

The Ordinal Recursive Complexity of of Timed-Arc Petri Nets, Data Nets, and Other Enriched Nets

Serge Haddad, Sylvain Schmitz, Philippe Schnoebelen

▶ **To cite this version:**

Serge Haddad, Sylvain Schmitz, Philippe Schnoebelen. The Ordinal Recursive Complexity of of Timed-Arc Petri Nets, Data Nets, and Other Enriched Nets. 27th ACM/IEEE Symposium on Logic in Computer Science, Jun 2012, Dubrovnik, Croatia. pp.355–364, 10.1109/LICS.2012.46 . hal-00793811

HAL Id: hal-00793811
https://hal.science/hal-00793811

Submitted on 23 Feb 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

The Ordinal-Recursive Complexity of Timed-Arc Petri Nets, Data Nets, and Other Enriched Nets

Serge Haddad, Sylvain Schmitz, and Philippe Schnoebelen
Laboratoire Spécification et Vérification (LSV),
ENS Cachan & CNRS
Cachan, France

***Abstract*—We show how to reliably compute fast-growing functions with timed-arc Petri nets and data nets. This construction provides ordinal-recursive lower bounds on the complexity of the main decidable properties (safety, termination, regular simulation, etc.) of these models. Since these new lower bounds match the upper bounds that one can derive from wqo theory, they precisely characterise the computational power of these so-called "enriched" nets.**

***Index Terms*—Complexity theory, fast-growing hierarchy, formal verification, Petri nets, well-structured systems**

I. Introduction

We call *enriched nets* a handful of Petri net extensions where tokens are coloured with data values, that still enjoy decidable verification problems: timed-arc Petri nets (TPN) where tokens carry real-valued clocks [6], data nets (DN) and Petri data nets (PDN) where they carry a datum from some dense domain [19], and constrained multiset rewriting systems (CMRS) where they carry positive integers [3]. Their richer structure makes enriched nets a natural choice when modelling for instance parameterised systems, protocols, workflows, or real-time systems—in fact, timed extensions of Petri nets have been in use since the 1970's for such modelling tasks. In spite of the presence of two "sources" of infiniteness, the number of tokens and their colours, enriched nets can be handled by the now standard toolkit of *well quasi-orders* (wqo) and *well-structured transition systems* (WSTS) [5, 14] so that e.g. safety—which in this context corresponds to the *coverability* problem—and other properties are decidable [4, 2, 9].

Recent investigations [1, 7] have shown that all these formalisms are expressively equivalent, i.e. they define the same class of so-called *coverability languages*, and thus in particular their coverability problems are inter-reducible. Their computational complexity, however, has rarely been analysed. The employed wqo and WSTS techniques are generally seen as non-constructive, hence the aforementioned works do not provide any complexity analysis of the algorithms they propose ([19, Prop. 3.2] gives a lower bound: PDNs can simulate lossy channel systems and hence inherit at least their F_{ω^ω} complexity [10], but this is far from optimal).

We prove in this paper that the complexity of enriched nets is exactly at level $F_{\omega^{\omega^\omega}}$ in the fast-growing hierarchy.

1) The upper bound is a consequence of a generic technique described in [22]: the length-function theorem for elementary wqos, here instantiated with $(\mathbb{N}^k)^*$ as the underlying wqo. It applies uniformly to DN, PDNs, TPNs, CMRSs (and to some further extensions); see Sec. IV.

2) The matching lower bound is our main contribution: it relies on the construction of PDNs with $O(k)$ unbounded places that can compute in a weak sense the fast-growing functions $F_{\omega^{\omega^k}}$ and their inverses and therefore simulate $F_{\omega^{\omega^k}}$-bounded computations (see Sec. V for an overview). This construction relies on several intermediate steps: we first define in Sec. VI-A a cumulative encoding of ordinals below $\omega^{\omega^{\omega^k}}$ in sequences of vectors of integers (or "codes") along with rewriting rules over codes implementing fast-growing computations and their inverses (see Sec. VI-B). Because this encoding is *robust*, i.e. safe wrt. Higman's ordering on codes, any *weak implementation* of the rewriting rules will yield the desired behaviour (Sec. VI-C); in particular, a weak PDN implementation is possible, as shown in Sec. VII. Once established for PDNs, the new lower bound automatically applies to TPNs and DNs.

3) Beyond the complexity of verification problems, our techniques are easily applied to the study of the coverability languages of WSTS models [17]. Here our construction directly yields separation results; see Sec. VIII.

The details of the proofs and of the construction of PDNs can be found in the long version of the paper.

II. Petri Data Nets

Although the complexity of decision problems in timed-arc Petri nets provided our prime motivation—an important body of literature is dedicated to their analysis [6, 4, 2, 9] and they are actually employed in tools [e.g. 18]—, we will work exclusively with Petri data nets, which proved easier to manipulate, and rely on known interreducibility results [7, 1] to capture the other classes of enriched nets.

A. Definitions

We denote by $\mathbf{0}$ the null vector in $\mathbb{N}^k$ for any k, and for a word $w = x_1 \cdots x_n$ we write $|w| = n$ and $w(i) = x_i$.

A *Petri Data Net* (PDN) is a Petri net where each token carries an *identity* from a linearly ordered and dense domain

$\mathbb{D}$. A marking s of a PDN with P as set of places could be seen, e.g., as a multiset of pairs in $\mathbb{D} \times P$, or as a map $s \in (\mathbb{N}^P)^{\mathbb{D}}$. However, two key features of PDNs will guide our choice of $(\mathbb{N}^{|P|} \setminus \mathbf{0})^*$ for representing markings:

1) a marking s only has finitely many tokens, thus denoting $d_1 < \cdots < d_m$ the identities that occur in s and gathering all the tokens that carry the same identity d_i, one obtains a (non-null) place vector v_i in $\mathbb{N}^{|P|} \setminus \mathbf{0}$: s can be written as a sequence $(d_1, v_1) \cdots (d_m, v_m)$, implicitly associating the null vector $\mathbf{0}$ with any $d \in \mathbb{D} \setminus \{d_1, \ldots, d_m\}$;
2) the concrete identities d_i are irrelevant, and only their relative *order* is useful wrt. the dynamics of the net, thus s can safely be abstracted as the sequence $v_1 \cdots v_m$ in $(\mathbb{N}^{|P|} \setminus \mathbf{0})^*$. (Also the choice of the concrete set $\mathbb{D}$ is irrelevant.)

Every transition t of a PDN specifies a sequence of n ordered potential identities and for any such identity specifies the tokens $cons$ to be consumed and $prod$ to be produced. Thus $cons(t)$ and $prod(t)$ are two sequences of n (possibly null) place vectors.

Definition 1 (Petri Data Nets)**.** A *k-dimensional Petri Data Net* (k-PDN) is a tuple $\mathcal{N} = (P, T, cons, prod, s_0)$ where

- P is a finite set of $k = |P|$ places,
- T is a finite set of transitions with $P \cap T = \emptyset$,
- for every t in T, $cons(t)$ and $prod(t)$ are finite sequences in $(\mathbb{N}^k)^*$ with $|cons(t)| = |prod(t)|$, and
- s_0 is an initial marking in $(\mathbb{N}^k \setminus \mathbf{0})^*$.

Consider now a marking $s \in (\mathbb{N}^k \setminus \mathbf{0})^*$. In order to fire a transition t with $|cons(t)| = n$, nondeterministically select n identities, consume some of their tokens as indicated by $cons(t)$, and produce new tokens with the identities specified by $prod(t)$. However, some of these n identities might not be present in s, and we should introduce null vectors wherever necessary: $s' \in (\mathbb{N}^k)^*$ is a **0**-*extension* of $s \in (\mathbb{N}^k \setminus \mathbf{0})^*$ (or s is the **0**-*contraction* of s') $\overset{\text{def}}{\Leftrightarrow}$ s is obtained by removing all **0**'s from s'. Once an extension s' is built, select in it a subword of n vectors $x_1, \ldots, x_n$ s.t. every vector contains enough tokens, i.e. with $x_i \geq cons(t)(i)$. If the condition is fulfilled, the corresponding tokens are consumed and $prod(t)(i)$ is added to the resulting vector, yielding a new sequence s''. This s'' may contain null vectors, e.g. when all tokens with some identity have been consumed, hence the reached marking really is the **0**-contraction of s''. Note that any way of firing t requires at most n insertions. Examples of PDNs will be found in Section VII.

Definition 2 (Semantics of PDNs)**.** The *transition system* associated with a k-PDN $\mathcal{N} = (P, T, cons, prod, s_0)$ is $(S, s_0, \to)$ with state set $S \overset{\text{def}}{=} (\mathbb{N}^k \setminus \mathbf{0})^*$ and transition relation $\to \overset{\text{def}}{=} \bigcup_{t \in T} \xrightarrow{t}$, where $s \xrightarrow{t} s'$ for $t \in T$ iff, letting $n = |cons(t)|$:

- there exists $u_0 x_1 u_1 \cdots u_{n-1} x_n u_n$ a **0**-extension of s with for all i, $u_i \in (\mathbb{N}^k)^*$ and $x_i \in \mathbb{N}^k$;
- for i in $\{1, \ldots, n\}$, $x_i \geq cons(t)(i)$;
- and defining $y_i = x_i - cons(t)(i) + prod(t)(i)$, s' is the **0**-contraction of $u_0 y_1 u_1 \cdots u_{n-1} y_n u_n$.

Below we consider three decision problems for PDNs:

(Strong) Coverability: Given $\mathcal{N}$ and $p \in P$, can we reach a configuration where p holds at least one token?

Boundedness: Given $\mathcal{N}$, is the set of reachable configurations in S finite?

Termination: Given $\mathcal{N}$, is every run finite?

B. PDNs as Well-Structured Transition Systems

A wqo $(A, \leq)$ is a set A endowed with a transitive and reflexive relation $\leq$ s.t. every infinite sequence $\sigma = a_0, a_1, \cdots$ of elements of A contains a pair $a_i \leq a_j$ for some $i < j$. Some classical examples of wqos are

Dickson's Lemma: $(\mathbb{N}^k, \leq)$ with the product ordering defined by $\mathbf{x} \leq \mathbf{y} \overset{\text{def}}{\Leftrightarrow} \forall 0 \leq j < k, \mathbf{x}[j] \leq \mathbf{y}[j]$,

Higman's Lemma: if $(A, \leq)$ is a wqo, then $(A^*, \leq_*)$ the set of finite sequences of elements of A along with the subword embedding ordering is also a wqo, where $\leq_*$ is defined by $s \leq_* s' \overset{\text{def}}{\Leftrightarrow} s = a_1 \cdots a_n$, $s' = x_0 b_1 x_1 \cdots b_n x_n$ with $x_0, \ldots, x_n$ in A^* and $a_i \leq b_i$ for all $1 \leq i \leq n$.

The transition system associated with a PDN (see Definition 2) is *well-structured* [5, 14] for the wqo $(S, \leq_*)$: if $s_1 \xrightarrow{t} s_2$ and $s_1 \leq_* s_3$, then there exists s_4 s.t. $s_2 \leq_* s_4$ and $s_3 \xrightarrow{t} s_4$. This (strict) *compatibility* of the transition relation with the ordering allows to employ generic algorithms for deciding coverability, boundedness, and termination. In fact, the same generic WSTS algorithms show that coverability, boundedness and termination are decidable

- for TPNs [2], even when extended with *read arcs* [9] or *transport arcs* [18],
- for CMRSs [3], and
- as far as coverability and termination are concerned, for DNs [19]. Compared to PDNs, these allow so-called *whole-place* operations that can e.g. duplicate or erase the whole contents of some places, and/or transfer them to other places, which makes their compatibility non-strict—and indeed their boundedness problem is undecidable.

III. Ordinal Recursive Complexity

The enormous complexity of some decidable problems on WSTSs requires the introduction of complexity classes spanning way beyond the usual polynomial or exponential hierarchies. The complexity classes we consider are generated by ordinal-indexed *subrecursive hierarchies*, like the Hardy hierarchy and the fast-growing hierarchy. See [12] for a self-contained presentation; we only recall below the notions and notations that are required for our construction in Section VI.

It is well-known that any ordinal $\alpha < \varepsilon_0$ can be written uniquely in Cantor Normal Form (CNF). In this paper we use a dotted addition symbol "$\dot{+}$" when we want to stress that an ordinal term is in CNF. Thus, when we write

$$\alpha = \omega^{\alpha_1} \mathbin{\dot{+}} \cdots \mathbin{\dot{+}} \omega^{\alpha_p} , \tag{1}$$

we mean that not only the equality (1) holds, but also that $\alpha_p \leq \ldots \leq \alpha_1 < \alpha$, as required by CNF. (NB: we allow writing $\alpha \dot{+} \alpha'$ when α or α' is 0.)

Subrecursive hierarchies are defined through assignments of *fundamental sequences* $(\lambda_n)_{n<\omega}$ for limit ordinals $\lambda < \varepsilon_0$, verifying $\lambda_n < \lambda$ for all x and $\lambda = \sup_n \lambda_n$. A standard assignment is defined by:

$$\left(\gamma \dot{+} \omega^{\alpha+1}\right)_n \stackrel{\text{def}}{=} \gamma \dot{+} \omega^{\alpha} \cdot \omega_n, \quad \left(\gamma \dot{+} \omega^{\lambda}\right)_n \stackrel{\text{def}}{=} \gamma \dot{+} \omega^{\lambda_n}, \quad (2)$$

together with $\omega_n \stackrel{\text{def}}{=} n$. Writing Ω for the ordinal $\omega^{\omega^{\omega}}$, this yields for instance $\Omega_k = \omega^{\omega^{\omega^k}}$ and $(\Omega_k)_k = \omega^{\omega^{\omega^{k-1} \cdot k}}$.

The *Hardy hierarchy* $(H^\alpha : \mathbb{N} \to \mathbb{N})_{\alpha<\varepsilon_0}$ is defined by $H^0(n) \stackrel{\text{def}}{=} n$ and

$$H^{\alpha+1}(n) \stackrel{\text{def}}{=} H^\alpha(n+1), \qquad H^\lambda(n) \stackrel{\text{def}}{=} H^{\lambda_n}(n). \quad (3)$$

Observe that H^1 is the successor function, and more generally H^α is the αth iterate of the successor function, using diagonalisation to treat limit ordinals. The *fast growing hierarchy* $(F_\alpha : \mathbb{N} \to \mathbb{N})_{\alpha<\varepsilon_0}$ can be defined by $F_\alpha \stackrel{\text{def}}{=} H^{\omega^\alpha}$, resulting in $F_0(n) = H^1(n) = n+1$, $F_1(n) = H^\omega(n) = H^n(n) = 2n$, $F_2(n) = H^{\omega^2}(n) = 2^n n$ being exponential, F_3 non-elementary, and F_ω an Ackermannian function.

By applying elementary closure operations to the collection of functions $(F_\beta)_{\beta\leq\alpha}$ along with the addition, projection and zero functions, one obtains a hierarchy $(\mathscr{F}_\alpha)_\alpha$ known as the *extended Grzegorczyck hierarchy* [20], which characterises several natural classes of functions; for instance $\mathscr{F}_0 = \mathscr{F}_1$ is the class of linear functions, $\mathscr{F}_2$ that of elementary ones, $\bigcup_{k\in\mathbb{N}} \mathscr{F}_k$ of primitive-recursive ones, and $\bigcup_{k\in\mathbb{N}} \mathscr{F}_{\omega^k}$ of multiply-recursive ones. The hierarchy is strict for all $0 < \alpha < \alpha'$: $\mathscr{F}_\alpha \subsetneq \mathscr{F}_{\alpha'}$, in particular because $F_{\alpha'} \notin \mathscr{F}_\alpha$.

The $(F_\alpha)_\alpha$ hierarchy provides a more abstract packaging of the main stops in the extended Grzegorczyk hierarchy and requires lighter notation than the Hardy hierarchy $(H^\alpha)_\alpha$. However, with its tail-recursive definition, the Hardy hierarchy is easier to implement as a while-program or as a counter machine. Below we weakly implement Hardy computations with PDNs. Formally, a (forward) *Hardy computation* is a sequence $\alpha_0, n_0 \to \alpha_1, n_1 \to \alpha_2, n_2 \to \cdots \to \alpha_\ell, n_\ell$ of evaluation steps implementing Eq. (3) seen as left-to-right rewrite rules. It guarantees $\alpha_0 > \alpha_1 > \alpha_2 > \cdots$ and $n_0 \leq n_1 \leq n_2 \leq \cdots$ and keeps $H^{\alpha_i}(n_i)$ invariant. We say it is *complete* when $\alpha_\ell = 0$ and then $n_\ell = H^{\alpha_0}(n_0)$ (we also consider incomplete computations). A *backward* Hardy computation is obtained by using Eq. (3) as right-to-left rules. For instance, $\Omega, k \to \Omega_k, k \to (\Omega_k)_k, k$ constitute the first three steps of the forward Hardy computation starting from Ω, k if $k > 0$.

