.*

Any $u\in\Sigma_K^*$ encodes a stack, and any stack below ω^K can be encoded with some $u\in\Sigma_K^*$. Such an encoding is not unique. However, there is a unique shortest one, called a *pure* encoding.

Definition 4.2 (Pure encodings) *An encoding $u\in\Sigma_K^*$ is* pure *if (1) it does not end with an ω^i symbol, and (2) it does not contain a factor of the form $\omega^i\omega^j$ with $i<j$.*

Note that the pure encodings are a regular subset of Σ_K^*.

The idea behind purity is to forbid useless symbols in an encoding. If u is not pure, this is witnessed by some occurrence of some ω^i. Removing that occurrence yields some shorter u' with $\Pi(u')=\Pi(u)$. Hence any impure u can be replaced by a shorter equivalent encoding. Reciprocally, if u is pure and u' is shorter than u, then $\Pi(u')\neq\Pi(u)$.

Purity allows transferring the monotonicity lemmas from stacks to their encodings. Write $u\sqsubseteq v$ when u is a (scattered) subword of v, i.e., u can be obtained from v by removing some letters (possibly none). The rest of this section proves the following proposition.

Proposition 4.3 *Let $u,v\in\Sigma_K^*$ and $n>0$. If $u\sqsubseteq v$ and v is pure, then $F_{\Pi(u)}(n)\leq F_{\Pi(v)}(n)$.*

The crux of the proof is the case where u and v only differ by one ordinal symbol:

Lemma 4.4 *$F_{\Pi(v_1v_2)}(n)\leq F_{\Pi(v_1\omega^pv_2)}(n)$ when $v_1\omega^pv_2$ is pure and $n>0$.*

Proof. Write $\pi = \alpha_1, \ldots, \alpha_k$ for $\Pi(v_1\omega^p v_2)$ and $\pi' = \alpha'_1, \ldots, \alpha'_k$ for $\Pi(v_1 v_2)$ (clearly, π and π' have same length). Write $l \in \{0, \ldots, k-1\}$ for the length of $\Pi(v_1)$. Then $\alpha'_i = \alpha_i$ for $i = 1, \ldots, l$ and, for $i = l+1, \ldots, k$, we can write α_i and α'_i under the following form:

$$\alpha_i = h(v_1) + \omega^p + \beta_i, \qquad \alpha'_i = h(v_1) + \beta_i,$$

where $\beta_{l+1}, \ldots, \beta_k$ is simply $\Pi(v_2)$. There are now two cases:
(1) If v_1 ends with some "l" symbol (or $v_1 = \varepsilon$), then $h(v_1) = \alpha_{l-1}$, putting $\alpha_0 = 0$ by convention. Observe that $F_{\alpha'_1,\ldots,\alpha'_l}(n) > |\alpha_l|$ as a consequence of (2.1.c) and (2.1.a). Thus (2.2.b) applies and we can prove that $F_{\alpha'_1,\ldots,\alpha'_i}(n) \leq F_{\alpha_1,\ldots,\alpha_i}(n)$ for all $i = l+1, \ldots, k$ by induction over i.
(2) Otherwise v_1 ends with some ω^r symbol. Observe that $r \geq p$ since $v_1\omega^p v_2$ is pure. This implies that $\alpha'_i \sqsubseteq \alpha_i$ for $i \geq l$ (and hence for all i's). We conclude with (2.2.a) and the other monotonicity properties. □

The case where u and v differ by one tally symbol is easier.

Lemma 4.5 $F_{\Pi(v_1 v_2)}(n) \leq F_{\Pi(v_1 \mathrm{l} v_2)}(n)$.

Proof. [Sketch] $\Pi(v_1 v_2)$ is obtained by removing one ordinal somewhere in $\Pi(v_1 \mathrm{l} v_2)$. Hence we can conclude with (2.1.a) and the other monotonicity properties. □

There remains to deal with the case where u and v differ by more than one symbol. Write $u \sqsubseteq_k v$ when $u \sqsubseteq v$ and $|v| = |u| + k$. Write $u \equiv_\Pi v$ when $\Pi(u) = \Pi(v)$.

Lemma 4.6 *If $u \sqsubseteq v$ and v is pure then there is a sequence*

$$u \equiv_\Pi u_1 \sqsubseteq_1 u_2 \sqsubseteq_1 \cdots \sqsubseteq_1 u_n = v$$

where all u_i's, $i = 1, \ldots, n$, are pure.

Proof. We let u_1 be the pure encoding of $\Pi(u)$: this is a subword of u, hence of v too. The sequence $u_1 \sqsubseteq_1 u_2 \sqsubseteq_1 \cdots \sqsubseteq_1 u_n$ is obtained by inserting in u_1, one by one, all the (occurrences of) symbols that are in v but missing in u_1. One first inserts all the missing tally symbols (in no particular order) and then, in a second phase, all the missing ordinal symbols (in no particular order). This ensures that all the u_i's are pure: In the first phase, a u_i inherits purity from u_{i-1}, starting with u_1, since $x\mathrm{l}y$ is pure when xy is. In the second phase, a u_i inherits purity from u_{i+1}, starting from $u_n = v$, since xy is pure when $x\omega^j y$ is. □

With Lemma 4.6 one can reduce Prop. 4.3 to repetitive applications of Lemmas 4.4 and 4.5, which concludes the proof of Proposition 4.3.

5 Fast-growing functions via lossy channels

5.1 Lossy channel systems

This section summarily defines *lossy channel systems* (LCS) and their behaviour. We refer the reader to [4, 8, 26] for more details.

A LCS is a tuple $S = (\mathtt{Q}, \mathtt{M}, \mathtt{C}, \Delta)$ where $\mathtt{Q} = \{q_1, q_2, \ldots\}$ is a finite set of *(control) locations*, $\mathtt{M} = \{a_1, a_2, \ldots, a_k\}$ is a finite *message alphabet*, $\mathtt{C} = \{\mathtt{c}_1, \mathtt{c}_2, \ldots, \mathtt{c}_l\}$ is a finite set of *channels*, and $\Delta \subseteq \mathtt{Q} \times \mathtt{C} \times \{!, ?\} \times \mathtt{M} \times \mathtt{Q}$ is a finite set of *transition rules*, with typical elements denoted δ. A rule of the form $(q, \mathtt{c}, !, a, q')$ (respectively, $(q, \mathtt{c}, ?, a, q')$) is called a *writing rule* (resp., a *reading rule*).