IV. Complexity Upper Bounds

A key insight for the complexity analysis of WSTS algorithms is that the use of wqos yields not only algorithm termination, but also upper complexity bounds:

Theorem 3 (Upper Bound)**.** *Coverability and termination for PDNs, DNs, TPNs, and CMRSs, are in $F_{\omega^\omega}\big(n + O(1)\big)$.*

For this result, as explained in [13, 22], we merely need to find out (1) what is the complexity of a step of the WSTS (in the case of PDNs, transitions perform simple affine operations in $\mathscr{F}_1$), and (2) the *maximal order type* of the wqo, which is a measure of its complexity (this is Ω_k for a k-PDN). By the length function theorem for elementary wqos (see full version of [22]), we then obtain a parameterised upper bound in $\mathscr{F}_{\omega^{\omega^k}}$ for the decision problems of k-PDNs mentioned in Section II-B, and a uniform F_{ω^ω} upper bound (which asymptotically majorises every function in $\bigcup_k \mathscr{F}_{\omega^{\omega^k}}$) when the dimension is not fixed.

These upper bounds hold more generally for k-DNs, as they have the same order type and their extra whole-place operations are still in $\mathscr{F}_1$. Regarding TPNs and CMRSs, the F_{ω^ω} upper bound also holds; however here the main parameter in the parameterised complexity—which appears as the exponent on top of the tower of ω's—is not simply the dimension k but km where m is the maximal constant that appears in the constraints put on transitions or in the initial marking.

V. Complexity Lower Bounds

We now describe the proof plan for our main result.

Theorem 4 (Lower Bound)**.** *Coverability and termination for PDNs are F_{ω^ω}-hard.*

This follows from a reduction from the halting problem for Minsky machines (MM) M with counters bounded by $F_{\omega^\omega}(|M|)$. The proof is done by assembling two constructions (described in the following sections). The schematics (see Fig. 1) are similar to earlier constructions for lossy channel systems or counter machines and the reader can refer to [23, 10] where more lower-level details are given. We outline it as a motivation for the following sections.

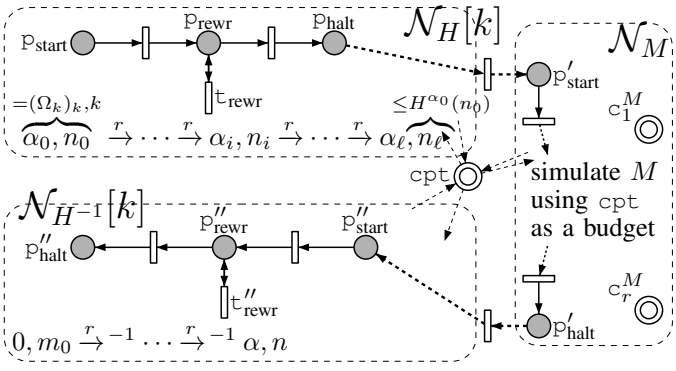


Fig. 1. Schematics for Theorem 4.

1) Direct Computation: For a provided size k, we first construct (see Section VII) a PDN $\mathcal{N}_H[k]$ initialised with a pair $\alpha_0, n_0 \stackrel{\text{def}}{=} (\Omega_k)_k, k$ and that tries to rewrite it $\alpha_0, n_0 \to \alpha_1, n_1 \to \cdots \to \alpha_\ell, n_\ell$ in a way that reflects precisely the complete Hardy derivation issuing from α_0, n_0, thus computing $n_\ell = H^{\alpha_0}(n_0) = H^\Omega(k)$. There are two difficulties here. First, one has to encode ordinals in sequences of vectors (i.e.

in PDN configurations) and this is the topic of Section VI. Secondly, our PDN only performs Hardy computations in a weak sense. What is guaranteed is the following:

Lemma 5 (See Section VII)**.** *$\mathcal{N}_H[k]$ is "complete": Starting with $\alpha_0 = (\Omega_k)_k$, $\mathcal{N}_H[k]$ can perform the exact Hardy computation and halt with $\alpha_\ell = 0$ and $n_\ell = H^\Omega(k)$.*
$\mathcal{N}_H[k]$ is "safe": Any halting computation in $\mathcal{N}_H[k]$, correct or incorrect, has $n_\ell \leq H^\Omega(k)$.

2) Simulation: Now consider some MM M of size k. An easy (see [11, §7] or [21, §4]) and classic construction yields a PDN $\mathcal{N}_M$ that simulates M as far as halting is concerned: $\mathcal{N}_M$ has unbounded places $\mathtt{c}_1^M, \ldots, \mathtt{c}_r^M$ to simulate the r counters of M. Starting from control place $\mathtt{p}'_{\text{start}}$ and with empty $\mathtt{c}_1^M, \ldots, \mathtt{c}_r^M$, it eventually reaches $\mathtt{p}'_{\text{halt}}$ with $\mathtt{c}_1^M, \ldots, \mathtt{c}_r^M$ empty iff M halts. We further modify $\mathcal{N}_M$ so that it uses $\mathtt{cpt}$ (where $\mathcal{N}_H[k]$ stores n_ℓ) as a budget, i.e. any incrementing of a $\mathtt{c}_i^M$ is matched by decrementing $\mathtt{cpt}$ and vice versa, see [23, §4]. Adding $\mathtt{cpt}$ as a budget has two consequences. First, M is now simulated with an upper bound of n_ℓ for (the sum of) its counters, at any time along its run. Second, when $\mathcal{N}_M$ reaches $\mathtt{p}'_{\text{halt}}$, witnessing that M halts, does not require testing $\mathtt{c}_1^M, \ldots, \mathtt{c}_r^M$ for emptiness: at this point, the value n_f of $\mathtt{cpt}$ is necessarily $\leq n_\ell$, and only equals n_ℓ if $\mathtt{c}_1^M, \ldots, \mathtt{c}_r^M$ are empty (NB: if M halts, $n_f = n_\ell$ is indeed feasible).

3) Inverse Computation: We now connect $\mathcal{N}_H[k]$, $\mathcal{N}_M$ and $\mathcal{N}_{H^{-1}}[k]$ so that they run sequentially. Here $\mathcal{N}_{H^{-1}}[k]$ is a PDN for backward Hardy computations. It starts with $\alpha'_0 = 0$ and $m_0 = n_f$ (passed on by $\mathcal{N}_M$). Its backward computation may reach $(\Omega_k)_k, k$ if $m_0 = H^\Omega(k)$. Here too, the PDN only computes H^{-1} in a weak sense but it is guaranteed that it can do exact backward computations (completeness) and that incorrect backward computations halting on $(\Omega_k)_k, n$ have $H^{(\Omega_k)_k}(n) \leq m_0$ (safety).

As a consequence, the resulting full PDN started with $(\Omega_k)_k, k$ can reach a configuration with $\mathtt{p}''_{\text{halt}}$ and a pair α, n that covers $(\Omega_k)_k, k$ (in terms of the places that store the current Hardy pair) *if, and only if*, the Minsky machine M with space bounded by $H^\Omega(k) = F_{\omega^\omega}(k)$ halts.

Indeed, if M halts within the space bound, the PDN may reach the required $\mathtt{p}''_{\text{halt}}, \alpha, n$ by chaining exact Hardy computations and the simulation of M by $\mathcal{N}_M$. More interestingly, if the required α, n is reached, we know, letting $h \stackrel{\text{def}}{=} H^\Omega(k)$, that $n_\ell \leq h$ (safety of $\mathcal{N}_H$), that $n_\ell \geq n_f = m_0$ (budget of $\mathcal{N}_M$), that $H^\alpha(n) \leq m_0$ (safety of $\mathcal{N}_{H^{-1}}$), and that $H^\alpha(n) \geq h$ (α, n covers $(\Omega_k)_k, k$). Thus $h \leq H^\alpha(n) \leq m_0 = n_f \leq n_\ell \leq h$. Necessarily $n_\ell = n_f$, witnessing that M halts, and $n_\ell = h$, witnessing that M runs in space bounded by $h = F_{\omega^\omega}(k)$.

In conclusion, the construction provides a (logspace) many-one reduction from the halting problem for Minsky machines running in space bounded by $F_{\omega^\omega}(k)$ where $k = |M|$ is the size of the MM description. Using standard complexity-theoretical arguments, Theorem 4 (for Coverability) follows.

4) Termination: Regarding termination, a similar reduction works. One makes sure that $\mathcal{N}_H[k]$ always halts or deadlocks (it does) and stores two copies of n_ℓ: one is a time budget that ensures the eventual halting-or-deadlock of $\mathcal{N}_M$, and the other witnesses $n_f = n_\ell$ as earlier. In the end, the whole system has to eventually stop, unless it can cover $(\Omega_k)_k, k$ with α, n, finally enabling an infinite loop. This reduces the same MM problem to termination for PDNs. More details can be found in [23, §7] where the same adaptation is done.

Using the simulations of PDNs by TPNs of [7] and of DNs by CMRSs of [1, §5], we conclude:

Corollary 6. *Coverability and Termination for TPNs, DNs, and CMRSs are F_{ω^ω}-hard.*

VI. Encoding Hardy Computations

We define in this section a so-called "cumulative" encoding of ordinals as *codes* (Section VI-A) and a rewriting system $\xrightarrow{r}$ operating on codes that performs Hardy computations (Section VI-B). Its crucial property is its *robustness*, which entails that weak implementations, like the PDN implementation we present in Section VII, are correct (see Section VI-C).

A. Encoding Ordinals as Cumulative Vector Sequences

Fix $k \in \mathbb{N}$. An ordinal $< \omega^k$ is "*small*" and we use $\beta, \beta', \ldots$ to denote small ordinals; an ordinal $< \omega^{\omega^k}$ is "*medium*" and we use $\alpha, \alpha', \ldots$ for such ordinals; finally, an ordinal $< \Omega_k$ is "*large*" and we use $\pi, \pi', \ldots$ for such ordinals. A medium ordinal can be written in CNF as $\alpha = \omega^{\beta_1} \dotplus \cdots \dotplus \omega^{\beta_p}$ where $\beta_1, \ldots, \beta_p$ are small ordinals, and a large ordinal can be written as $\pi = \omega^{\alpha_1} \dotplus \cdots \dotplus \omega^{\alpha_m}$ where $\alpha_1, \ldots, \alpha_m$ are medium ordinals.

We now introduce an encoding of large ordinals that will allow the computation of the Hardy functions with PDNs. These data structures are *1)* k-dimensional *vectors* in $\mathbb{N}^k$ for small ordinals, *2) vector sequences* in $(\mathbb{N}^k)^*$ for medium ordinals, and *3) cumulative encodings* in $(\mathbb{N}^k \uplus \{\#\})^*$ for large ordinals, where $\#$ is a fresh tally symbol.

1) Small Ordinals as Vectors: For $v \in \mathbb{N}^k$ and an index $0 \leq i < k$, let $v[i] \in \mathbb{N}$ denote the i-th component of v. We use two different orderings over $\mathbb{N}^k$: the *product ordering*, denoted $v \leq v'$ and the *lexicographic ordering*, denoted $v \leq_{\text{lex}} v'$, with most significant component at index $k-1$. Recall that $\leq$ is a wqo, and that $\leq_{\text{lex}}$ is a linearization of $\leq$.

With a vector $v \in \mathbb{N}^k$, we associate the small ordinal

$$\beta(v) \stackrel{\text{def}}{=} \omega^{k-1} \cdot v[k-1] \dotplus \cdots \dotplus \omega^0 \cdot v[0] \,. \tag{4}$$

This establishes a bijective correspondence between $\mathbb{N}^k$ and small ordinals, and we write $v(\beta)$ for $\beta^{-1}(\beta)$. We write $\mathbf{1}_i$ for the vector with $v[i] = 1$ and $v[j] = 0$ for all $j \neq i$. Hence $\beta(\mathbf{0}) = 0$ and $\beta(\mathbf{1}_i) = \omega^i$.

The bijection relates the two linear orderings of small ordinals and of vectors in $\mathbb{N}^k$ since

$$v \leq_{\text{lex}} v' \text{ iff } \beta(v) \leq \beta(v') \,. \tag{5}$$

2) Medium Ordinals as Vector Sequences: With a finite sequence $\mathbf{V} = v_1 v_2 \cdots v_p \in \mathbb{N}^{k*}$, we associate the ordinal

$$\alpha(\mathbf{V}) = \alpha(v_1 v_2 \cdots v_p) \stackrel{\text{def}}{=} \omega^{\beta(v_1)} + \cdots + \omega^{\beta(v_p)} \,. \quad (6)$$

This surjective[1] embedding of $\mathbb{N}^{k*}$ into ω^{ω^k} satisfies $\alpha(\mathbf{V}\mathbf{V}') = \alpha(\mathbf{V}) + \alpha(\mathbf{V}')$. Write $\boldsymbol{\varepsilon}$ for the empty sequence in $\mathbb{N}^{k*}$. Then $\alpha(\mathbf{V}) = 0$ iff $\mathbf{V} = \boldsymbol{\varepsilon}$, and $\alpha(\mathbf{V}) = 1$ iff $\mathbf{V} = \mathbf{0}$.

Example 7. Consider $k = 2$: $\alpha(\mathbf{1}_0) = \alpha(|^{\mathbf{0}}_{\mathbf{1}}) = \omega^{\beta(\mathbf{1}_0)} = \omega^{\omega^0 \cdot 1} = \omega^1 = \omega$, thus $\alpha(|^{\mathbf{0}}_{\mathbf{1}}\,|^{\mathbf{0}}_{\mathbf{1}}) = \omega \cdot 2$, while $\alpha(2 \times \mathbf{1}_0) = \alpha(|^{\mathbf{0}}_{\mathbf{2}}) = \omega^{\omega^0 \cdot 2} = \omega^2$. □

We order vector sequences with $\leq_*$, the sequence extension of $\leq$: it is a wqo since $\leq$ is.

We say that $\mathbf{V} = v_1 \cdots v_p$ is *pure* if $v_1 \geq_{\text{lex}} v_2 \geq_{\text{lex}} \cdots \geq_{\text{lex}} v_p$: restricted to pure vector sequences, the embedding in (6) is bijective since the expression giving $\alpha(\mathbf{V})$ in Eq. (6) is in CNF. We write $pure(\mathbf{V})$ for the only pure $\mathbf{V}'$ such that $\alpha(\mathbf{V}) = \alpha(\mathbf{V}')$: one obtains $pure(\mathbf{V})$ by removing in $\mathbf{V} = v_1 \cdots v_p$ any v_i such that $v_i <_{\text{lex}} v_j$ for some $j > i$. Hence $pure(\mathbf{V}) \leq_* \mathbf{V}$.

3) Cumulative Encodings for Large Ordinals: Fix a special *tally symbol* $\#$ and let $\mathbb{N}^k_\# \stackrel{\text{def}}{=} \mathbb{N}^k \cup \{\#\}$. A *cumulative ordinal description*, or simple a "code", is a sequence x in ${\mathbb{N}^k_\#}^*$. Below we see them as sequences in $[\mathbb{N}^{k*}\#]^*\mathbb{N}^{k*}$, i.e. we single out the tally symbols and factor codes under the form

$$x = \mathbf{V}_1 \# \mathbf{V}_2 \# \cdots \# \mathbf{V}_m \# \mathbf{V}_{\text{rest}} \,, \quad (7)$$

where the $\mathbf{V}_i$'s are vector sequences. We extend $\leq_*$ from vector sequences to codes in the natural way, by requiring that a $\#$ embeds into a $\#$: this is still a wqo.