Assume that $S = (\mathtt{Q}, \mathtt{M}, \mathtt{C}, \Delta)$ is a LCS with l channels. A *configuration* of S is a pair $(q, \boldsymbol{u})$, where $q \in \mathtt{Q}$ is the current location and $\boldsymbol{u} \in (\mathtt{M}^*)^l$, is the contents of the channels. $(q, \boldsymbol{u})$ is sometimes written $(q, u_1, \ldots, u_l)$ where $u_i \in \mathtt{M}^*$ is the sequence of messages contained in channel $\mathtt{c}_i$ (by convention, reading occurs at the head of u_i and writing at its tail). We write $Conf = \{\sigma, \rho, \ldots\}$ for the set $\mathtt{Q} \times (\mathtt{M}^*)^l$ of configurations (of S).

Configurations are compared via the subword ordering:

$$(q, u_1, \ldots, u_l) \sqsubseteq (q', u'_1, \ldots, u'_l) \stackrel{\text{def}}{\Leftrightarrow} q = q' \wedge \bigwedge_{i=1,\ldots,l} u_i \sqsubseteq u'_i.$$

Observe that, since $\mathtt{Q}$ and $\mathtt{M}$ are finite, $(Conf, \sqsubseteq)$ is a well-partial-order as a consequence of Higman's Lemma.

The operational semantics of S is given under the form of a transition system $\mathcal{T}_S = (Conf, \to)$. Assume that $\sigma = (q, u_1, \ldots, u_l)$ and $\sigma' = (q', u'_1, \ldots, u'_l)$ are two configurations. There is a (lossy) step from σ to σ' via rule δ, denoted $\sigma \xrightarrow{\delta} \sigma'$, when
— **case 1**: $\delta \in \Delta$ is a reading rule of the form $(q, \mathtt{c}_i, ?, a, q')$ and $u_i = a u'_i$ while $u_j = u'_j$ for $j \neq i$, or
— **case 2**: δ is a writing rule $(q, \mathtt{c}_i, !, a, q')$ and $u'_i = u_i a$ while $u_j = u'_j$ for $j \neq i$, or
— **case 3**: δ is a writing rule $(q, \mathtt{c}_i, !, a, q')$ and $u_j = u'_j$ for all $j = 1, ..., l$.
Hence a message can be lost (case 3) during a step that attempts to write it in the channels. Once in the channels, messages cannot be lost, they can only be removed by reading steps.

A *perfect step*, written $\sigma \xrightarrow{\delta}_{\text{perf}} \sigma'$, is a step that is derived from case 1 or 2 only. Perfect steps are the expected behaviour of finite-state channel systems [7]. A (lossy) *run* is a sequence of consecutive steps. *Perfect runs*, that only use perfect steps, are a special case of lossy runs.

When writing steps, we usually omit the δ superscript when it is not useful. We use the standard notations "$\xrightarrow{n}$", "$\xrightarrow{+}$" and "$\xrightarrow{*}$" for, respectively, the n-fold composition, the

transitive closure and the reflexive-transitive closure of a transition relation "$\rightarrow$".

Remark 5.1 (Different ways to lose messages) *The literature proposes several different notions of message losses in LCS's. The definition we used above is called "write-lossy" and is the most convenient for the technical developments that follow. This choice has no important impact on the behaviour of lossy systems, and no impact at all on our main results that apply equivalently to other standard notions of lossy steps (see Appendix A).*

5.2 A channel system that computes fast-growing functions

We now construct a LCS, called W_K, that weakly computes the F_α functions for all $\alpha < \omega^K$. It can also weakly compute their inverses F_α^{-1} as we explain later.

W_K uses two channels. The first channel, p, stores a word $u \in \Sigma_K^*$ that encodes a stack of ordinals as in Section 4. The second channel, d, stores a number $n > 0$ in unary (using n times the tally symbol, or I^n). Thus a pair $\langle\langle \pi ; n \rangle\rangle$ is stored in two channels. An extra marker symbol # is written at the end of these encodings to recognize their extremity during the manipulations.

The overall structure of W_K is illustrated in Fig. 1 (see Appendix B for the details of the components). When explaining its behaviour, we call "*single-pass* run" any run that does not visit the state loop. In state beg, W_K

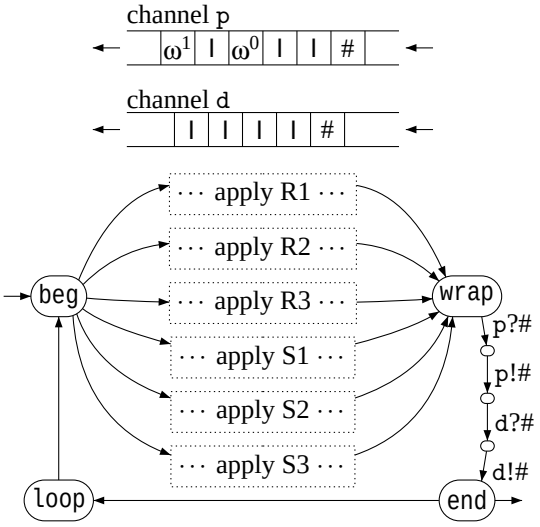


Figure 1. A schematic view of W_K.

will traverse one of six possible "components" where it transforms the pair $\langle\langle \pi ; n \rangle\rangle$ (more precisely, its encoding) stored in the channels by one application of the rewriting rules R1 to R3 (from section 2), or the inverse rules S1 to S3. With our encodings of pairs, each of these rules can be seen as a finite-state transduction. The LCS's that implement these components are described in Appendix B. Implementing one rewriting step, W_K will replace $\langle\langle \pi ; n \rangle\rangle$ with the resulting $\langle\langle \pi' ; n' \rangle\rangle$, that is, unless message losses corrupt the result. Then W_K reaches state wrap where it reads the end markers and writes them back after $\langle\langle \pi' ; n' \rangle\rangle$. In state end W_K can terminate and exit, or loop back to beg and transform $\langle\langle \pi' ; n' \rangle\rangle$ again, therefore computing the transitive closure of $\hookrightarrow_R$.