With x we associate a large ordinal $\pi(x)$ via the following

$$\begin{aligned} &\pi(\mathbf{V}_1 \# \mathbf{V}_2 \# \cdots \# \mathbf{V}_m \# \mathbf{V}_{\text{rest}}) \stackrel{\text{def}}{=} \\ &\quad \omega^{\alpha(\mathbf{V}_1 \mathbf{V}_2 \cdots \mathbf{V}_m)} \dot{+} \cdots \dot{+} \omega^{\alpha(\mathbf{V}_1 \mathbf{V}_2)} \dot{+} \omega^{\alpha(\mathbf{V}_1)} \,. \end{aligned} \quad (8)$$

The above definition explains why codes are called cumulative. One can also define π inductively by

$$\pi(\mathbf{V}) = 0 \,, \qquad \pi(\mathbf{V}\#x) = \omega^{\alpha(\mathbf{V})} \cdot \pi(x) \dot{+} \omega^{\alpha(\mathbf{V})} \,. \quad (9)$$

We say that x is *pure* if each $\mathbf{V}_i$, $i = 1, \ldots, m$, is pure, and if in addition $\mathbf{V}_{\text{rest}} = \boldsymbol{\varepsilon}$. (NB: purity of, e.g., $\mathbf{V}_1 \# \mathbf{V}_2 \#$, does not guarantee purity of $\mathbf{V}_1 \mathbf{V}_2$.) For a code x, the unique pure x' such that $\pi(x) = \pi(x')$, denoted $x' = pure(x)$, is given by

$$\begin{aligned} &pure(\mathbf{V}_1 \# \mathbf{V}_2 \# \cdots \# \mathbf{V}_m \# \mathbf{V}_{\text{rest}}) = \\ &\quad pure(\mathbf{V}_1) \# pure(\mathbf{V}_2) \# \cdots \# pure(\mathbf{V}_m) \# \boldsymbol{\varepsilon} \,. \end{aligned} \quad (10)$$

Lemma 8. *$x \leq_* x'$ implies $pure(x) \leq_* pure(x')$.*

Lemma 9 (Bijection)**.** *Pure codes in ${\mathbb{N}^k_\#}^*$ and large ordinals in Ω_k are in bijection by π.*

[1]This is not bijective, e.g. for $v <_{\text{lex}} v'$, $\alpha(v\,v') = \omega^{\beta(v)} + \omega^{\beta(v')} = \omega^{\beta(v')} = \alpha(v')\big(\neq \omega^{\beta(v')} \dot{+} \omega^{\beta(v)}\big)$.

If we write $V(x)$ for the vector sequence obtained by removing all tally symbols from x, i.e. the result of the projection $\# \mapsto \varepsilon$ applied to x, then

$$\pi(x_1 \# x_2) = \pi(pure(V(x_1) x_2)) \dot{+} \pi(x_1 \#) \,. \quad (11)$$

Example 10. Let $k > 1$; the initial Hardy computation step $x_0, k \to x_1, k$ with pure codes x_0, x_1 is defined by

$$\begin{aligned} &x_0 \stackrel{\text{def}}{=} (\mathbf{1}_{k-1})^k \# \,; && \pi(x_0) = (\Omega_k)_k \,; \\ &x_1 \stackrel{\text{def}}{=} (\mathbf{1}_{k-1})^{k-1} (\mathbf{1}_{k-2})^k \# \,; && \pi(x_1) = \omega^{\omega^{\omega^{k-1} \cdot (k-1) + \omega^{k-2} \cdot k}} \,. \end{aligned}$$

B. Rewriting of Ordinal Codes

Let us turn to the encoding of Hardy computations (below Ω_k) as rewriting rules on codes. Such a system should e.g. map x_0 to x_1 in Example 10. It turns out that the bulk of the task when computing Hardy functions lies in computing the elements in the fundamental sequences of limit ordinals.

1) Limit Ordinals: Observe that a code denotes a successor ordinal if it is of the form $\#x$, as indeed $\pi(\#x) = \omega^0 \cdot \pi(x) + \omega^0 = \pi(x) + 1$. Conversely, a pure code of form $\mathbf{V}v\#x$ denotes a limit ordinal $\pi(\mathbf{V}vx) \dot{+} \omega^{\alpha(\mathbf{V}v)}$ s.t. $\big(\pi(\mathbf{V}v\#x)\big)_n = \pi(\mathbf{V}vx) \dot{+} (\omega^{\alpha(\mathbf{V}v)})_n$. We want to define a similar mapping $(.)_n$ from codes to codes s.t. $\pi\big((x)_n\big) = \big(\pi(x)\big)_n$; this mapping essentially needs to treat the head $\mathbf{V}v$, which contributes the smallest term $\omega^{\alpha(\mathbf{V}v)}$ to the encoded ordinal. Several cases arise depending on v:

- if $v = \mathbf{0}$, i.e. $\omega^{\beta(v)} = 1$, then $\omega^{\alpha(\mathbf{V}\mathbf{0})} = \omega^{\alpha(\mathbf{V})+1}$ verifies $(\omega^{\alpha(\mathbf{V}\mathbf{0})})_n = \omega^{\alpha(\mathbf{V})} \cdot n$, encoded through

$$(\mathbf{V}\mathbf{0})_n \stackrel{\text{def}}{=} \mathbf{V}\#^n \,. \quad (12)$$

  Thus we verify

$$\begin{aligned} \pi\big((\mathbf{V}\mathbf{0}\#x)_n\big) &\stackrel{\text{def}}{=} \pi(\mathbf{V}\#^n \mathbf{0} x) \\ &= \pi(\mathbf{V}\mathbf{0}x) \dot{+} \omega^{\alpha(\mathbf{V})} \cdot n \\ &= \big(\pi(\mathbf{V}\mathbf{0}\#x)\big)_n \,. \end{aligned} \quad (13)$$

- if $v \neq \mathbf{0}$ then $(\omega^{\alpha(\mathbf{V}v)})_n = \omega^{\alpha(\mathbf{V}) \dot{+} (\omega^{\beta(v)})_n}$ (recall that $\mathbf{V}v$ is pure) is encoded by

$$(\mathbf{V}v)_n \stackrel{\text{def}}{=} \mathbf{V}(v)_n \# \,, \quad (14)$$

  and we need to further distinguish two cases: let $i \in \{0, \ldots, k-1\}$ be the smallest index with $v[i] > 0$. Then $\beta(v)$ is a successor ordinal if $i = 0$ and a limit ordinal otherwise, hence the definition

$$(v)_n \stackrel{\text{def}}{=} \begin{cases} (v - \mathbf{1}_0)^n & \text{if } i = 0, \\ v - \mathbf{1}_i + n \cdot \mathbf{1}_{i-1} & \text{otherwise.} \end{cases} \quad (15)$$

  Since every vector in the sequence $(v)_n$ is $<_{\text{lex}} v$, this verifies

$$\begin{aligned} \pi\big((\mathbf{V}v\#x)_n\big) &\stackrel{\text{def}}{=} \pi(\mathbf{V}(v)_n \# vx) \\ &= \pi(\mathbf{V}(v)_n vx) \dot{+} \big(\omega^{\alpha(\mathbf{V}v)}\big)_n \\ &= \pi(\mathbf{V}vx) \dot{+} \big(\omega^{\alpha(\mathbf{V}v)}\big)_n \,. \end{aligned} \quad (16)$$

The definitions (12–16) thus result for a pure $\mathbf{V}v$ in

$$\pi\big((\mathbf{V}v\#x)_n\big) = \pi\big((\mathbf{V}v)_n vx\big) = \big(\pi(\mathbf{V}v\#x)\big)_n \,. \quad (17)$$

2) Rewriting System: We define a set of rewriting rules $\xrightarrow{r}$ working on pairs (x, n) of a code x and a number $n \in \mathbb{N}$, that together encode an intermediate stage $H^{\pi(x)}(n)$ in the course of a Hardy computation.

Definition 11 ($\xrightarrow{r}$)**.** The relation $x, n \xrightarrow{r} y, m$ is given by rules (R1–R2) below.

$$\#x, n \xrightarrow{r} x, n+1 \tag{R1}$$

$$\mathbf{V}v\#x, n \xrightarrow{r} \begin{cases} (\mathbf{V}v)_n x, n & \text{if } x = \varepsilon \text{ or } x = v'x' \\ & \text{with } v <_{\text{lex}} v' \\ (\mathbf{V}v)_n vx, n & \text{if } x = \#x' \text{ or } x = v'x' \\ & \text{with } v' \leq_{\text{lex}} v \end{cases} \tag{R2}$$

Rule (R2) implements the case of limit ordinals and is correct by (17)—the first subcase includes a purification step when $\pi((\mathbf{V}v)_n vx) = \pi((\mathbf{V}v)_n x)$—while rule (R1) handles successor ordinals:

Proposition 12 (Correctness of $\xrightarrow{r}$)**.** *$x, n \xrightarrow{r} y, m$ and x pure imply $H^{\pi(x)}(n) = H^{\pi(y)}(m)$.*

Remark 13 (Purity is required). A step $x, n \xrightarrow{r} y, m$ is not always correct when x is not pure: e.g. if $k = 1$, $\pi(01\#) = \omega + \omega^\omega = \omega^\omega$ but $01\#, n \xrightarrow{r} 0^{n+1}\#, n$, which encodes ω^{n+1}, and $H^{\omega^\omega}(n) = H^{\omega^n}(n) < H^{\omega^{n+1}}(n)$. □

It is convenient to work with pure codes in proofs: the one-to-one correspondence between pure codes and ordinals in Ω_k yields a one-to-one correspondence between a pair (x, n) and a snapshot of a Hardy computation $H^{\pi(x)}(n)$, allowing to transfer results from Hardy computations to $\xrightarrow{r}$.

More importantly, note that Proposition 12 entails the correctness of $\xrightarrow{r}$ even when applied backwards: we capture both forward and backward Hardy computations with the same rewriting system.

C. Robustness of $\xrightarrow{r}$

So far, our encoding of ordinals in Ω_k and the rewriting system $\xrightarrow{r}$ can be seen as a (rather convoluted) way of performing forward and backward Hardy computations using sequences of vectors. Their critical interest compared to more basic ordinal encodings is that $\xrightarrow{r}$ is *robust*: if instead of computing with x, n we first decrease the configuration in an uncontrolled way to some y, m with $y \leq_* x$ and $m \leq n$, we obtain a configuration that codes a smaller value $H^{\pi(y)}(m) \leq H^{\pi(x)}(n)$. This result is subject to hygienic conditions on x, n and y, m; see Proposition 16 for the exact statement.

Remark 14 (Non-Robustness of CNF). Let us pause for a moment and consider a natural encoding χ of large ordinals. In this encoding, we use the CNF of the ordinal and separate pure vector sequences with "$\dot{+}$" symbols s.t. $\chi(\mathbf{V}_1 \dot{+} \cdots \dot{+} \mathbf{V}_m) \stackrel{\text{def}}{=} \omega^{\alpha(\mathbf{V}_1)} \dot{+} \cdots \dot{+} \omega^{\alpha(\mathbf{V}_m)}$; e.g. $p = 1\dot{+}0$ codes $\chi(p) = \omega^\omega \dot{+} \omega$ for $k = 1$. However, $q = 10$ verifies $q \leq_* p$ and codes the much larger ordinal $\chi(q) = \omega^{\omega+1}$, with $H^{\chi(p)}(n) = H^{\omega^\omega \dot{+} \omega}(n) < H^{\omega^\omega \cdot (n-1) \dot{+} \omega^n}(n) = H^{\chi(q)}(n)$ when $n > 0$. By contrast, with cumulative codes, "losing" a tally symbol results in the loss of a summand in the corresponding ordinal, which immediately leads to smaller Hardy values. □

a) Trim Codes: We introduce a restriction on codes that allows to ensure that $\xrightarrow{r}$ behaves as expected, especially when performing backward computations: a pure code x is *n-trim* if, for any vector v occurring in x, there exists $0 \leq i < k$ s.t. $v[i] \leq n$ and for all $0 \leq j < i$, $v[j] = 0$ and all $i < j < k$, $v[j] \leq n - 1$ (this restricts the ordinal $\beta(v)$). A configuration x, n of $\xrightarrow{r}$ is *trim* if x is n-trim, and a computation $x_0, n_0 \xrightarrow{r} x_1, n_1 \xrightarrow{r} \cdots \xrightarrow{r} x_m, n_m$ is *trim* if every configuration x_i, n_i is trim. Write $x', n \xrightarrow{\text{trim}} x, n$ if $x \leq_* x'$ and x is n-trim (and thus pure) and call *trimming* the transformation from x', n to x, n (nondeterministic but always possible, e.g. by decreasing vector values in excess, or removing vectors). In particular, $\xrightarrow{\text{trim}} \subseteq \geq_*$ where we let $x, n \leq_* x', n' \stackrel{\text{def}}{\Leftrightarrow} x \leq_* x'$ and $n \leq n'$.

Trimness allows us to focus on particular computations of $\xrightarrow{r}$:

Lemma 15. *If x is n-trim, then there exists a trim computation $x, n \xrightarrow{r}{}^* \varepsilon, H^{\pi(x)}(n)$.*

As our initial code x_0 defined in Example 10 is k-trim, it suffices in the following to consider trim computations, i.e. forward computations in $\xrightarrow{rt}$ or backward computations in $\xrightarrow{rt}{}^{-1}$, where $x, n \xrightarrow{rt} y, m \stackrel{\text{def}}{\Leftrightarrow} x, n \xrightarrow{r} y, m$ and x, n and y, m are trim. (In other words, $\xrightarrow{rt} = \xrightarrow{r} \cap \{x, n \mid x, n \text{ is trim}\}^2$).

The next proposition states the key monotonicity property of trim computations:

Proposition 16 (Robustness)**.** *Let x, x' be pure codes and $n' > 0$. If x' is n'-trim and $x, n \leq_* x', n'$, then $H^{\pi(x)}(n) \leq H^{\pi(x')}(n')$.*

b) Weak Implementations: The efforts put into defining a robust computation for the Hardy functions pay when one tries to implement them in a "weak" model like PDNs, as we do in Section VII—but this could also be used in other models. By a weak implementation, we mean—as usual in the Petri net literature—an implementation that guarantees

1) *completeness*: it includes the desired behaviour, and
2) *safety*: it might also yield "smaller" results.

In the case at hand, we provide sufficient conditions (see Definition 18) for two relations $\xrightarrow{d}$ and $\xrightarrow{b}$ on configurations to be called *weakenings* of $\xrightarrow{r}$ and $\xrightarrow{r}{}^{-1}$. The conditions will be easy to check on the actual implementation by PDNs of Section VII, and they entail:

Theorem 17 (Weak Implementations)**.** *If $\xrightarrow{d}$ is a weakening of $\xrightarrow{r}$ and $\xrightarrow{b}$ a weakening of $\xrightarrow{r}{}^{-1}$, then*

1) *For any n_0-trim x_0, $x_0, n_0 \xrightarrow{d}{}^* \varepsilon, H^{\pi(x_0)}(n_0)$ and $\varepsilon, H^{\pi(x_0)}(n_0) \xrightarrow{b}{}^* x_0, n_0$.*
2) *If x_0 is n_0-trim and $x_0, n_0 \xrightarrow{d}{}^* \varepsilon, n$ then $n \leq H^{\pi(x_0)}(n_0)$, and if $\varepsilon, m \xrightarrow{b}{}^* x, n$, then $H^{\pi(x)}(n) \leq m$.*

Note that these are exactly the two properties required in the main proof of Section V from the PDNs $\mathcal{N}_H[k]$ and $\mathcal{N}_{H^{-1}}[k]$.

Here are our sufficient conditions:

Definition 18 (Weakenings)**.** A relation $\xrightarrow{d}$ on codes is a *weakening* of $\xrightarrow{r}$ if $\xrightarrow{rt} \subseteq \xrightarrow{d} \subseteq \geq_*; \xrightarrow{\text{trim}}; \xrightarrow{rt}; \geq_*; \xrightarrow{\text{trim}}$. Similarly, a relation $\xrightarrow{b}$ is a *weakening* of $\xrightarrow{r}{}^{-1}$ if $\xrightarrow{rt}{}^{-1} \subseteq \xrightarrow{b} \subseteq \geq_*; \xrightarrow{\text{trim}}; \xrightarrow{rt}{}^{-1}; \geq_*; \xrightarrow{\text{trim}}$.

Proof of Theorem 17: For (1), by Lemma 15, for an n_0-trim x_0, $x_0, n_0 \xrightarrow{r}{}^* \boldsymbol{\varepsilon}, H^{\pi(x_0)}(n)$ implies $x_0, n_0 \xrightarrow{rt}{}^* \boldsymbol{\varepsilon}, H^{\pi(x_0)}(n)$ and $\boldsymbol{\varepsilon}, H^{\pi(x_0)}(n) \ (\xrightarrow{rt}{}^{-1})^* \ x_0, n_0$ since $\boldsymbol{\varepsilon}$ is $\big(H^{\pi(x_0)}(n)\big)$-trim.