The construction ensures the following features:

sanity check: The rule components assume that each channel contain a Σ_K-word followed by at most one marker symbol #. With this assumption, the components check that the channels contain proper inputs. Formally, there is a single-pass run from $(\mathsf{beg}, u\#, v\#)$ to state end only if u is some pure encoding, and v is some I^n for some $n > 0$. If this is not the case, on impure u or incorrect v, W_K will stop in a deadlock. If a final # is missing, W_K will loop without reaching end.

one-pass transduction: If the channels contain proper inputs, a single-pass run from $(\mathsf{beg}, u\#, v\#)$ to some (wrap, w, w') reads u and v completely, write some new data u' and v', and does not touch the end markers. Hence $w = \#u'$ and $w' = \#v'$.

rule applicability: When going from beg to end, W_K chooses nondeterministically what rule component will be traversed. It may be the case that the corresponding rule is not applicable to the current channel contents: this is checked by W_K and it will stop in a deadlock if the rule is not applicable.

We can now state formally how W_K implements $\hookrightarrow_R$.

Lemma 5.2 (Single-pass perfect runs in W_K) *Assume that $u, u' \in \Sigma_K^*$ are the pure encodings of two stacks π and π'. Assume $n, n' > 0$. Then $\langle\langle \pi ; n \rangle\rangle \hookrightarrow_R \langle\langle \pi' ; n' \rangle\rangle$ if, and only if, W_K has a single-pass* perfect *run of the form*

$$(\mathsf{beg}, u\#, \mathsf{I}^n\#) \xrightarrow{*}_{\text{perf}} (\mathsf{end}, u'\#, \mathsf{I}^{n'}\#).$$

Proof. **[Idea]** The "$\Rightarrow$" direction is obvious since W_K implements exactly the six rules that define $\hookrightarrow_R$ (see Appendix B). Reciprocally, the **rule-applicability** features ensure that end is only reached by one proper step of rewriting. Hence the "$\Leftarrow$" direction. □

The corollary is:

Theorem 5.3 (W_K weakly computes the F_α's) *Assume that $u, u' \in \Sigma_K^*$ are the pure encodings of two stacks π and*

π'. *Assume* $n, n' > 0$. *Then* $F_\pi(n) \geq F_{\pi'}(n')$ *if, and only if,* W_K *has a* lossy *run of the form*

$$(\mathsf{beg}, u\#, \mathsf{l}^n\#) \xrightarrow{*} (\mathsf{end}, u'\#, \mathsf{l}^{n'}\#).$$

Proof. ($\Rightarrow$): Write a for $F_\pi(n)$ and b for $F_{\pi'}(n')$. By Lemma 3.3, there exist rewriting sequences of the form $\langle\langle \pi;\, n\rangle\rangle \leftrightarrow_R^* \langle\langle \varepsilon;\, a\rangle\rangle$ and $\langle\langle \varepsilon;\, b\rangle\rangle \leftrightarrow_R^* \langle\langle \pi';\, n'\rangle\rangle$, and it is even possible to ensure $\langle\langle \pi;\, n\rangle\rangle \leftrightarrow_R^+ \langle\langle \varepsilon;\, a\rangle\rangle$ by inserting extra rewriting steps. These rewriting steps entail the existence of corresponding single-pass perfect runs (Lemma 5.2). Concatenating these, we deduce that W_K has two perfect runs of the form $(\mathsf{beg}, u\#, \mathsf{l}^n\#) \xrightarrow{*}_{\text{perf}} (\mathsf{end}, \#, \mathsf{l}^a\#)$ and $(\mathsf{beg}, \#, \mathsf{l}^b\#) \xrightarrow{*}_{\text{perf}} (\mathsf{end}, u'\#, \mathsf{l}^{n'}\#)$. Since $a \geq b$, there also exists a lossy run $(\mathsf{beg}, u\#, \mathsf{l}^n\#) \xrightarrow{*} (\mathsf{end}, \#, \mathsf{l}^b\#)$ obtained by losing $a-b$ tally symbols in $\mathtt{d}$ during the last single-pass of the first run. Concatenating with the second run we obtain the required lossy run $(\mathsf{beg}, u\#, \mathsf{l}^n\#) \xrightarrow{*}_{\text{perf}} (\mathsf{end}, u'\#, \mathsf{l}^{n'}\#)$.

($\Leftarrow$): Write k for the number of times the run $(\mathsf{end}, u\#, \mathsf{l}^n\#) \xrightarrow{*} (\mathsf{end}, u'\#, \mathsf{l}^{n'}\#)$ visits state $\mathtt{loop}$. We prove the implication by induction over k. If $k = 0$, then the run has length zero, $u = u'$, $n = n'$ and we are done. Now assume $k > 0$. The run has the form

$$(\mathsf{end}, u\#, \mathsf{l}^n\#) \to (\mathtt{loop}, u\#, \mathsf{l}^n\#) \to \overbrace{(\mathsf{beg}, u\#, \mathsf{l}^n\#) \xrightarrow{*} (\mathsf{end}, w, w')}^{\text{single-pass}} \underbrace{\xrightarrow{*} (\mathsf{end}, u'\#, \mathsf{l}^{n'}\#)}_{k-1 \text{ remaining visits}}.$$

After two steps, the first single-pass reaches (end, w, w') by traversing one of the six components of W_K. Traversing the same component, W_K has a perfect single-pass run $(\mathsf{beg}, u\#, \mathsf{l}^n\#) \xrightarrow{*} (\mathsf{end}, v\#, \mathsf{l}^m\#)$ satisfying

$$F_{\Pi(u)}(n) = F_{\Pi(v)}(m) \tag{1}$$

thanks to Lemma 5.2. With our write-lossy semantics, the **one-pass transduction** features ensure that w and w' are subwords of, respectively, $v\#$ and $\mathsf{l}^m\#$. Observe that w and w' are proper inputs, i.e., w is some $v'\#$ for some pure v, and w' is some $\mathsf{l}^{m'}\#$ for some $m' > 0$. Indeed, either $k > 1$ and the **sanity check** features require a proper input (otherwise the next single-pass would not succeed), or $k = 1$, implying that $w = u'\#$ and $w' = \mathsf{l}^{n'}\#$. Therefore, the induction hypothesis applies, yielding

$$F_{\Pi(v')}(m') \geq F_{\Pi(u')}(n'). \tag{2}$$

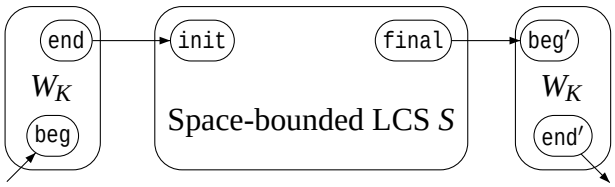
Now, since $v'\# \sqsubseteq v\#$ and $\mathsf{l}^{m'} \sqsubseteq \mathsf{l}^m\#$, i.e., $v' \sqsubseteq v$ and $m' \leq m$, since v is pure and $m' > 0$, Lemmas 2.1.b and 4.3 imply

$$F_{\Pi(v')}(m') \leq F_{\Pi(v)}(m). \tag{3}$$

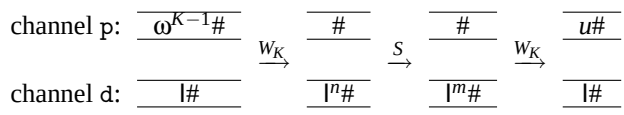
Combining (1–3) provides the required $F_{\Pi(u)}(n) \geq F_{\Pi(u')}(n')$. □

5.3 Lower bounds for LCS's

W_K can be used to check that a possibly lossy run is actually perfect in space-bounded LCS's. Formally, a space-bounded LCS is a LCS operating on one channel and whose transition rules write exactly as many messages as they read (see [26]). Hence the number of messages in the channel remains constant during perfect runs, and it can only decrease during lossy runs. Given a space-bounded S, and



some $K \in \mathbb{N}$, we build the LCS S_K by inserting two copies of W_K, one before and one after S, as schematically depicted above. S does not use $\mathtt{p}$, only $\mathtt{d}$. The idea is that the first W_K will be started with a pair $\langle\langle \omega^{K-1};\, 1\rangle\rangle$ in the channels, will write some large $\mathsf{l}^n\#$ in $\mathtt{d}$, that will be used by S, that will return $\mathsf{l}^m\#$ to be fed to the second W_K:



The construction of S_K has some simple sanity checks (not depicted) between the W_K's and the S part, ensuring that the # markers are not lost, etc.

Now, assume S_K has a run of the form

$$\begin{aligned}(\mathsf{beg}, \omega^{K-1}\#, \mathsf{l}\#) &\xrightarrow{*} (\mathsf{end}, u\#, \mathsf{l}^n\#) \\ &\to (\mathtt{init}, u\#, \mathsf{l}^n\#) \xrightarrow{*} (\mathtt{final}, u\#, \mathsf{l}^m\#) \quad (\dagger) \\ &\to (\mathsf{beg}', u\#, \mathsf{l}^m\#) \xrightarrow{*} (\mathsf{end}', \omega^{K-1}\#, \mathsf{l}\#)\end{aligned}$$

Then the construction of W_K ensures that $n \leq F_{\omega^{K-1}}(1)$ and $F_{\omega^{K-1}}(1) \leq m$ (by Theorem 5.3). Since S is space-bounded, $n \geq m$. Hence a run like (†) requires $n = m$ $(= F_{\omega^{K-1}}(1))$, so that the sub-run $(\mathtt{init}, u\#, \mathsf{l}^n\#) \xrightarrow{*} (\mathtt{final}, u\#, \mathsf{l}^m\#)$ must be perfect. Reciprocally, a run $(\mathsf{beg}, \omega^{K-1}\#, \mathsf{l}\#) \xrightarrow{*} (\mathsf{end}', \omega^{K-1}\#, \mathsf{l}\#)$ in S'_K must be decomposable under the form of (†).

Corollary 5.4 S_K *has a run from* $(\mathsf{beg}, \omega^{K-1}\#, \mathsf{l}\#)$ *to* $(\mathsf{end}', \omega^{K-1}\#, \mathsf{l}\#)$ *if, and only if,* S *has an accepting* perfect *run using space* $F_{\omega^{K-1}}(1)$.

Theorem 5.5 (Main result) *Reachability for lossy channel systems does not belong to* $\mathfrak{F}_{<\omega^\omega}$.

Proof. Using S_K, it is possible to reduce the problem of whether a space-bounded LCS S has an accepting

perfect run using space $\leq F_{\omega^{K-1}}(1)$ to a LCS-reachability question of size polynomial in K and $|S|$. We conclude by observing that perfect space-bounded LCS's have the same computational power than space-bounded Turing machines, and that the hierarchy $(\mathfrak{F}_{<\omega^K})_{K=1,2,3,\ldots}$ is strict. □

There exists a similar construction, again using W_K, that reduces the existence of perfect space-bounded runs to *termination* of LCS's, rather than *reachability* (along the lines of [26, section 4.2]). The consequences are similar:

Theorem 5.6 *Termination for lossy channel systems does not belong to* $\mathfrak{F}_{<\omega^\omega}$.

6 Upper bounds

In this section, we explain how Cichon's and Tahhan Bittar's analysis of Higman's Lemma [12] leads to:

Observation 6.1 *Reachability and termination for lossy channel systems are in* $\mathfrak{F}_{\omega^\omega}$.

Since we showed that these problems do not belong to $\mathfrak{F}_{<\omega^\omega}$, this concludes the proof of our main result.

Longest bad sequences. Let Σ be an alphabet containing p letters. Two l-tuples $\boldsymbol{u},\boldsymbol{u}'$ of words from Σ^* can be compared using the subword ordering component-wise, i.e., defining

$$\boldsymbol{u}=(u_1,\ldots,u_l)\sqsubseteq(u'_1,\ldots,u'_l)=\boldsymbol{u}' \stackrel{\text{def}}{\Leftrightarrow} u_1\sqsubseteq u'_1\wedge\cdots\wedge u_l\sqsubseteq u'_l.$$

A sequence $\boldsymbol{u}_0,\boldsymbol{u}_1,\boldsymbol{u}_2,\ldots$, is *bad* if there are no $i<j$ such that $\boldsymbol{u}_i\sqsubseteq\boldsymbol{u}_j$, it is *good* otherwise. For $r\in\mathbb{N}$, we say the sequence is r-good if it contains an increasing subsequence of length $r+2$, i.e., if $\boldsymbol{u}_{i_1}\sqsubseteq\boldsymbol{u}_{i_2}\sqsubseteq\cdots\sqsubseteq\boldsymbol{u}_{i_{r+2}}$ for some $i_1<i_2<\cdots<i_{r+2}$ (and it is r-bad otherwise). By Higman's Lemma, $\sqsubseteq$ is a well-partial-order on $(\Sigma^*)^l$, hence any bad sequence, and any r-bad sequence, is finite. Arbitrarily long bad sequences can be produced, for example by starting with $\boldsymbol{u}_0$ large enough. However, if one considers *controlled* sequences, i.e., sequences such that $|\boldsymbol{u}_i|\leq i+|\boldsymbol{u}_0|$ for all i, it is easy to see (Hint: use König's Lemma) that there is an upper bound on the lengths of controlled bad sequences that start from a given $\boldsymbol{u}_0$.