For (2), we reconstruct step by step pieces of a computation of $\xrightarrow{rt}$ or $\xrightarrow{rt}{}^{-1}$. For $\xrightarrow{d}$, if x, n is trim and $x, n \geq_* x', n' \xrightarrow{\text{trim}} x'', n'' \xrightarrow{rt} y'', m'' \geq_* y', m' \xrightarrow{\text{trim}} y, m$, then

$$\begin{aligned} H^{\pi(x)}(n) &\geq H^{\pi(x'')}(n'') && \text{(by Prop. 16)} \\ &= H^{\pi(y'')}(m'') && \text{(by Prop. 12)} \\ &\geq H^{\pi(y)}(m)\,, && \text{(by Prop. 16)} \end{aligned}$$

from which a simple induction yields the result.

Similarly for $\xrightarrow{b}$, if y, m is trim and $y, m \geq_* y', m' \xrightarrow{\text{trim}} y'', m'' \xrightarrow{rt}{}^{-1} x'', n'' \geq_* x', n' \xrightarrow{\text{trim}} x, n$, then

$$\begin{aligned} H^{\pi(y)}(m) &\geq H^{\pi(y'')}(m'') && \text{(by Prop. 16)} \\ &= H^{\pi(x'')}(n'') && \text{(by Prop. 12)} \\ &\geq H^{\pi(x)}(n)\,, && \text{(by Prop. 16)} \end{aligned}$$

and we proceed again by induction. ∎

VII. Petri Data Net Implementation

We explain in this section how to construct $\mathcal{N}_H$ and $\mathcal{N}_{H^{-1}}$, the PDNs that we announced and used in Section V. They transform pairs x, n via a relation $\xrightarrow{d}$ (or $\xrightarrow{b}$ for $\mathcal{N}_{H^{-1}}$) that is a weakening of $\xrightarrow{r}$ (resp., of $\xrightarrow{r}{}^{-1}$) so that Theorem 17 is a proof of Lemma 5. We have to explain how to represent pairs x, n in a PDN, how to transform them correctly, and to engineer definitions for $\xrightarrow{d}$ and $\xrightarrow{b}$ that are both simple enough for PDN implementability, but rigorous and complete enough to fulfil the requirements of Definition 18. One can loosely describe $\xrightarrow{d}$ and $\xrightarrow{b}$ as "trying to perform $\xrightarrow{r}$ or $\xrightarrow{r}{}^{-1}$ on codes, tolerating decreases (wrt $\leq_*$) on x and n, all the while trimming x regularly because Definition 18 requires it." The PDNs are highly nondeterministic (unlike $\xrightarrow{r}$) and may deadlock, but this is not a concern.

What makes PDNs relatively powerful is that they can make weak copies of a counter and even of a sequence, and they can use these weak copies for bounding the number of times a loop is executed ("weak control"). We designed codes and robustness precisely to fit this weak computational power.

In the rest of this section we explain how codes are represented in a PDN (Section VII-A) and how to perform trimming. Due to lack of space, the definitions and the implementation of $\xrightarrow{d}$ and $\xrightarrow{b}$ can be found in the full version of the paper, but all the main implementation ideas are already present with the trimming process.

A. Encoding Configurations of $\xrightarrow{r}$ in a PDN

The weak implementation of Hardy computations has to maintain a PDN representation of a code/counter pair x, n.

1) Counter: The counter n is represented via two places `cpt` and `cpt.id`. Place `cpt.id` is an identity place for relevant tokens: the *current value of the counter* will be the number of tokens in `cpt` whose identity match `cpt.id`.

2) Code: For a code x of length l, distinct identities $I_1 < \ldots < I_l$ identify each item in x. Every item of the code is identified by a unique identity, and the ordering of identities lets one recover the code. All the identities that have been used for items of the current *and past* codes are stored in two places, `vect` and `tally`, letting one distinguish between vector and # occurrences in x; note that each # occurrence has a different identity. The representation of a vector v identified by some I in a code is done via places $\mathtt{c}_0, \ldots, \mathtt{c}_{k-1}$: $v[i]$ is the number of tokens in $\mathtt{c}_i$ with identity I.

Identities evolve during a computation. In order to prevent tokens with now irrelevant identities from disturbing the computation, $\mathcal{N}_H$ uses two identity places, `low` and `high`. We make sure that at any time each of these two places contains a single token, and we just write `low` or `high` to denote the identity carried by that single token. Initially, one has `cpt.id` $<$ `low` $<$ `high` and the identities $I_1, \ldots, I_l$ for the (current) code are exactly those with `low` $< I <$ `high`; other identities are irrelevant for x.

When simulating an $\xrightarrow{r}$ step (and except in simple cases), `cpt.id` is decreased, `high` is increased and `low` is set to the previous value of `high`. Thus tokens with (now) irrelevant identities will never match the current value of `cpt.id` nor belong to the open interval (`low`, `high`).

B. Counter Duplication

In most cases, $\xrightarrow{r}$ requires that we iterate some operation at most n times (or $n-1$, or ...) where n is the current value of the counter. In $\mathcal{N}_H$ this is systematically done in a modular way by first *duplicating the counter* and then *consuming the tokens of the duplicate*, thus controlling the number of iterations.

For this, $\mathcal{N}_H$ uses two places, `dpt` and `dpt.id`, where it stores duplicates of the tokens in `cpt` and `cpt.id`. The net of Fig. 2 depicts the duplication.[2] Transition $\mathtt{dp}_1$ performs the identity updating: `cpt.id` acquires a smaller identity $\mathtt{C}' < \mathtt{C}$ while `dpt.id` is updated with the previous identity of `cpt.id`, namely `C`. Then transition $\mathtt{dp}_2$ transfers the tokens of `cpt` (corresponding to the previous identity of the counter `D`) both to the original counter and to the duplicated counter. Transition

[2] We rely on the standard graphical depiction of enriched nets and use (pictures of) Petri nets where arcs connected to a transition t are labelled with bags of variables that must be instantiated by ordered identities. The number of these variables is exactly $|cons(t)|$ and the ordering of the corresponding identities is carried by the transition. For concision and readability, it is convenient to allow orderings of the variables that are not total: this stands for all possible linearizations. We also use graphical conventions for better readability: *control* places containing black tokens are greyed or filled somehow, *identity* places containing at most one token per identity are represented by simple circles, and the other places, used for counters or general storage, are represented by double circles.

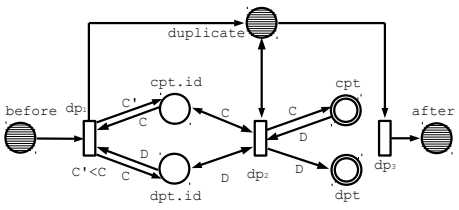


Fig. 2. Duplication of the counter value.

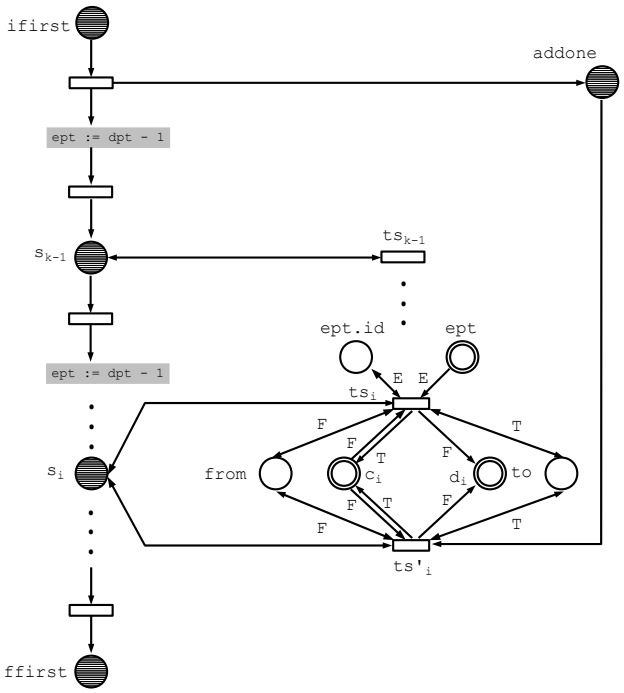


Fig. 3. Copying the first vector (case i).

$\mathtt{dp}_3$ stops the process, and is slightly modified if we need to put $n-1$ rather than n in `dpt`. (In order to avoid a special case for the first duplication, the initial marking has `dpt` empty and `dpt.id` with the same identity as `cpt.id`.)

This simple mechanism must be refined for the loops in the trimming process (see below) where the value of n is used to control that every component of a vector in the code is $\leq$ (some value related to) n. Here one cannot just iterate the previous mechanism: since every duplication possibly decreases n and could violate the property already established for previous components. A more elaborate implementation is required: $\mathcal{N}_H$ uses a second auxiliary counter `ept` and `ept.id` (initialised using `dpt` and `dpt.id`) for such multiple controls (as in Fig. 3). In order to avoid a clash of identities for counters, at every initialisation of `ept`, the new identity of `dpt.id`, namely $\mathtt{D}'$ is selected by the guard $\mathtt{C} < \mathtt{D}' < \mathtt{D}$ where `D` and `C` are the current identities of `dpt.id` and `cpt`.

C. Weak Trimming

During most weak rule applications, a trimming is performed *on-the-fly* while the exact rule is simulated, i.e. we actually weakly implement $\xrightarrow{rt}$ and its inverse. This trimming consists in implementing $\geq_*; \xrightarrow{\text{trim}}$ during the selection and copy of the rule left-hand side and is simultaneously ensured from the rule right-hand side: it turns a configuration t, n into another one $t', n' \leq_* t, n$ which is trim and pure.

- $\mathcal{N}_H$ first duplicates the counter `cpt`, yielding a new value $n' \leq n$ (see Fig. 2). Below we assume that this stage is already passed.
- $\mathcal{N}_H$ scans (in increasing order) relevant identities (the ones in `vect` or `sharp`, between `low` and `high`), purifies the code and copies it beyond `high` as we explain.
- It purifies, one at a time, sequences separated by #s.
- When copying a vector sequence, the first vector is copied but also duplicated in auxiliary places $\mathtt{d}_0, \ldots, \mathtt{d}_{k-1}$ interpreted as the $\mathtt{c}_i$'s. The remaining vectors are also copied and duplicated. Purity is enforced by checking that any (copy of a) vector is lexicographically below the previous vector, as stored in $\mathtt{d}_0, \ldots, \mathtt{d}_{k-1}$.
- Finally, both vectors should fulfil the trimming constraint: for some $i < k$, $v[i] \leq n$, and $v[j] < n$ when $j > i$, and $v[j] = 0$ when $j < i$.

1) Controlling Trimming: Let us detail how this is controlled. $\mathcal{N}_H$ uses three additional identity places: `from`, `to` and `with`. The current item's identity is `from`, its copy after trimming has the new identity `to`, and the purification of a vector requires comparisons with the previous vector in the sequence, whose identity is recorded in `with`, letting one select the appropriate tokens in $\mathtt{d}_0, \ldots, \mathtt{d}_{k-1}$. Fig. 4 describes the overall control of this process, started by `beg.pur`, looping, and concluded by `end.pur`. The body of the loop copies one vector sequence followed by a #. If non empty, the sequence has just one vector, or more, requiring two different treatments. For readability, the labeling of the crucial transitions is specified in the lower part of the picture:

- At start, `beg.pur` produces identity tokens in `from` and `to` within the appropriate intervals (wrt. `low` and `high`), guessing the identity of an item to be copied.
- When the treatment ends, `end.pur` updates `low` and `high` to their new value.
- After copying the *first* vector, `efirst` guesses a new identity (to be copied) in `from` and a new (target) identity in `to`, while recording the current identity in `with`.
- After copying a *remaining* vector, `erem` guesses fresh identities `from` and `to`, and updates the recorded identity in `with`.
- `csharp` copies a # symbol, consuming a token in `sharp` with identity `from` and producing a token with identity `to` (while updating `from` and `to` as usual).

Observe that a bad guess in `from` can lead to deadlock but no infinite looping is possible (as required by the proof of Theorem 4).

2) Copy of a Vector Sequence:

a) First Vector: First, in order to guarantee a trim representation, the copy of the first vector non deterministically selects a component i, which is allowed to be less than or equal to n. The rest of the process is depicted in Fig. 3. It

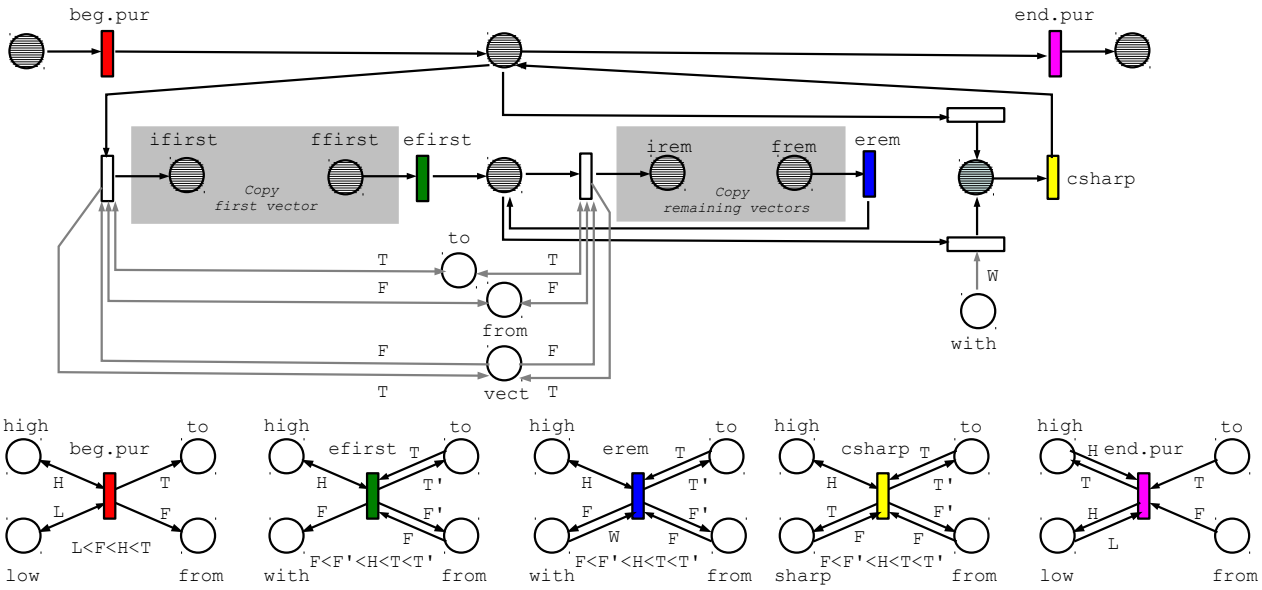


Fig. 4. Control part of the trimming phase.

consists for $j > i$ in:

- setting the auxiliary counter ept to dpt;
- "updating" tokens in place c_j from identity F to identity n, and at the same time in memorising the transferred tokens in place d_j for $j > i$. With the help of the counter ept, at most $n-1$ tokens are transferred. This is performed by transition ts_j.

We then perform the same transfer for component i, but allow one more token thanks to the firing of transition ts'_i. No token is transferred for any component $j < i$.

b) Remaining Vectors: For the sake of readability, we do not represent the management of trimming, which is performed as with the first vector, but rather focus on the purity of the vector sequence.

Let us call v the vector to be copied (identified by variable F), v'' the last vector that has been copied (identified by W) and $v' \leq v$ the vector to be copied (identified by T). In order for v' to be lexicographically smaller than v'' it must satisfy:

- either for all i, $v'(i) \leq v''(i)$
- or there exists some i s.t. for all $j > i$, $v'(i) \leq v''(i)$ and $v'(i) \leq v''(i) - 1$.

Then the simulation non deterministically selects one of these cases. The purity check is thus largely similar to the trimming one: copying is limited by some values, depending on W and cpt.

Observe that one of the possible results of weak trimming is (exact) trimming of the code, and that the other ones are trimmings of a weaker code.

VIII. On Well-Structured Languages

Well-structured transition systems can be seen as language acceptors (or generators). For $\mathbb{M}$ a class of WSTS models, e.g. $\mathbb{M}$ = the Data Nets, let $\mathbb{L}(\mathbb{M})$ be the class $\{L(M) \mid M \in \mathbb{M}\}$ of languages (nondeterministically) accepted by systems in $\mathbb{M}$ when their transitions carry labels, possibly ε, over some alphabet, and when the set of "final", or "accepting", states is *upward-closed*. Geeraerts et al. [17] shows convincingly that this notion of *well-structured languages* (WSL), also called *coverability languages*, is most relevant.

A series of recent papers (see [17, 1, 8] and the references therein) successfully use WSLs as a tool for comparing the descriptive power of varied WSTS models, showing equivalence, e.g. of PDNs and TPNs, or, separating them from the less expressive LCSs (lossy channel systems) or APNs (affine Petri nets [15]).

It turns out that the simulation we develop in this paper (and the matching complexity upper bounds) leads to a (relative but) precise characterisation: Let $L_0 = \{w\#^n \mid n = |w|\}$ collect all words (over a two-letter alphabet) equipped with a length witness: $L_0 \subseteq (a+b)^*\#^*$ is deterministic context-free.

Theorem 19.

1) $L \in \mathbb{L}(\text{PDN})(= \mathbb{L}(\text{TPN}) = \mathbb{L}(\text{DN}))$ *implies* $L \in \bigcup_{k\in\mathbb{N}} \text{Time}(F_{\omega^{\omega^k}}(n))$.
2) $L \in \bigcup_{k\in\mathbb{N}} \text{Time}(F_{\omega^{\omega^k}}(n))$ *implies* $L \cap L_0 \in \mathbb{L}(\text{PDN})$.

The proof relies on the possibility of simulating a space-bounded MM. Using the simulations in [10, 23] and the upper bounds in [13, 22] we derive in a similar way:

Theorem 20. *For any* $L \subseteq L_0$*:*

1) $L \in \bigcup_{k\in\mathbb{N}} \text{Time}(F_{\omega^k}(n))$ *iff* $L \in \mathbb{L}(\text{LCS})$.
2) $L \in \bigcup_{k\in\mathbb{N}} \text{Time}(F_k(n))$ *iff* $L \in \mathbb{L}(\text{APN})$.

This immediately entails separation results like $\mathbb{L}(\text{APN}) \subsetneq \mathbb{L}(\text{LCS}) \subsetneq \mathbb{L}(\text{PDN})(= \mathbb{L}(\text{TPN}) = \mathbb{L}(\text{DN}))$ and the non-collapse of hierarchies like $\{\mathbb{L}(\text{k-PDN})\}_{k\in\mathbb{N}}$, $\{\mathbb{L}(\text{k-LCS})\}_k$ and $\{\mathbb{L}(\text{k-APN})\}_k$ where k-DN, k-APN and k-LCS restrict

to nets with at most k places (resp., to channel systems with a k-letter internal message alphabet). These first separation results are not stronger than those of [1, 8], but they provide a standard measure (using Turing, or equivalently Minsky, machines) rather than a myriad of relative ones.

IX. Concluding Remarks

Theorems 3 and 4 close the open question of the complexity of decision problems over the family of "enriched" nets (our terminology), and have immediate consequences, e.g. for separating various WSTS models according to their computational power. Interestingly, we are not aware of any other natural decision problem sitting exactly at level F_{ω^ω} [16], which makes of enriched net problems the canonical examples for this complexity class.

Our main technical contribution is the robust encoding in $(\mathbb{N}^{k*}, \leq_*)$ of ordinals in Ω_k, together with rewrite rules that describe Hardy computations. Enriched nets are not the only computational model in which these rules can be weakly implemented, and one may use them for proving complexity lower bounds in other settings.

Finally, let us mention two questions raised by this work:

1) Can one improve on Theorem 20? We would prefer an *exact* characterisation of $\mathbb{L}(\mathrm{PDN})$, not relatively to L_0.
2) What about ν-Petri nets [21] and unordered PDNs? The underlying wqo is simpler than $\mathbb{N}^{k*}$, hence we expect lower complexities.

Acknowledgment

Work supported by ANR grant 11-BS02-001-01 and by the Leverhulme Trust. The third author is currently visiting the Computer Science Department at Oxford University.

References

[1] P. A. Abdulla, G. Delzanno, and L. Van Begin, "A classification of the expressive power of well-structured transition systems," *Inform. and Comput.*, vol. 209, pp. 248–279, 2011.

[2] P. Abdulla, P. Mahata, and R. Mayr, "Dense-timed Petri nets: Checking Zenoness, token liveness and boundedness," *Logic. Meth. in Comput. Sci.*, vol. 3, p. 1, 2007.

[3] P. A. Abdulla and G. Delzanno, "On the coverability problem for constrained multiset rewriting," in *AVIS 2006*, 2006.

[4] P. A. Abdulla and A. Nylén, "Timed Petri nets and BQOs," in *Petri Nets 2001*, ser. LNCS, vol. 2075. Springer, 2001, pp. 53–70.

[5] P. A. Abdulla, K. Čerāns, B. Jonsson, and Y.-K. Tsay, "Algorithmic analysis of programs with well quasi-ordered domains," *Inform. and Comput.*, vol. 160, pp. 109–127, 2000.

[6] T. Bolognesi, F. Lucidi, and S. Trigila, "From timed Petri nets to timed LOTOS," in *PSTV '90*. North-Holland, 1990, pp. 395–408.

[7] R. Bonnet, A. Finkel, S. Haddad, and F. Rosa-Velardo, "Comparing Petri Data Nets and Timed Petri Nets," LSV, ENS Cachan, Research Report LSV-10-23, Dec. 2010. [Online]. Available: http://tinyurl.com/82vwcxf

[8] ——, "Ordinal theory for expressiveness of well structured transition systems," in *FoSSaCS 2011*, ser. LNCS, vol. 6604. Springer, 2011, pp. 153–167.

[9] P. Bouyer, S. Haddad, and P.-A. Reynier, "Timed Petri nets and timed automata: On the discriminating power of Zeno sequences," *Inform. and Comput.*, vol. 206, pp. 73–107, 2008.

[10] P. Chambart and Ph. Schnoebelen, "The ordinal recursive complexity of lossy channel systems," in *LICS 2008*. IEEE, 2008, pp. 205–216.

[11] C. Dufourd, A. Finkel, and Ph. Schnoebelen, "Reset nets between decidability and undecidability," in *ICALP '98*, ser. LNCS, vol. 1443. Springer, 1998, pp. 103–115.

[12] M. V. H. Fairtlough and S. S. Wainer, "Ordinal complexity of recursive definitions," *Inform. and Comput.*, vol. 99, pp. 123–153, 1992.

[13] D. Figueira, S. Figueira, S. Schmitz, and Ph. Schnoebelen, "Ackermannian and primitive-recursive bounds with Dickson's Lemma," in *LICS 2011*. IEEE, 2011, pp. 269–278.

[14] A. Finkel and Ph. Schnoebelen, "Well-structured transition systems everywhere!" *Theor. Comput. Sci.*, vol. 256, pp. 63–92, 2001.

[15] A. Finkel, P. McKenzie, and C. Picaronny, "A well-structured framework for analysing Petri nets extensions," *Inform. and Comput.*, vol. 195, pp. 1–29, 2004.

[16] H. M. Friedman, "Some decision problems of enormous complexity," in *LICS 1999*. IEEE, 1999, pp. 2–13.

[17] G. Geeraerts, J.-F. Raskin, and L. V. Begin, "Well-structured languages," *Acta Inf.*, vol. 44, pp. 249–288, 2007.

[18] L. Jacobsen, M. Jacobsen, M. Møller, and J. Srba, "Verification of timed-arc Petri nets," in *SOFSEM 2011*, ser. LNCS, vol. 6543. Springer, 2011, pp. 46–72.

[19] R. Lazić, T. Newcomb, J. Ouaknine, A. Roscoe, and J. Worrell, "Nets with tokens which carry data," *Fund. Inform.*, vol. 88, pp. 251–274, 2008.

[20] M. Löb and S. Wainer, "Hierarchies of number theoretic functions, I," *Arch. Math. Logic*, vol. 13, pp. 39–51, 1970.

[21] F. Rosa-Velardo and D. de Frutos-Escrig, "Decidability and complexity of Petri nets with unordered data," *Theor. Comput. Sci.*, vol. 412, pp. 4439–4451, 2011.

[22] S. Schmitz and Ph. Schnoebelen, "Multiply-recursive bounds with Higman's Lemma," in *ICALP 2011*, ser. LNCS, vol. 6756. Springer, 2011, pp. 441–452, Available: arXiv:1103.4399 [cs.LO].

[23] Ph. Schnoebelen, "Revisiting Ackermann-hardness for lossy counter machines and reset Petri nets," in *MFCS 2010*, ser. LNCS, vol. 6281. Springer, 2010, pp. 616–628.

APPENDIX

A. Subrecursive Hierarchies and Monotonicity

1) Properties of the Hardy Hierarchy: Let us remind a few useful facts about the Hardy hierarchy (see [12] or [22, Appendix C] for details).

The first fact is that each Hardy function is expansive and monotone in its argument n:

Fact 21 (Expansiveness and Monotonicity, see e.g. 22, Lemmata C.9 and C.10)**.** *For all α, α' and $n > 0, m$,*

$$n \leq H^\alpha(n)\,, \tag{18}$$

$$n < m \text{ implies } H^\alpha(n) \leq H^\alpha(m)\,. \tag{19}$$

However, the Hardy functions are not monotone in the ordinal parameter: $H^{n+1}(n) = 2n+1 > 2n = H^n(n) = H^\omega(n)$, though $n+1 < \omega$. We will introduce two ordinal orderings in Section A2 and Section A4 that ensure monotonicity of the Hardy functions.

Another handful fact is that we can decompose Hardy computations:

Fact 22 (see e.g. 22, Lemma C.7)**.** *For all α, γ in Ω, and x,*

$$H^{\gamma \dot{+} \alpha}(x) = H^\gamma(H^\alpha(x))\,. \tag{20}$$

2) Pointwise Ordering: The classical "pointwise at n" ordering used e.g. in [12] and [22, Appendix C] is defined for any $n \in \mathbb{N}$ as the smallest transitive relation $\prec_n$ s.t.

$$\alpha \prec_n \alpha + 1\,, \qquad \lambda_n \prec_n \lambda\,. \tag{21}$$

The inductive definition of $\prec_n$ implies

$$\alpha' \prec_n \alpha \text{ iff } \begin{cases} \alpha = \beta + 1 \text{ is a successor and } \alpha' \preccurlyeq_n \beta, \text{ or} \\ \alpha = \lambda \text{ is a limit and } \alpha' \preccurlyeq_n \lambda_n. \end{cases}$$

This can be understood a "descent" through ordinals, eventually reaching *predecessor* ordinals, which are defined by

$$P_n(\alpha+1) \stackrel{\text{def}}{=} \alpha\,, \qquad P_n(\lambda) \stackrel{\text{def}}{=} P_n(\lambda_n)\,, \tag{22}$$

and indeed

$$0 \preccurlyeq_n \alpha\,, \qquad P_n(\alpha) \prec_n \alpha\,.$$

The interesting observation here is that the ordinals that appear in a Hardy computation $\alpha_0, n \to \alpha_1, n \to \cdots \to \alpha_\ell, n$ where n remains constant, i.e. no successor steps are used, are all related by $\prec_n$: $\alpha_0 \prec_n \alpha_1 \prec_n \cdots \prec_n \alpha_\ell$. The first successor step will occur with $P_n(\alpha_0)+1, n \to P_n(\alpha_0), n+1$.

Obviously $\prec_n$ is a restriction of $<$, the linear ordering of ordinals. For example, $n = \omega_n \prec_n \omega$ but $n+1 \not\prec_n \omega$. The $\prec_n$ relations are linearly ordered themselves, and $<$, can be recovered in view of [see 22, Appendix C.2]:

$$\prec_0 \subset \cdots \subset \prec_n \subset \prec_{n+1} \subset \cdots \subset \Big(\bigcup_{n\in\mathbb{N}} \prec_n\Big) = <\,. \tag{23}$$

Fact 23 (Congruence, see 22, Lemma C.2)**.** *For all α, α', γ and all $n > 0$*

$$\alpha \prec_n \alpha' \text{ implies } \gamma \dot{+} \alpha \prec_n \gamma \dot{+} \alpha'\,, \tag{24}$$

$$\alpha \prec_n \alpha' \text{ implies } \omega^\alpha \prec_n \omega^{\alpha'}\,. \tag{25}$$

Fact 24 (Monotonicity, see e.g. 22, Lemma C.9)**.** *For all α, α' and n, m,*

$$\alpha \prec_n \alpha' \text{ implies } H^\alpha(n) \leq H^{\alpha'}(n)\,. \tag{26}$$

Since $F_\alpha = H^{\omega^\alpha}$, the same statement holds for F using (25).

3) Almost Lean Ordinals: Leanness is a norm on ordinal terms used extensively in [22]: Let n be in $\mathbb{N}$. We say that an ordinal $\alpha < \varepsilon_0$ is *n-lean* if it only uses coefficients $< n$, or, more formally, when it is written under the strict CNF $\alpha = \omega^{\beta_1} \cdot c_1 \dot{+} \cdots \dot{+} \omega^{\beta_m} \cdot c_m$ with $\beta_1 > \cdots > \beta_m$, if we have $c_i < n$ and if, inductively, β_i is also n-lean, this for all $i = 1, \ldots, m$.

Let us introduce a slight variant of n-lean ordinals [see 22, Lemma D.2]: let $\alpha = \omega^{\beta_1} \cdot c_1 \dot{+} \cdots \dot{+} \omega^{\beta_m} \cdot c_m$ be an ordinal in CNF with $\alpha > \beta_1 > \cdots > \beta_m$ and $\omega > c_1, \ldots, c_m > 0$. We say that α is *almost n-lean* if either (i) $c_m \leq n$ and both $\sum_{i<m} \omega^{\beta_i}$ and β_m are n-lean, or (ii) $c_m \leq n$, $\sum_{i<m} \omega^{\beta_i}$ is n-lean, and β_m is almost n-lean. An almost n-lean ordinal is not necessarily n-lean, but an n-lean ordinal is always almost n-lean.

The interest of almost n-leanness is that it is an invariant of the ordinals appearing during forward Hardy computations. As the initial ordinal of (10) is almost k-lean, this property is preserved by perfect computations.

Lemma 25. *If a limit ordinal λ is n-lean, then λ_n is almost n-lean.*

Proof: By induction on λ, letting $\lambda = \omega^{\beta_1} \cdot c_1 \dot{+} \cdots \dot{+} \omega^{\beta_m} \cdot c_m$ as above. If β_m is a successor ordinal $\beta + 1$ (thus β is n-lean), $\lambda_n = \omega^{\beta_1} \cdot c_1 \dot{+} \cdots \dot{+} \omega^{\beta_m} \cdot (c_m - 1) + \omega^\beta \cdot n$ is almost n-lean. If β_m is a limit ordinal, $\lambda_n = \omega^{\beta_1} \cdot c_1 \dot{+} \cdots \dot{+} \omega^{\beta_m} \cdot (c_m - 1) + \omega^{(\beta_m)_n}$ is n-lean by ind. hyp. on β_m. ∎

Lemma 26. *If a successor ordinal $\alpha+1$ is almost n-lean then α is n-lean.*

Proof: If $\alpha + 1 = \omega^{\beta_1} \cdot c_1 \dot{+} \cdots \dot{+} \omega^{\beta_m} \cdot c_m$ as above, it means $\beta_m = 0$, thus we are in case (i) of almost n-lean ordinals with $c_m \leq n$, and $\alpha = \omega^{\beta_1} \cdot c_1 \dot{+} \cdots \dot{+} \omega^{\beta_m} \cdot (c_m - 1)$ is n-lean. The converse implication is immediate. ∎

Lemma 27. *If a limit ordinal λ is almost n-lean then λ_n is almost n-lean.*

Proof: We proceed by induction on λ, letting $\lambda = \omega^{\beta_1} \cdot c_1 \dot{+} \cdots \dot{+} \omega^{\beta_m} \cdot c_m$ as above.

- If β_m is a successor ordinal $\beta + 1$, $\lambda_n = \omega^{\beta_1} \cdot c_1 \dot{+} \cdots \dot{+} \omega^{\beta_m} \cdot (c_m - 1) + \omega^\beta \cdot n$, and either (i) $c_m \leq n$ and β_m is n-lean, and then λ_n also verifies (i), or (ii) $c_m < n$ and $\beta + 1$ is almost n-lean and thus β is n-lean by Lemma 26, and λ_n is again almost n-lean verifying condition (i).
- If β_m is a limit ordinal, then $\lambda_n = \omega^{\beta_1} \cdot c_1 \dot{+} \cdots \dot{+} \omega^{\beta_m} \cdot (c_m - 1) + \omega^{(\beta_m)_n}$. Either (i) $c_m \leq n$ and β_m is n-lean, and by Lemma 25 $(\beta_m)_n$ is almost n-lean and λ_n is almost n-lean by condition (ii), or (ii) $c_m < n$ and β_m is almost n-lean, and by ind.

hyp. $(\beta_m)_n$ is almost n-lean, and λ_n almost n-lean by condition (ii).