Cichon and Tahhan Bittar consider the function $H(p,l,r,n)$, defined as the length of the longest controlled r-good sequence with $|\boldsymbol{u}_0|=n$ for an alphabet Σ of size p [2]. They show that $H(p,l,r,n)\leq F_{\omega^{f(p)}}(\max(l,r,n))$ for some function f that is left implicit but that is definitely primitive-recursive [11] (see also [27]). Hence H belongs to $\mathfrak{F}_{\omega^\omega}$.

[2] $H(p,l,r,n)$ is our notation for what Cichon and Tahhan Bittar denote $Hig(\omega^p.l,r,Succ)(n)$.

Bounding termination and reachability. When configurations of a LCS are compared with $\sqsubseteq$, there are similar notions of a bad, and of an r-bad, run $\sigma_0\rightarrow\sigma_1\rightarrow\ldots\rightarrow\sigma_n$. With such a run, we associate its sequence of channel contents $\boldsymbol{u}_0,\ldots,\boldsymbol{u}_n$, obtained by forgetting the control state part of a configuration $\sigma_i=(q_i,\boldsymbol{u}_i)$. Observe that if the run is bad then the sequence $(\boldsymbol{u}_i)_{i=0,\ldots,n}$ is $(|\mathtt{Q}|-1)$-bad (by the pigeonhole principle). Hence bad runs have length bounded in $O\big(F_{\omega^\omega}(\max(|\mathtt{Q}|,|\mathtt{C}|,|\sigma_0|))\big)$, or even in $O\big(F_{\omega^{f(p)}}(\max(|\mathtt{Q}|,|\mathtt{C}|,|\sigma_0|))\big)$.

Now, since deciding termination can be done by checking that all runs from σ_0 are bad (this is the classic algorithm, see [16, 4, 17]), termination of LCS is in $\mathfrak{F}_{\omega^\omega}$.

Regarding reachability, the backward-chaining algorithm [4, 17] also builds a bad sequence of configurations: the minimal elements of $Pre^*(Goal)$ for some upward-closed $Goal\subseteq Conf$ defined by its minimal elements. By construction, this sequence is controlled (even though it is not a run *per se*). Hence the running time of the algorithm is in $\mathfrak{F}_{\omega^\omega}$ too.

We observe that these two algorithms handle equally well our write-lossy semantics or the standard lossy semantics (see Appendix A).

Variants and restrictions. From the above observations, one concludes that termination and reachability are in $\mathfrak{F}_{\omega^{f(p)}}$ if we restrict ourselves to LCS's having a message alphabet of size at most p. This indicates that the size of `M`, not the number of channels, or the number of control states, or the size of the initial configuration, is the key parameter affecting complexity. (Note that, in section 5, we used an alphabet of size $K+2$ to build LCS's whose complexity was not in $\mathfrak{F}_{<\omega^K}$.) Since the cumulative hierarchy $(\mathfrak{F}_\alpha)_{\alpha<\omega^\omega}$ is strict, we deduce that increasing the alphabet size of LCS's gives rise to a strict hierarchy of verification problems (more precisely, a hierarchy that contains a strict sub-hierarchy). This further explains why LCS's with large message alphabets cannot be simulated by LCS's with a fixed alphabet (more exactly, not via a reduction in $\mathfrak{F}_{<\omega^\omega}$) unlike the way Turing machines can be restricted to alphabets of size 2. Contrast this with the fact that LCS's with l channels can be simulated (via a many-one polynomial-time reduction) by LCS's with a single channel and an alphabet enlarged with a single extra symbol.

In the same spirit, let us observe that Lossy Counter Machines [23], which can be seen as LCS's where the alphabet has size 1, can be verified in $\mathfrak{F}_l$, where l is the number of counters. This is a direct consequence of McAloon's bounds on the length of bad sequences in $\mathbb{N}^l$ ordered by the component-wise ordering [24]. When l is not fixed, reacha-

bility and termination for these Lossy Counter Machines is exactly in $\mathfrak{F}_\omega$ [26].

7 Conclusion

Our main construction shows that lossy channel systems can compute, in a weak sense, the Fast-Growing Functions F_α of the extended Grzegorczyk hierarchy, for all $\alpha < \omega^\omega$. This construction can be used to show that reachability and termination for lossy channel systems cannot be in $\mathfrak{F}_{<\omega^\omega}$, i.e., cannot be multiply-recursive. We further explain how they belong to $\mathfrak{F}_{\omega^\omega}$ as a consequence of Cichon and Tahhan Bittar's analysis of Higman's Lemma. Hence we could precisely locate the complexity of these problems in the Fast-Growing Hierarchy.

Using known reductions, these results apply to other verification problems for LCS's: safety, inevitability, regular equivalences [20], game-theoretical properties [2, 5], probabilistic verification [1, 6], etc. They also apply to problems, like the Regular Post Embedding Problem, that have been equated to LCS reachability [9, 10]. The hardness proof also applies to problems to which LCS reachability reduces, like one-clock alternating time automata [21, 3] and many other problems.

References

[1] P. A. Abdulla, N. Bertrand, A. Rabinovich, and Ph. Schnoebelen. Verification of probabilistic systems with faulty communication. *Information and Computation*, 202(2):141–165, 2005.

[2] P. A. Abdulla, A. Bouajjani, and J. d'Orso. Monotonic and downward closed games. *Journal of Logic and Computation*, 18(1):153–169, 2008.

[3] 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.

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

[5] C. Baier, N. Bertrand, and Ph. Schnoebelen. On computing fixpoints in well-structured regular model checking, with applications to lossy channel systems. In *Proc. LPAR 2006*, volume 4246 of *Lecture Notes in Artificial Intelligence*, pages 347–361. Springer, 2006.

[6] C. Baier, N. Bertrand, and Ph. Schnoebelen. Verifying nondeterministic probabilistic channel systems against ω-regular linear-time properties. *ACM Transactions on Computational Logic*, 9(1), 2007.