■

The following lemma relates leanness, the pointwise ordering, and the linear ordinal ordering; this is a refinement of [22, Lemma B.1], as it handles the almost n-lean case instead of the n-lean one:

Lemma 28. *Let α be almost n-lean. Then $\alpha < \alpha'$ iff $\alpha \prec_n \alpha'$.*

Proof: If $\alpha = 0$, we are done so we assume $\alpha > 0$ and hence $n > 0$, thus $\alpha = \omega^{\beta_1} \cdot c_1 \dot{+} \cdots \dot{+} \omega^{\beta_m} \cdot c_m$ in CNF with $m > 0$.

We prove the claim by induction on α', considering two cases:

1) if $\alpha' = \alpha'' + 1$ is a successor then $\alpha < \alpha'$ implies $\alpha \leq \alpha''$, hence $\alpha \overset{ih}{\preccurlyeq_n} \alpha'' \prec_n \alpha'$.
2) if α' is a limit, we claim that $\alpha \leq \alpha'_n$, from which we deduce $\alpha \overset{ih}{\preccurlyeq_n} \alpha'_n \prec_n \alpha'$. We prove the claim by induction and considering three subcases on α':
   a) if $\alpha' = \omega^\lambda$ with λ a limit, then $\alpha < \alpha'$ implies $\beta_1 < \lambda$, hence $\beta_1 \leq \lambda_n$ by ind. hyp., applicable since β_1 is also almost n-lean. Thus $\alpha \leq \omega^{\lambda_n} = (\omega^\lambda)_n = \alpha'_n$.
   b) if $\alpha' = \omega^{\beta+1}$ then $\alpha < \alpha'$ implies $\beta_1 < \beta + 1$, hence $\beta_1 \leq \beta$. Now, since α is almost n-lean, either
      i) $c_1 = n$ and $m = 1$, hence $\alpha = \omega^{\beta_1} \cdot n \leq \omega^\beta \cdot n = (\omega^{\beta+1})_n = \alpha'_n$, or
      ii) $c_1 < n$, hence $\alpha < \omega^{\beta_1} \cdot n \leq \omega^\beta \cdot n = (\omega^{\beta+1})_n = \alpha'_n$.
   c) if $\alpha' = \gamma \dot{+} \omega^\beta$ with $0 < \gamma, \beta$, then either $\alpha \leq \gamma$, hence $\alpha < \gamma \dot{+} (\omega^\beta)_n = \alpha'_n$, or $\alpha > \gamma$, and then α can be written as $\alpha = \gamma \dot{+} \gamma'$ with $\gamma' < \omega^\beta$. In that case $\gamma' \leq (\omega^\beta)_n$ by ind. hyp., applicable since γ' is also almost n-lean. Thus $\alpha = \gamma \dot{+} \gamma' \leq \gamma \dot{+} (\omega^\beta)_n = (\gamma \dot{+} \omega^\beta)_n = \alpha'_n$.

■

4) Embedding Ordering: We introduce a partial ordering $\sqsubseteq_{\mathrm{o}}$ on ordinals, called *embedding*, and which can be seen as a tree embedding on CNF's that respects layers. Formally, it is defined inductively as

$$\alpha \sqsubseteq_{\mathrm{o}} \beta \overset{\text{def}}{\Leftrightarrow} \begin{cases} \alpha = \omega^{\alpha_1} \dot{+} \cdots \dot{+} \omega^{\alpha_p} \\ \beta = \omega^{\beta_1} \dot{+} \cdots \dot{+} \omega^{\beta_m} \\ \alpha_1 \sqsubseteq_{\mathrm{o}} \beta_{i_1} \wedge \cdots \wedge \alpha_p \sqsubseteq_{\mathrm{o}} \beta_{i_p} \\ \quad \text{for some } i_1 < i_2 < \ldots < i_p \,. \end{cases} \tag{27}$$

Note that $0 \sqsubseteq_{\mathrm{o}} \alpha$ for all α, that $1 \sqsubseteq_{\mathrm{o}} \alpha$ for all $\alpha > 0$. Observe that, in general, $\alpha \not\sqsubseteq_{\mathrm{o}} \omega^\alpha$ and $\lambda_n \not\sqsubseteq_{\mathrm{o}} \lambda$. This ordering is obviously congruent for addition and ω-exponentiation:

$$\alpha \sqsubseteq_{\mathrm{o}} \alpha' \text{ and } \beta \sqsubseteq_{\mathrm{o}} \beta' \text{ imply } \alpha \dot{+} \beta \sqsubseteq_{\mathrm{o}} \alpha' \dot{+} \beta' \,, \tag{28}$$

$$\alpha \sqsubseteq_{\mathrm{o}} \alpha' \text{ implies } \omega^\alpha \sqsubseteq_{\mathrm{o}} \omega^{\alpha'} \,, \tag{29}$$

and could in fact be defined alternatively by the axiom $0 \sqsubseteq_{\mathrm{o}} \alpha$ and the two deduction rules (28) and (29).

When considering the encoding of small ordinals described in Section VI-A-*1)*, the following holds:

$$v \leq v' \text{ implies } \beta(v) \sqsubseteq_{\mathrm{o}} \beta(v') \,. \tag{30}$$

The reciprocal of (30) does not hold in general, e.g. $\beta(\mathbf{1}_1) = \omega \sqsubseteq_{\mathrm{o}} \omega^2 = \beta(\mathbf{1}_2)$ while $\mathbf{1}_1 \not\leq \mathbf{1}_2$.

We list a few useful consequences of the definition of $\sqsubseteq_{\mathrm{o}}$:

$$\alpha \sqsubseteq_{\mathrm{o}} \gamma \dot{+} \omega^\beta \text{ implies } \alpha \sqsubseteq_{\mathrm{o}} \gamma \text{ or } \alpha = \gamma' \dot{+} \omega^{\beta'} \text{ with } \gamma' \sqsubseteq_{\mathrm{o}} \gamma \text{ and } \beta' \sqsubseteq_{\mathrm{o}} \beta \,, \tag{31}$$

$$n \leq m \text{ implies } \lambda_n \sqsubseteq_{\mathrm{o}} \lambda_m \,, \tag{32}$$

$$\alpha \sqsubseteq_{\mathrm{o}} \lambda \text{ implies } \alpha \sqsubseteq_{\mathrm{o}} \lambda_n \text{ or } \alpha \text{ is a limit and } \alpha_n \sqsubseteq_{\mathrm{o}} \lambda_n \,. \tag{33}$$

Proof: (31): Intuitively, there are two cases when we consider an embedding $\alpha \sqsubseteq_{\mathrm{o}} \alpha' = \gamma \dot{+} \omega^\beta$: either the ω^β summand of α' is in the range of the embedding or not. If it is not, then already $\alpha \sqsubseteq_{\mathrm{o}} \gamma$. If it is, then α must be some $\gamma' \dot{+} \omega^{\beta'}$ and $\omega^{\beta'} \sqsubseteq_{\mathrm{o}} \omega^\beta$.
(32): By induction on λ: indeed if $\lambda = \gamma \dot{+} \omega^{\beta+1}$ then $\lambda_m = \gamma \dot{+} \omega^\beta \cdot m$ by (2) which is $\lambda_n \dot{+} \omega^\beta \cdot (m-n)$. If $\lambda = \gamma \dot{+} \omega^\mu$, the i.h. gives $\mu_n \sqsubseteq_{\mathrm{o}} \mu_m$, hence $\lambda_n = \gamma \dot{+} \omega^{\mu_n} \sqsubseteq_{\mathrm{o}} \gamma \dot{+} \omega^{\mu_m} = \lambda_m$.
(33): By induction on λ. λ is some $\gamma \dot{+} \omega^\beta$ with $\beta > 0$ so that $\lambda_n = \gamma \dot{+} (\omega^\beta)_n$. If $\alpha \sqsubseteq_{\mathrm{o}} \gamma$, then $\alpha \sqsubseteq_{\mathrm{o}} \lambda_n$ trivially. If $\alpha = \gamma' \dot{+} 1$ is a successor, $1 \sqsubseteq_{\mathrm{o}} (\omega^\beta)_n$ and again $\alpha \sqsubseteq_{\mathrm{o}} \lambda_n$. There remains the case where $\alpha = \gamma' \dot{+} \omega^{\beta'}$ is a limit (i.e. $\beta' > 0$) with $\gamma' \sqsubseteq_{\mathrm{o}} \gamma$ and $\beta' \sqsubseteq_{\mathrm{o}} \beta$. If β is a limit, then by i.h. either $\beta' \sqsubseteq_{\mathrm{o}} \beta_n$ and hence $\alpha \sqsubseteq_{\mathrm{o}} \lambda_n$, or β' is a limit and $\beta'_n \sqsubseteq_{\mathrm{o}} \beta_n$, hence $\alpha_n \sqsubseteq_{\mathrm{o}} \lambda_n$. Finally, if $\beta = \delta + 1$ is a successor, then either $\beta' \sqsubseteq_{\mathrm{o}} \delta$ so that $\alpha \sqsubseteq_{\mathrm{o}} \gamma + \omega^\delta \sqsubseteq_{\mathrm{o}} \gamma + \omega^\delta \cdot n = \lambda_n$, otherwise by (31), β' is a successor $\delta' + 1$ with $\delta' \sqsubseteq_{\mathrm{o}} \delta$, and then $(\omega^{\beta'})_n = \omega^{\delta'} \cdot n \sqsubseteq_{\mathrm{o}} \omega^\delta \cdot n = (\omega^\beta)_n$, hence $\alpha_n \sqsubseteq_{\mathrm{o}} \lambda_n$.

■

Proposition 29 (Monotonicity)**.**

$$\alpha \sqsubseteq_o \beta \textit{ and } 0 < n \leq m \textit{ imply } F_\alpha(n) \leq F_\beta(m) \,, \tag{34}$$

$$\alpha \sqsubseteq_o \alpha' \textit{ implies } H^\alpha(n) \leq H^{\alpha'}(n) \,. \tag{35}$$

Proof of (34)*:* We prove (34) by induction on β. There are three cases:
1. If $\beta = 0$ then $\alpha \sqsubseteq_{\mathrm{o}} \beta$ implies $\alpha = 0$ and we are done.
2. If $\beta = \lambda$ is a limit, then by (33) either $\alpha \sqsubseteq_{\mathrm{o}} \lambda_n$ or α is a limit and $\alpha_n \sqsubseteq_{\mathrm{o}} \lambda_n$. In the first case $F_\alpha(n) \leq F_{\lambda_n}(m)$ by i.h., in the second case $F_\alpha(n) = F_{\alpha_n}(n) \leq F_{\lambda_n}(m)$, again by i.h. Now (32) and the i.h. entail $F_{\lambda_n}(m) \leq F_{\lambda_m}(m) = F_\lambda(m)$ and we are done.
3. If $\beta = \beta' \dot{+} 1$ is a successor, then by (31) either $\alpha \sqsubseteq_{\mathrm{o}} \beta'$, or $\alpha = \alpha' \dot{+} 1$ with $\alpha' \sqsubseteq_{\mathrm{o}} \beta'$.

In the first case, $F_\alpha(n) \leq F_{\beta'}(m)$ (by i.h.) $\leq F^m_{\beta'}(m)$ (by expansiveness) $= F_\beta(m)$.

In the second case, $F_\alpha(n) = F_{\alpha' \dot{+} 1}(n) = F^n_{\alpha'}(n)$. Now, since $\alpha' \sqsubseteq_{\mathrm{o}} \beta'$, the i.h. gives $F^k_{\alpha'}(n) \leq F^k_{\beta'}(m)$ for all $k \in \mathbb{N}$ (by ind. on k). In particular $F^n_{\alpha'}(n) \leq F^n_{\beta'}(m) \leq F^m_{\beta'}(m)$ (by expansiveness) $= F_\beta(n)$. ■

Proof of (35)*:* Let us proceed by induction on a proof of $\alpha \sqsubseteq_{\mathrm{o}} \alpha'$, based on the deduction rules (28) and (29). For the base case, $0 \sqsubseteq_{\mathrm{o}} \alpha'$ implies $H^0(n) = n \leq H^{\alpha'}(n)$ by expansiveness. For inductive step with (28), if $\alpha \sqsubseteq_{\mathrm{o}} \alpha'$ and $\beta \sqsubseteq_{\mathrm{o}} \beta'$, then

$$\begin{aligned} H^{\alpha\dot{+}\beta}(n) &= H^{\alpha}\big(H^{\beta}(n)\big) && \text{(by (20))} \\ &\leq H^{\alpha}\big(H^{\beta'}(n)\big) && \text{(by ind. hyp. and (19))} \\ &\leq H^{\alpha'}\big(H^{\beta'}(n)\big) && \text{(by ind. hyp.)} \\ &= H^{\alpha'\dot{+}\beta'}(n)\,. && \text{(by (20))} \end{aligned}$$

For the inductive step with (29), $H^{\omega^\alpha}(n) = F_\alpha(n) \leq F_{\alpha'}(n) = H^{\omega^{\alpha'}}(n)$ by (34). ∎

B. Monotonicity for Codes

This Appendix details the proof of Proposition 16.

1) Atomic Losses: Let us first investigate a few properties of $\leq_*$ over pure codes. Write $x \leq^1_* x'$ when $x \leq_* x'$ and the difference between two pure codes x and x' is in some sense "minimal".[3] Formally, the relation is defined by three axioms:

$$x_1x_2 \leq^1_* x_1\#x_2 \quad x_1x_2 \leq^1_* x_1vx_2 \quad x_1vx_2 \leq^1_* x_1(v+\mathbf{1}_j)x_2 \tag{36}$$

with $0 \leq j < k$.

It is plain that $\leq_*$ is the reflexive and transitive closure of $\leq^1_*$. The following lemma allows reducing Proposition 16 to the simpler case $x \leq^1_* x'$:

Lemma 30. *If $x \leq_* x'$ are two pure codes, then there exists $x = x_0 \leq^1_* x_1 \leq^1_* x_2 \leq^1_* \cdots \leq^1_* x_\ell = x'$ where the x_i's are pure.*

Proof idea: We explain how to build the sequence of intermediary x_i's in three steps.

1) One starts with x and adds all missing $\#$ symbols one by one: this maintains purity.
2) One then adds vectors in place where they are missing. In order to maintain purity, an empty position is filled by duplicating the vector immediately to the right of the empty slot (or add $\mathbf{0}$ if there is a $\#$ to the right). Any such addition maintains purity.
   For example, assume
   $$x = \ldots\#v_1v_3v_6\#\ldots \text{ and } x' = \ldots\#v'_1v'_2v'_3v'_4v'_5v'_6v'_7\#\ldots$$
   with $v_i \leq v'_i$ for $i \in \{1,3,6\}$. Then x can be filled (in 4 steps) with
   $$x = \ldots\#v_1v_3v_6\#\ldots \leq^1_* \leq^1_* \leq^1_* \leq^1_* \ldots\#v_1v_3v_3v_6v_6v_6\mathbf{0}\#\ldots$$
   If this filling process is done from right to left, every inserted vector is smaller than the corresponding vector in x' (since x' is pure) hence the constructed x_{i+1} remains $\leq_* x'$.
3) We have now reduced the problem to the case where x and x' have same length. It suffices to add enough unit vectors to every v until we reach the corresponding v' in x' whenever $v < v'$. If this is done from left to right, purity is maintained.

∎

2) Code Honesty: We investigate in this section two restrictions on the size of representation of codes during computations. One is an upper bound on the *length* of the code, and is true of any forward or backward computation with $\xrightarrow{r}$. The second, *trimness*, is a restriction on the values that can appear in the vectors of the code: it is guaranteed by our forward computation, but need to be enforced on backward ones; however there exists one "perfect" backward computation that verifies it: it suffices to reverse the forward computation!

a) Length Hierarchy: We define a hierarchy of functions $H_\alpha(n)$ that bounds the length of any pure code x s.t. $H^{\pi(x)}(n) = H^\alpha(n)$. The strategy we adopt is to employ the rules of Definition 11 in reverse from a configuration $(\boldsymbol{\varepsilon}, H^\alpha(n))$, and bound the size of the resulting code. It will turn out that this hierarchy is already known in the literature as the *length hierarchy* [24].