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

[8] 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.

[9] P. Chambart and Ph. Schnoebelen. Post embedding problem is not primitive recursive, with applications to channel systems. In *Proc. FST&TCS 2007*, volume 4855 of *Lecture Notes in Computer Science*, pages 265–276. Springer, 2007.

[10] P. Chambart and Ph. Schnoebelen. The ω-regular Post embedding problem. In *Proc. FOSSACS 2008*, volume 4962 of *Lecture Notes in Computer Science*, pages 97–111. Springer, 2008.

[11] E. A. Cichon. Personal communication, Dec. 2007.

[12] E. A. Cichon and E. Tahhan Bittar. Ordinal recursive bounds for Higman's theorem. *Theoretical Computer Science*, 201(1–2):63–84, 1998.

[13] D. H. J. de Jongh and R. Parikh. Well-partial orderings and hierarchies. *Indag. Math.*, 39(3):195–207, 1977.

[14] S. Demri and R. Lazić. LTL with the freeze quantifier and register automata. In *Proc. LICS 2006*, pages 17–26. IEEE Comp. Soc. Press, 2006.

[15] M. V. Fairtlough and S. S. Wainer. Hierarchies of provably recursive functions. In S. Buss, editor, *Handbook of Proof Theory*, volume 137 of *Studies in Logic*, chapter 3, pages 149–207. North-Holland, 1998.

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

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

[18] 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.

[19] B. Konev, F. Wolter, and M. Zakharyaschev. Temporal logics over transitive states. In *Proc. CADE 2005*, volume 3632 of *Lecture Notes in Computer Science*, pages 182–203. Springer, 2005.

[20] A. Kučera and Ph. Schnoebelen. A general approach to comparing infinite-state systems with their finite-state specifications. *Theoretical Computer Science*, 358(2–3):315–333, 2006.

[21] S. Lasota and I. Walukiewicz. Alternating timed automata. In *Proc. FOSSACS 2005*, volume 3441 of *Lecture Notes in Computer Science*, pages 250–265. Springer, 2005.

[22] R. Lazić, T. Newcomb, J. Ouaknine, A. W. Roscoe, and J. Worrell. Nets with tokens which carry data. In *Proc. ICATPN 2007*, volume 4546 of *Lecture Notes in Computer Science*, pages 301–320. Springer, 2007.

[23] R. Mayr. Undecidable problems in unreliable computations. *Theoretical Computer Science*, 297(1–3):337–354, 2003.

[24] K. McAloon. Petri nets and large finite sets. *Theoretical Comput. Sci.*, 32(1–2):173–183, 1984.

[25] H. E. Rose. *Subrecursion: Functions and Hierarchies*, volume 9 of *Oxford Logic Guides*. Oxford Univ. Press, 1984.

[26] Ph. Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity. *Information Processing Letters*, 83(5):251–261, 2002.

[27] H. Touzet. *Propriétés combinatoires pour la terminaison de systèmes des réécriture*. Thèse de doctorat, Université de Nancy 1, France, Sept. 1997.

[28] S. S. Wainer. A classification of the ordinal recursive functions. *Arch. math. Logik Grundlag.*, 13(3–4):136–153, 1970.

A Comparing two different ways of losing messages

In section 5.1, the operational semantics of S is given under the form of a transition system $\mathcal{T}_S = (Conf, \rightarrow)$ where we adopted the "write-lossy" semantics, more convenient for our purposes since it provides a better control of the non-determinism in message losses.

The standard semantics, that we shall denote $\mathcal{T}_S^{\text{std}} = (Conf, \rightarrow_{\text{sl}})$, assumes that any messages can be lost before and after any perfect step. That is, it puts

$$\rightarrow_{\text{sl}} \stackrel{\text{def}}{=} \sqsupseteq \circ \rightarrow_{\text{perf}} \circ \sqsupseteq \tag{4}$$

where $\sqsupseteq$ is $\sqsubseteq^{-1}$, i.e., $\sigma \sqsupseteq \sigma'$ iff σ' is obtained from σ by losing messages. Note that the only difference between $\rightarrow_{\text{sl}}$ and $\rightarrow_{\text{perf}} \circ \sqsupseteq$ is that $\rightarrow_{\text{sl}}$ can lose a message that has just been written by the $\rightarrow_{\text{perf}}$ part in (4). Since this can be done by write-lossy steps "$\rightarrow$", $\rightarrow_{\text{sl}}$ and $\rightarrow \circ \sqsupseteq$ coincide!

Lemma A.1 *For all $n > 0$, $\stackrel{n}{\rightarrow}_{\text{sl}} = \stackrel{n}{\rightarrow} \circ \sqsupseteq$.*

Proof. By induction on n. As we just observed, the base case $n = 1$ holds. For the inductive step, we use

$$\begin{aligned}
\stackrel{n+1}{\longrightarrow}_{\text{sl}} = \stackrel{n}{\rightarrow}_{\text{sl}} \circ \rightarrow_{\text{sl}} &= \stackrel{n}{\rightarrow} \circ \sqsupseteq \circ \rightarrow_{\text{sl}} && \text{by ind. hyp.} \\
&= \stackrel{n}{\rightarrow} \circ \rightarrow_{\text{sl}} && \text{using (4)} \\
&= \stackrel{n}{\rightarrow} \circ \rightarrow \circ \sqsupseteq && \text{using case "}n = 1\text{"} \\
&= \stackrel{n+1}{\longrightarrow} \circ \sqsupseteq\,.
\end{aligned}$$

□

In other words, $\sigma \stackrel{n}{\rightarrow}_{\text{sl}} \sigma'$ iff $\rho \stackrel{n}{\rightarrow} \sigma'$ for some $\rho \sqsubseteq \sigma$.

Corollary A.2 *Assume σ has the form $\langle q, \varepsilon, \ldots, \varepsilon \rangle$. Then*
1. σ' is reachable from σ in $\mathcal{T}_S$ iff it is reachable from σ in $\mathcal{T}_S^{\text{std}}$, and
2. there is an infinite run from σ in $\mathcal{T}_S$ iff there is one in $\mathcal{T}_S^{\text{std}}$.