We define accordingly

$$H_0(n) = 0\,, \tag{37}$$

$$H_{\alpha+1}(n) = H_\alpha(n+1) + 1\,, \tag{38}$$

$$H_\lambda(n) = H_{\lambda_n}(n)\,. \tag{39}$$

Observe that, indeed, $\boldsymbol{\varepsilon}$ is of length 0, thus justifying (37); that if x is of length $\leq H_\alpha(n+1)$, then applying rule (R1) in reverse increments this length by 1, thus justifying (38); finally, if x is of length $\leq H_{\lambda_n}(n)$, then applying rule (R2) in reverse either decreases this length, or preserves it in case of a rewrite "$\mathbf{V}v\#x, n \xrightarrow{r} \mathbf{V}(v)_n\#x, n$" with $(v)_n = v - \mathbf{1}_i + n\cdot\mathbf{1}_{i-1}$ for some $i > 0$, justifying (39). By Proposition 12, we deduce:

Lemma 31. *If x is pure and $x, n \xrightarrow{r}{}^* \boldsymbol{\varepsilon}, H^\alpha(n)$, then $|x| \leq H_\alpha(n)$.*

The length hierarchy is closely related to the Hardy hierarchy; in particular [see e.g. 22, Eq. (65)]:

$$H_\alpha(n) = H^\alpha(n) - n\,. \tag{40}$$

An easy observation in the same line as Lemma 31 is that backward rule applications from $\boldsymbol{\varepsilon}, H^\alpha(n)$ cannot increment the values in vectors to more that the total computed value $H^\alpha(n)$. Thus,

Lemma 32. *If x_1vx_2 is pure and $x_1vx_2, n \xrightarrow{r}{}^* \boldsymbol{\varepsilon}, H^\alpha(n)$, then $v[j] < H^\alpha(n)$ for all $0 \leq j < k$.*

b) Almost Lean and Trim Computations: Lemma 32 does not provide us with enough information on the values in vectors for our purposes. Recall the definition of almost n-lean ordinals from Appendix A3. Let us call a pure code x *almost n-lean* if $\pi(x)$ is almost n-lean. By extension, a configuration x, n of $\xrightarrow{r}$ is *almost lean* if x is pure and almost n-lean, and a computation $x_0, n_0 \xrightarrow{r} x_1, n_1 \xrightarrow{r} \cdots \xrightarrow{r} x_m, n_m$ is *almost lean* if every configuration x_i, n_i is almost lean, for every $0 \leq i \leq m$. By Lemmas 25 to 27, we deduce:

[3] It would be minimal for arbitrary codes if the second axiom was reading $x_1x_2 \leq^1_* x_1\mathbf{0}x_2$, but it would not always relate pure codes to pure codes.

Lemma 33. *If $\pi(x)$ is almost n-lean, then there exists an almost lean computation $x, n \xrightarrow{r}{}^{*} \varepsilon, H^{\pi(x)}(n)$.*

Note that our initial code x_0 from (10) is almost k-lean.

However, almost n-leanness is not a robust property of codes: e.g. $|^{\mathbf{2}}_{\mathbf{0}}|^{\mathbf{1}}_{\mathbf{0}}|^{\mathbf{1}}_{\mathbf{0}}|^{\mathbf{1}}_{\mathbf{0}}\#$ encodes $\omega^{\omega^{\omega^2 \dot{+} \omega \cdot 3}}$ and is almost 2-lean, but the smaller code $|^{\mathbf{1}}_{\mathbf{0}}|^{\mathbf{1}}_{\mathbf{0}}|^{\mathbf{1}}_{\mathbf{0}}|^{\mathbf{1}}_{\mathbf{0}}\#$ that encodes $\omega^{\omega^{\omega \cdot 4}}$ is not almost 2-lean. We therefore introduce a slight relaxation of almost leanness: a pure code x is *n-trim* if, for any decomposition $x = x_1 v x_2$, the ordinal $\beta(v)$ is almost n-lean, i.e. there exists $0 \leq i < k$ s.t. $v[i] \leq n$ and for all $0 \leq j < i$, $v[j] = 0$ and all $i < j < k$, $v[j] < n$. By analogy with almost leanness, call a computation *trim* if in every configuration x_i, n_i the code x_i is n_i-trim. Unlike almost leanness, trimness is clearly preserved by $\geq_*$. Interestingly, it is also preserved by direct computations, as shown by Lemma 15, which is a version of Lemma 33 that restricts computations to trim ones instead of almost lean ones:

Proof of Lemma 15: Define a large n-trim ordinal as $\pi(x)$ where x is n-trim. We need to prove that if π' is an n-trim large ordinal and $\pi \prec_n \pi'$, then π is n-trim. This is obvious for $\pi' = \pi + 1$, and we turn now to the different limit cases when $\pi = \pi'_n$. If $\pi = (\gamma \dot{+} \omega^{\alpha+1})_n = \gamma \dot{+} \omega^\alpha \cdot n$, this holds; otherwise $\pi = (\gamma \dot{+} \omega^\alpha)_n = \gamma \dot{+} \omega^{\alpha_n}$ and we only need to prove that the small ordinals in $\alpha_n = (\sum_{i=1}^{p} \omega^{\beta_i})_n$ are almost n-lean, under the hypothesis that each β_i is almost n-lean since π' is n-trim. If $\beta_p = \beta + 1$ is a successor ordinal, then $\alpha_n = \sum_{i=1}^{p-1} \omega^{\beta_i} \dot{+} \omega^\beta \cdot n$, then β is clearly almost n-lean. If β_p is a limit ordinal, then $\alpha_n = \sum_{i=1}^{p-1} \omega^{\beta_i} \dot{+} \omega^{(\beta_p)_n}$ is s.t. β_n is almost n-lean by Lemma 27. Hence π is n-trim in all cases. ∎

Lemma 15 implies that forward computations preserve trimness, but more importantly that we can restrict our backward rule applications to enforce trimness. Such a restriction is required because backward rule applications do not necessarily preserve trimness: for instance with $k = 2$, we can go from a configuration $(0, n)\#, n$ to a configuration $\#(0, n)\#, n - 1$ by applying (R1), and if n is sufficiently large, later to a configuration $(0, n + 1)\#, n'$ for a considerably smaller n'. What Lemma 33 entails is that there is another computation that fits our needs: for this example, applying (R2) backwards on $(0, n)\#, n$ yields $(1, 0)\#, n$ instead.

3) Monotonicity in Presence of Losses: We prove a series of monotonicity results that allow to handle losses in codes. As we work with ordinals of form ω^α, it is more convenient to express these results using $F_\alpha = H^{\omega^\alpha}$.

Lemma 34. *Let α, α', γ be ordinals. If $\gamma + \alpha$ is almost n-lean and $\alpha < \alpha'$, then $\gamma + \alpha \prec_n \gamma + \alpha'$.*

Proof: Write $\gamma = \gamma_1 \dot{+} \gamma_2 \dot{+} \gamma_3$ so that $\gamma + \alpha = \gamma_1 \dot{+} \gamma_2 \dot{+} \alpha$ and $\gamma + \alpha' = \gamma_1 \dot{+} \alpha'$. Now $\gamma_2 \dot{+} \alpha < \alpha'$ and $\gamma_2 \dot{+} \alpha$ is almost n-lean, so that $\gamma_2 \dot{+} \alpha \prec_n \alpha'$ by Lemma 28 and $\gamma + \alpha = \gamma_1 \dot{+} \gamma_2 \dot{+} \alpha \prec_n \gamma_1 \dot{+} \alpha' = \gamma + \alpha$ by (24). ∎

Lemma 35. *If $\gamma \prec_n \gamma'$ then $F_{\gamma+\alpha}(n) \leq F_{\gamma+\alpha'}(n)$.*

Proof: We proceed by induction over α.

For $\alpha = 0$, $\gamma \prec_n \gamma'$ entails $F_\gamma(n) \leq F_{\gamma'}(n)$ by (26).

For $\alpha = \beta + 1$ a successor ordinal, the ind. hyp. and (23) gives $F_{\gamma+\beta}(m) \leq F_{\gamma'+\beta}(m)$ for any $m \geq n$, hence $F^n_{\gamma+\beta}(n) \leq F^n_{\gamma'+\beta}(n)$ by (19) and (18), hence $F_{\gamma+\beta+1}(n) \leq F_{\gamma'+\beta+1}(n)$.

For $\alpha = \lambda$ a limit ordinal, we immediately have $F_{\gamma+\lambda}(n) = F_{\gamma+\lambda_n}(n) \leq F_{\gamma'+\lambda_n}(n) = F_{\gamma'+\lambda}(n)$ by ind. hyp. ∎

We exploit Lemma 34 in the two following lemmata, which match cases 2 and 3 of atomic losses in codes:

Lemma 36. *If γ is n-lean, then $F_{\gamma+\alpha'+\alpha}(n) \leq F_{\gamma+\alpha'+\omega^\beta+\alpha}(n)$.*

Proof: If $\alpha' > 0$, the lemma is trivial, as putting $\gamma = \gamma_1 \dot{+} \gamma_2$ and $\gamma + \alpha' = \gamma_1 \dot{+} \alpha$ shows that $\gamma_1 \dot{+} \alpha' + \alpha \sqsubseteq_{\mathrm{o}} \gamma_1 \dot{+} \alpha' \dot{+} \omega^\beta + \alpha$ and we conclude by (34). Assume therefore $\alpha' = 0$; then Lemma 34 applies to show $\gamma \prec_n \gamma + \omega^\beta$ since γ is almost n-lean and $0 < \omega^\beta$, and applying Lemma 35 yields the result. ∎

Lemma 37. *If $\gamma + \omega^\beta$ is almost n-lean and $\beta \sqsubseteq_o \beta'$, then $F_{\gamma+\alpha'\dot{+}\omega^\beta+\alpha}(n) \leq F_{\gamma+\alpha'\dot{+}\omega^{\beta'}+\alpha}(n)$.*

Proof: As in the previous proof, the case $\alpha' > 0$ is trivial since $\omega^\beta \sqsubseteq_{\mathrm{o}} \omega^{\beta'}$ by (29). Assume therefore $\alpha' = 0$, then $\omega^\beta < \omega^{\beta'}$ yields $\gamma + \omega^\beta \prec_n \gamma + \omega^{\beta'}$ by Lemma 34, and applying Lemma 35 yields the result. ∎

The following proposition together with (19) immediately proves Proposition 16:

Proposition 38. *Let x, x' be pure codes and $n > 0$. If x' is n-trim and $x \leq_* x'$, then $H^{\pi(x)}(n) \leq H^{\pi(x')}(n)$.*

Proof: We proceed by induction on the number of $\leq^1_*$-steps between x and x'. If $x = x'$ the result hold vacuously. Consider therefore for the induction step a pure code x'' with $x'' \leq_* x'$ and $x \leq^1_* x''$; clearly x and x'' are also n-trim. By ind. hyp., $H^{\pi(x'')}(n) \leq H^{\pi(x')}(n)$, and we only need to prove $H^{\pi(x)}(n) \leq H^{\pi(x'')}(n)$.

1) The first axiom is easy to treat: if $x = x_1 x_2$ and $x'' = x_1 \# x_2$, then $\pi(x) \sqsubseteq_{\mathrm{o}} \pi(x'')$ and thus $H^{\pi(x)}(n) \overset{(35)}{\leq} H^{\pi(x'')}(n)$. This simple proof is the main rationale for our cumulative encoding of ordinals.

The next two axioms require more work. In both cases, we decompose x'' into $x_1 \# \mathbf{V}_1 v \mathbf{V}_2 \# x_2$ where v is the particular vector modified by $\leq^1_*$, so that $x = x_1 \# \mathbf{V} \# x_2$ verifies either $\mathbf{V} = \mathbf{V}_1 \mathbf{V}_2$ (in the case of the second axiom) or $\mathbf{V} = \mathbf{V}_1 (v - \mathbf{1}_j) \mathbf{V}_2$ for some $0 \leq j < k$ (in the case of the third axiom). By Lemma 15, there exists a trim computation on x''; its initial phase is of form:

$$x_1 \# \mathbf{V} v \mathbf{V}' \# x_2, n \xrightarrow{r}{}^{*} \mathit{pure}(V(x_1) \mathbf{V}_1 v \mathbf{V}_2 \# x_2), n' \quad (41)$$

which we reach by evaluating x_1 in full, i.e.

$$x_1 \#, n \xrightarrow{r}{}^{*} \varepsilon, n' \,, \quad (42)$$

where we define

$$n' \overset{\text{def}}{=} H^{\pi(x_1\#)}(n) \,. \quad (43)$$

Assume that the following claim holds for all vector sequences $\mathbf{V}'$ and all $r \geq n'$:

$$H^{\omega^{\alpha(V(x_1)\mathbf{V}\mathbf{V}')}}(r) \leq H^{\omega^{\alpha(V(x_1)\mathbf{V}_1 v \mathbf{V}_2 \mathbf{V}')}}(r)\ . \qquad (44)$$

Let further $x_2 \stackrel{\text{def}}{=} \mathbf{V}_3\#\cdots\#\mathbf{V}_m\#$, and consider the prefix corresponding to (41) in the computation on x: it encodes a Hardy computation

$$\begin{aligned}
&H^{\pi(x_1\#\mathbf{V}\#x_2)}(n)\\
&= H^{\pi(V(x_1)\mathbf{V}\mathbf{V}_3\#\cdots\#\mathbf{V}_m\#)}\big(H^{\omega^{\alpha(V(x_1)\mathbf{V})}}(n')\big) \qquad \text{(by (20))}\\
&\leq H^{\pi(V(x_1)\mathbf{V}\mathbf{V}_3\#\cdots\#\mathbf{V}_m\#)}\big(H^{\omega^{\alpha(V(x_1)\mathbf{V}_1 v \mathbf{V}_2)}}(n')\big)\\
&\qquad\qquad \text{(by (44) with } \mathbf{V}' = \boldsymbol{\varepsilon} \text{ and (19))}\\
&= H^{\pi(V(x_1)\mathbf{V}\mathbf{V}_3\mathbf{V}_4\#\cdots\#\mathbf{V}_m\#)}\big(H^{\omega^{\alpha(V(x_1)\mathbf{V}\mathbf{V}_3)}}(n'')\big)\\
&\qquad \text{(by (20) and setting } n'' \stackrel{\text{def}}{=} H^{\omega^{\alpha(V(x_1)\mathbf{V}_1 v \mathbf{V}_2)}}(n'))\\
&\leq H^{\pi(V(x_1)\mathbf{V}\mathbf{V}_3\mathbf{V}_4\#\cdots\#\mathbf{V}_m\#)}\big(H^{\omega^{\alpha(V(x_1)\mathbf{V}_1 v \mathbf{V}_2 \mathbf{V}_3)}}(n'')\big)\\
&\qquad\qquad \text{(by (44) with } \mathbf{V}' = \mathbf{V}_3 \text{ and (19))}\\
&\ \ \vdots\\
&\leq H^{\pi(V(x_1)\mathbf{V}_1 v \mathbf{V}_2 \# x_2)}(n')\\
&= H^{\pi(x'')}(n)\ .
\end{aligned}$$

It only remains to prove the claimed (44). To this end, two comments on the prefix of the computation (41) are in order: by combining Lemma 31 and (40) on the computation (42), we deduce that $|x_1| \leq n' - n < n'$ since $n > 0$, thus $\alpha(V(x_1))$ can be written as $\sum_{i=1}^{p} \omega^{\beta_i}$ with $p < n'$. Using Lemma 32, each β_i, which is encoded by a vector appearing in x_1, is n'-lean, hence:

$$\alpha(V(x_1)) \text{ is } n'\text{-lean.} \qquad (45)$$

In the same way, since the computation is trim, $\beta(v)$ is almost n'-lean. Recall that the CNF of $\alpha(V(x_1))$ has $p < n'$ terms. As it is also n'-lean, adding an almost n'-lean term yields an almost n'-lean term:

$$\alpha(V(x_1)) + \omega^{\beta(v)} \text{ is almost } n'\text{-lean.} \qquad (46)$$

Note that (45) and (46) also hold for all $r \geq n'$.