Proof. [Sketch] Since $\rightarrow \subseteq \rightarrow_{\text{sl}}$, we only have to prove the "$\Leftarrow$" implications. 1. is direct from Lemma A.1: when σ has empty channels, $\rho \sqsubseteq \sigma$ requires $\rho = \sigma$. 2. is a consequence of 1., using König's Lemma. □

Therefore, when the initial configuration has empty channels, a LCS satisfy exactly the same reachability and termination properties under the standard semantics, or under the write-lossy semantics we adopted. In particular, exactly the same algorithms can be used.

In the general case where the initial configuration is not necessarily empty, it is easy to reduce reachability and termination from one semantics to the other: one simply encodes the initial channel contents (and its residuals) in the control states, and adds transition rules for these extra states, encoding the original semantics. This many-one reduction is PSPACE when reducing from the standard lossy semantics to the write-lossy semantics,[3] and NLOGSPACE when reducing in the other direction.[4]

B Channel systems that implement stack rewriting

Rule R1 is "$\langle\langle 0, \pi;\, n\rangle\rangle \rightarrow_R \langle\langle \pi;\, n+1\rangle\rangle$". With our differential encoding of stacks, this requires the following transformation:

$$\begin{array}{lcl}
\text{channel } \mathtt{p}: & \overline{\mathtt{l}\,u\,\#} & \overline{u\,\#} \\
 & \stackrel{*}{\longrightarrow} & \\
\text{channel } \mathtt{d}: & \overline{\mathtt{l}^n\,\#} & \overline{\mathtt{l}^{n+1}\,\#}
\end{array}$$

where u is pure. This transformation is performed by the

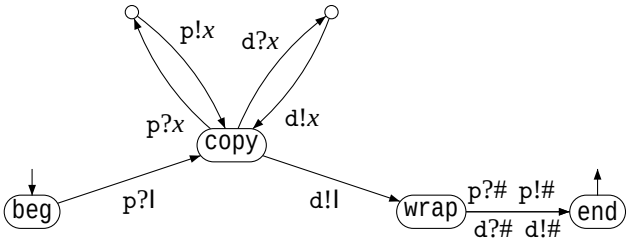


Figure 2. LCS component that implements rule R1 (assuming purity)

LCS depicted in Fig. 2. Here, and in the rest of this section, two simplifying conventions are assumed:

Purity check: the system depicted in Fig. 2 does not check that p contains a pure encoding. This is for improving the clarity of the diagram but, of course, it is easy to check purity (a simple regular property) while performing the transformation. We assume our system deadlocks before reaching state end when purity is not satisfied.

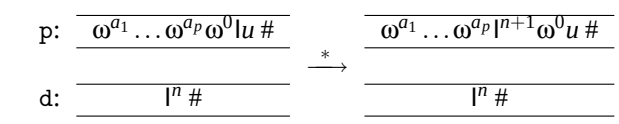
Abbreviated rules: our pictures for LCS uses implicit variables or patterns in order to describe several similar rules at once. For example, the loop $\mathtt{copy} \stackrel{\mathtt{p?}x\ \mathtt{p!}x}{\longrightarrow} \mathtt{copy}$ in Fig. 2 uses x as a variable standing for any message $m \in \mathtt{M}$ so that, letting $k = |\mathtt{M}|$, it abbreviates k loops

[3] Assume the initial configuration $\sigma = (q, u_1, \ldots, u_m)$ has contents of size $k_1 + \cdots + k_m$, writing k_i for $|u_i|$. We need to replace Q with a set N times larger for $N \stackrel{\text{def}}{=} \prod_{i=1}^{m} 2^{k_i}$. For the transition rules, the increasing factor is $O(N^2)$.

[4] Here the increasing factor is only $N \stackrel{\text{def}}{=} \prod_{i=1}^{m} (k_i + 1)$ since less subconfigurations of $u_1, \ldots, u_m$ are meaningful.

(each with a different intermediary state). Other examples are i in Fig. 3, a in Fig. 4, and so on. For these variables, the allowed instantiations are sometimes constrained, as with "$(i > 0)$" or "$(i > a)$" in Fig. 3 and 4.

Rule R2 is "$\langle\langle \alpha+1, \pi;\ n\rangle\rangle \rightarrow_R \langle\langle \overbrace{\alpha, \alpha, ..., \alpha}^{n+1 \text{ times}}, \pi;\ n\rangle\rangle$". With our differential encoding of stacks, this requires the following transformation:



where we assume that $\omega^0 u$ is pure, otherwise the ω^0 is not copied to the right-hand side, as is done in state $*$ (Fig. 3).

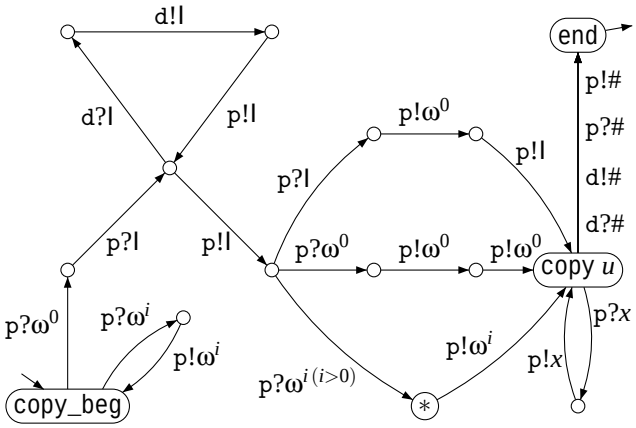
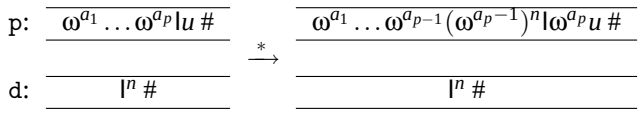


Figure 3. LCS component that implements rule R2 (assuming purity)

Our channel system is actually more complex than depicted in Fig. 3 since it only accepts pure encodings. For example, it will check that $K > a_1 \geq a_2 \geq \cdots a_{p-1} \geq a_p = 0$ while performing the first copy loop (in state `copy_beg`).

Rule R3 is "$\langle\langle \lambda, \pi;\ n\rangle\rangle \rightarrow_R \langle\langle \lambda_n, \pi;\ n\rangle\rangle$". With our differential encoding of stacks, this requires the following transformation:



where it is assumed that $\omega^{a_p} u$ is pure, otherwise the ω^{a_p} is not copied to the right-hand side (see state $*$ in Fig. 4). On top of the usual implicit check for purity "$a_1 \geq a_2 \geq \cdots \geq a_p$", the system depicted in Fig. 4 checks that $(a =) a_p > 0$ so that $\alpha_1 \in \mathit{Lim}$.

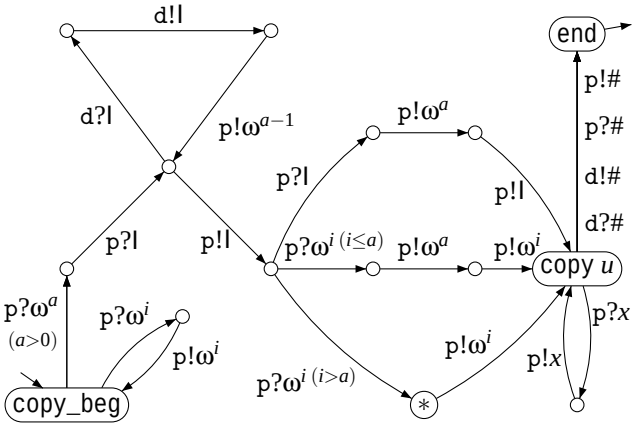
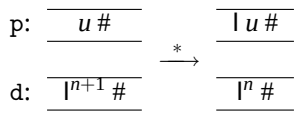


Figure 4. LCS component that implements rule R3 (assuming purity)

Rule S1 is "$\langle\langle \pi;\ n+1\rangle\rangle \rightarrow_S \langle\langle 0, \pi;\ n\rangle\rangle$". With our differential encoding of stacks, this requires the following transformation:



The component that implements S1 behaves like the component for R1, only backwards.

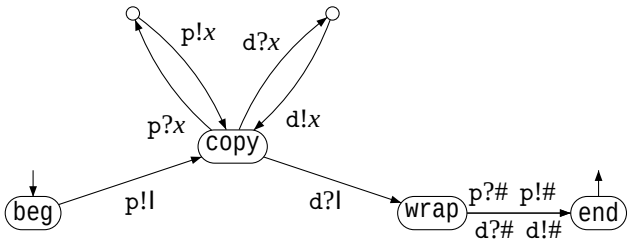
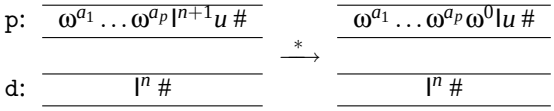


Figure 5. LCS component that implements rule S1 (assuming purity)

Rule S2 is "$\langle\langle \overbrace{\alpha, \alpha, ..., \alpha}^{n+1 \text{ times}}, \pi;\ n\rangle\rangle \rightarrow_S \langle\langle \alpha+1, \pi;\ n\rangle\rangle$" assuming that α does not occur in π.

With our differential encoding of stacks, this requires the following transformation:



where it is now checked that u does not start with |. The component that implements S2 is depicted in Fig. 6. An important feature is the ability to check that the number $n+1$

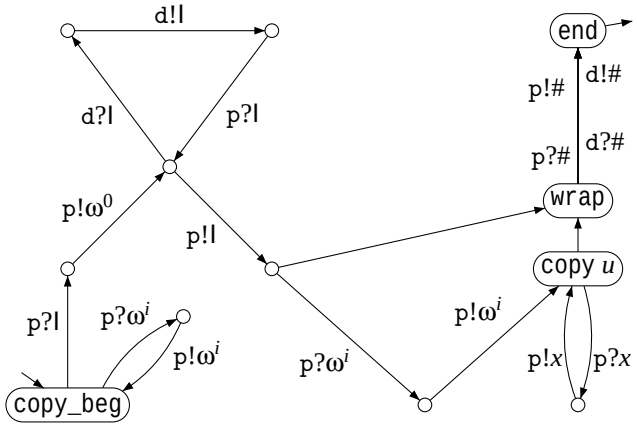


Figure 6. LCS component that implements rule S2 (assuming purity)

of tally symbols after the first ordinal symbols in p matches the number in l^n in d. If there is a mismatch, our system will never reach end.

Rule S3 is "$\langle\langle\lambda_n,\pi\,;\,n\rangle\rangle \rightarrow_S \langle\langle\lambda,\pi\,;\,n\rangle\rangle$" assuming that π does not start with some $\alpha_2 < \lambda$.

With our differential encoding of stacks, this requires the following transformation:

$$\begin{array}{lcc} \mathtt{p}: & \overline{\omega^{a_1}\ldots\omega^{a_p}(\omega^a)^n\mathsf{l}u\,\#} & \overline{\omega^{a_1}\ldots\omega^{a_p}\omega^{a+1}\mathsf{l}v\,\#} \\ & \xrightarrow{\ *\ } & \\ \mathtt{d}: & \overline{\mathsf{l}^n\,\#} & \overline{\mathsf{l}^n\,\#} \end{array}$$

where it is required that $a_p > a$, and where v is obtained from u. More precisely, if u is ε then $v = \varepsilon$, while if u is some $\omega^b u'$, then $v = u$ if $b > a+1$ and $v = u'$ if $b = a+1$. The rule does not apply if $b \leq a$ or if u starts with l, indicating that $\alpha_2 < \lambda$. Here again, the component has to perform a crucial comparison: the number n that is encoded in d must match the number of ω^a symbols in the encoding of the first ordinal in the stack. (As before, the depiction in Fig. 7 does not feature the implicit purity check.)

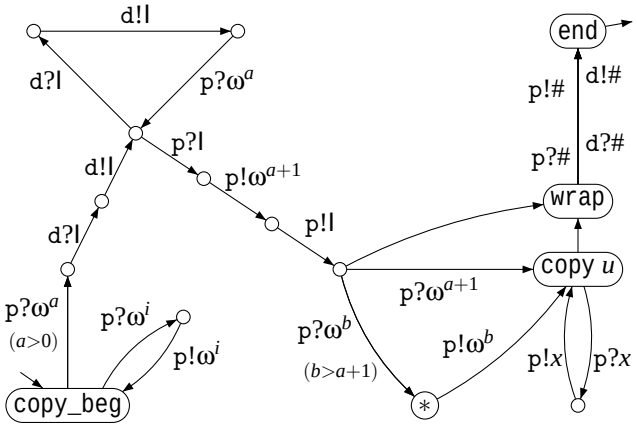


Figure 7. LCS component that implements rule S3 (assuming purity)