Turning to the two remaining axioms:

2) For the second axiom, recall that $\mathbf{V} = \mathbf{V}_1\mathbf{V}_2$. Put $\gamma = \alpha(V(x_1))$, $\alpha' = \alpha(\mathbf{V}_1)$, $\beta = \beta(v)$, and $\alpha = \alpha(\mathbf{V}_2\mathbf{V}')$. Then, by (45) and Lemma 36, $H^{\omega^{\gamma+\alpha'+\alpha}}(r) = F_{\gamma+\alpha'+\alpha}(r) \leq F_{\gamma+\alpha'\dot{+}\omega^{\beta}+\alpha}(r) = H^{\omega^{\gamma+\alpha'\dot{+}\omega^{\beta}+\alpha}}(r)$.
3) For the third axiom, recall that $\mathbf{V} = \mathbf{V}_1(v-\mathbf{1}_j)\mathbf{V}_2$. Put $\gamma = \alpha(V(x_1))$, $\alpha' = \alpha(\mathbf{V}_1)$, $\beta = \beta(v-\mathbf{1}_j)$, $\beta' = \beta(v)$, and $\alpha = \alpha(\mathbf{V}_2\mathbf{V}')$. By (30), $\beta \sqsubseteq_{\text{o}} \beta'$, and by (46) and Lemma 37, $H^{\omega^{\gamma+\alpha'\dot{+}\omega^{\beta}+\alpha}}(r) = F_{\gamma+\alpha'\dot{+}\omega^{\beta}+\alpha}(r) \leq F_{\gamma+\alpha'\dot{+}\omega^{\beta'}+\alpha}(r) = H^{\omega^{\gamma+\alpha'\dot{+}\omega^{\beta'}+\alpha}}(r)$.

■

C. Proof of Theorem 20 (Section VIII)

1. If L is $L(\mathcal{N})$ for some PDN $\mathcal{N}$, then the question whether $w \in L$ reduces to a coverability problem for $\mathcal{N}' \stackrel{\text{def}}{=} w \otimes \mathcal{N}$, a PDN obtained by a synchronized product of $\mathcal{N}$ and (an FSA for) w. Since $|\mathcal{N}'| = O(|w|)$ (here $\mathcal{N}$ is a constant of size $O(1)$) and since $\mathcal{N}'$ is k-dimensional when $\mathcal{N}$ is, we have reduced L to coverability for k-PDNs, a problem in $\textsc{Time}(F_{\omega^{\omega^k+O(1)}}(n))$ by Theorem 3.

2. With a Minsky machine (MM) M and some k, the construction in section V associates a PDN that simulates M with space bounded by $F_{\omega^{\omega^k}}(n)$. It is easy to modify the PDN so that it (1) guesses a word w of length n; (2) outputs w while weakly storing $w\#^n$ in the work space of the MM and weakly storing n in `cpt`; (3) generates $F_{\omega^{\omega^k}}(n)$ extra workspace for M and runs it on w; (4) after/if M accepts w, folds back the workspace and reconstructs `cpt`; (5) outputs $\#^{n'}$ where n' is value now stored in `cpt`. As in Section V, the nature of weak computations guarantees $n' \leq n$, and one only has $n' = n$ if the simulation of the MM (and storing w) was perfect. Hence the PDN only outputs words $w\#^n$ s.t. $n = |w|$ and that are accepted by M in space $F_{\omega^{\omega^k}}(n)$ (or s.t. $n < |w|$ and hence that are not in L_0).

D. Weak Implementation of $\xrightarrow{r}$

In Table I, we present the so-called "weak" rules that define $\xrightarrow{d}$, a weakening of both $\xrightarrow{r}$ and $\xrightarrow{r}{}^{-1}$ in the sense of Theorem 17, which proves its weak correctness. Note that a general implicit condition on the rules is that they take and produce trim configurations. Rule 1 corresponds to rule (R1) but we have split the exact rule (R2) of $\xrightarrow{r}$. Indeed in order to correctly define $\xrightarrow{d}$, we need to make explicit the implicit different cases of rule (R2). More precisely, the application of this rule may vary depending on two criteria:

- The type of the vector that is found in front of the first $\#$. Rules 2–5 correspond to the case of the null vector, as in (12). Rules 6–9 correspond to the case of a vector whose first component is non null, which is the first case in (15). Rules 10–13 correspond to the remaining case of (15).
- Inside any group of rules, there are four cases depending on what follows the first $\#$. It can be the empty sequence, a second $\#$ or a vector. This last case is again split into two subcases depending on a lexicographic relation.

This laborious presentation of all the rules has its uses, firstly because writing the weak backward rules and performing trimming on-the-fly without first breaking up the various cases turned out to be an error-prone task, and secondly because there is sufficient variation between the various cases of $\xrightarrow{r}$ and especially of $\xrightarrow{r}{}^{-1}$ to warrant handling them separately in a PDN implementation.

Let us repeat that these weak rules always produce a trim representation. In order to prove the correction of the the weak rules, we prove that they are weakenings as defined in Definition 18.

	Exact Rules	Weak forward rules	Weak backward rules
Rule 1	$\#t, n \xrightarrow{r} t, n+1$	$s\#t, n \xrightarrow{d} t, n+1$	$t, n \xrightarrow{d} \#t', n'+1$ with $n' \leq n$ and $t' \leq_* t$
Rule 2	$\mathbf{V0}\#, n \xrightarrow{r} \mathbf{V}\#^n, n$	$rvs\#t, n \xrightarrow{d} \mathbf{V}\#^{n''}, n'$ with $n'' \leq n' \leq n$ and $\mathbf{V} \leq_* r$	$r\#t_0\# \dots \#t_{n'}, n \xrightarrow{d} \mathbf{V0}\#, n'$ with $n' \leq n$ and $\mathbf{V} \leq_* r$
Rule 3	$\mathbf{V0}\#\#t, n \xrightarrow{r} \mathbf{V}\#^n\mathbf{0}\#t, n$	$rvs\#s'\#t, n \xrightarrow{d} \mathbf{V}\#^{n''}\mathbf{0}\#t', n'$ with $n'' \leq n' \leq n$, $\mathbf{V} \leq_* r$ and $t' \leq_* t$	$r\#t_0\# \dots \#t_{n'}vs\#t, n \xrightarrow{d} \mathbf{V0}\#\#t', n'$ with $n' \leq n$, $\mathbf{V} \leq_* r$ and $t' \leq_* t$
Rule 4	$\mathbf{V0}\#\mathbf{0}t, n \xrightarrow{r} \mathbf{V}\#^n\mathbf{00}t, n$	$rvs\#s'wt, n \xrightarrow{d} \mathbf{V}\#^{n''}\mathbf{00}t', n'$ with $n'' \leq n' \leq n$, $\mathbf{V} \leq_* r$ and $t' \leq_* t$	$r\#t_0\# \dots \#t_{n'}vsv't, n \xrightarrow{d} \mathbf{V0}\#\mathbf{0}t', n'$ with $n' \leq n$, $\mathbf{V} \leq_* r$ and $t' \leq_* t$
Rule 5	$\mathbf{V0}\#wt, n \xrightarrow{r} \mathbf{V}\#^n wt, n$ with $w > 0$	$rvs\#s'wt, n \xrightarrow{d} \mathbf{V}\#^{n''}w't', n'$ with $n'' \leq n' \leq n$, $\mathbf{V} \leq_* r$, $t' \leq_* t$ and $\mathbf{0} < w' \leq w$	$r\#t_0\# \dots \#t_{n'}wt, n \xrightarrow{d} \mathbf{V0}\#w't', n'$ with $n' \leq n$, $\mathbf{V} \leq_* r$ and $t' \leq_* t$ and $\mathbf{0} < w' \leq w$
Rule 6	$\mathbf{V}v\#, n \xrightarrow{r} \mathbf{V}(v-\mathbf{1}_0)^n\#, n$ with $\mathbf{1}_0 \leq v$	$rvs\#t, n \xrightarrow{d} \mathbf{V}v_0 \dots v_{n''}\#, n'$ with $n'' \leq n' \leq n$, $\mathbf{V} \leq_* r$, $\mathbf{1}_0 \leq v$ and $v_0 \leq (v-\mathbf{1}_0)$	$rv_0t_0 \dots t_{n'-1}v_{n'}t, n \xrightarrow{d} \mathbf{V}(v'+\mathbf{1}_0)\#, n'$ with $n' \leq n$, $\mathbf{V} \leq_* r$ and $v' \leq \min(v_i)$
Rule 7	$\mathbf{V}v\#\#t, n \xrightarrow{r} \mathbf{V}(v-\mathbf{1}_0)^n\#v\#t, n$ with $\mathbf{1}_0 \leq v$	$rvs\#s'\#t, n \xrightarrow{d} \mathbf{V}v_0 \dots v_{n''}\#v_{n''}+\mathbf{1}_0\#t', n'$ with $n'' \leq n' \leq n$, $\mathbf{V} \leq_* r$, $t' \leq_* t$ $\mathbf{1}_0 \leq v$, $v_0 \leq (v-\mathbf{1}_0)$ and $v_{n''} \leq v_{n''}$	$rv_0t_0 \dots t_{n'-1}v_{n'}t_{n'}\#sv_{n'}+\mathbf{1}_0s'\#t, n$ $\xrightarrow{d}\mathbf{V}(v'+\mathbf{1}_0)\#\#t', n'$ with $n' \leq n$, $\mathbf{V} \leq_* r$ and $t' \leq_* t$ and $v' \leq \min(v_i)$
Rule 8	$\mathbf{V}v\#wt, n \xrightarrow{r} \mathbf{V}(v-\mathbf{1}_0)^n\#vwt, n$ with $\mathbf{1}_0 \leq v$ and $w \leq_{\text{lex}} v$	$rvs\#s'wt, n \xrightarrow{d} \mathbf{V}v_0 \dots v_{n''}\#v_{n''}+\mathbf{1}_0w't', n'$ with $n'' \leq n' \leq n$, $\mathbf{V} \leq_* r$ and $t' \leq_* t$ $\mathbf{1}_0 \leq v$, $v_0 \leq (v-\mathbf{1}_0)$ and $v_{n''} \leq v_{n''}$	$rv_0t_0 \dots t_{n'-1}v_{n'}t_{n'}\#sv_{n'}+\mathbf{1}_0s'wt, n$ $\xrightarrow{d}\mathbf{V}(v'+\mathbf{1}_0)\#w't', n'$ with $n' \leq n$, $\mathbf{V} \leq_* r$ and $t' \leq_* t$ $v' \leq \min(v_i)$, $w' \leq w$ and $w' \leq_{\text{lex}} v'+\mathbf{1}_0$
Rule 9	$\mathbf{V}v\#wt, n \xrightarrow{r} \mathbf{V}(v-\mathbf{1}_0)^n\#wt, n$ with $\mathbf{1}_0 \leq v$ and $v <_{\text{lex}} w$	$rvs\#s'wt, n \xrightarrow{d} \mathbf{V}v_0 \dots v_{n''}\#w't', n'$ with $n'' \leq n' \leq n$, $\mathbf{V} \leq_* r$, $t' \leq_* t$ $\mathbf{1}_0 \leq v$, $v_0 \leq (v-\mathbf{1}_0)$ and $v_0+\mathbf{1}_0 <_{\text{lex}} w'$	$rv_0t_0 \dots t_{n'-1}v_{n'}t_{n'}\#swt, n$ $\xrightarrow{d}\mathbf{V}(v'+\mathbf{1}_0)\#w't', n'$ with $n' \leq n$, $\mathbf{V} \leq_* r$ and $t' \leq_* t$ $v' \leq \min(v_i)$, $w' \leq w$ and $v'+\mathbf{1}_0 <_{\text{lex}} w$
Rule 10	$\mathbf{V}v\#, n \xrightarrow{r} \mathbf{V}v''\#, n$ $\exists i > 0\ v(i) > 0 \wedge \forall j < i\ v(j) = 0$, $\forall j \notin \{i-1, i\}\ v''(j) = v(j)$, $v''(i) = v(i) - 1$ and $v''(i-1) = n$	$rvs\#t, n \xrightarrow{r} \mathbf{V}v''\#, n'$ $\mathbf{V} \leq_* r$, $\exists v' \leq v\ \mathbf{V}v'$ pure $\exists i > 0\ v'(i) > 0 \wedge \forall j < i\ v'(j) = 0$ $v'' \leq v' - \mathbf{1}_i + (n')\mathbf{1}_{i-1}$	$rvs\#t, n \xrightarrow{r} \mathbf{V}(v''+\mathbf{1}_i)\#, n'$ $i > 0$, $\forall j < i\ v''(j) = 0$, $v(i-1) \geq n'$ $\forall j > i\ v''(j) \leq v(i)$ $\mathbf{V} \leq_* r$
Rule 11	$\mathbf{V}v\#\#t, n \xrightarrow{r} \mathbf{V}v''\#v\#t, n$ $\exists i > 0\ v(i) > 0 \wedge \forall j < i\ v(j) = 0$, $\forall j \notin \{i-1, i\}\ v''(j) = v(j)$, $v''(i) = v(i) - 1$ and $v''(i-1) = n$	$rvs\#s'\#t, n \xrightarrow{r} \mathbf{V}v''\#v'\#, n'$ $\mathbf{V} \leq_* r$, $v' \leq v\ \mathbf{V}v'$ pure $\exists i > 0\ v'(i) > 0 \wedge \forall j < i\ v'(j) = 0$ $v'' \leq v' - \mathbf{1}_i + (n')\mathbf{1}_{i-1}$	$rvs\#s'us''\#t, n \xrightarrow{r} \mathbf{V}(v''+\mathbf{1}_i)\#\#, n'$ $\exists v' \leq u$, $\forall j < i\ v'(j) = 0 \wedge v' \geq (v''+\mathbf{1}_i)$ $i > 0$, $v \geq v' - \mathbf{1}_i + (n')\mathbf{1}_{i-1}$ $\mathbf{V} \leq_* r$
Rule 12	$\mathbf{V}v\#wt, n \xrightarrow{r} \mathbf{V}v''\#vwt, n$ $\exists i > 0\ v(i) > 0 \wedge \forall j < i\ v(j) = 0$, $\forall j \notin \{i-1, i\}\ v''(j) = v(j)$, $v''(i) = v(i) - 1$ and $v''(i-1) = n$ $w \leq_{\text{lex}} v$	$rvs\#s'wt, n \xrightarrow{r} \mathbf{V}v''\#v'w't, n'$ $\mathbf{V} \leq_* r$, $v' \leq v\ \mathbf{V}v'$ pure $\exists i > 0\ v'(i) > 0 \wedge \forall j < i\ v'(j) = 0$ $v'' \leq v' - \mathbf{1}_i + (n')\mathbf{1}_{i-1}$ $w' \leq w$ and $t' \leq_* t$	$rvs\#s'us''wt, n \xrightarrow{r} \mathbf{V}(v''+\mathbf{1}_i)\#w't', n'$ $\exists v' \leq u$, $\forall j < i\ v'(j) = 0 \wedge v' \geq (v''+\mathbf{1}_i)$ $i > 0$, $v \geq v' - \mathbf{1}_i + (n')\mathbf{1}_{i-1}$ $w' \leq_{\text{lex}} v'$, $\mathbf{V} \leq_* r$ and $t' \leq_* t$
Rule 13	$\mathbf{V}v\#wt, n \xrightarrow{r} \mathbf{V}v''\#wt, n$ $\exists i > 0\ v(i) > 0 \wedge \forall j < i\ v(j) = 0$, $\forall j \notin \{i-1, i\}\ v''(j) = v(j)$, $v''(i) = v(i) - 1$ and $v''(i-1) = n$ $v <_{\text{lex}} w$	$rvs\#s'wt, n \xrightarrow{r} \mathbf{V}v''\#w't, n'$ $\mathbf{V} \leq_* r$, $\exists v' \leq v\ \mathbf{V}v'$ pure $\exists i > 0\ v'(i) > 0 \wedge \forall j < i\ v'(j) = 0$ $v'' \leq v' - \mathbf{1}_i + (n')\mathbf{1}_{i-1}$ $w' \leq w\ v' <_{\text{lex}} w$ and $t' \leq_* t$	$rvs\#s'wt, n \xrightarrow{r} \mathbf{V}(v''+\mathbf{1}_i)\#w't', n'$ $i > 0$, $\forall j < i\ v''(j) = 0$, $v(i-1) \geq n'$ $\forall j > i\ v''(j) \leq v(i)$ $(v''+n'\mathbf{1}_{i-1}) <_{\text{lex}} w \leq w$ $\mathbf{V} \leq_* r$ and $t' \leq_* t$

TABLE I
THE EXACT AND WEAK RULES.

E. PDN Simulation of $\xrightarrow{d}$

The implementation of all the various cases described by the rules of Table I is highly redundant, and relies on the same core ideas that we have illustrated in Section VII. We only present a few salient points that ought to convince the reader that all rules can be implemented in a PDN.

1) Weak Forward Rule 1: Let us recall this rule:

$$s\#t, n \xrightarrow{d} t, n+1 \tag{D1}$$

This (simple) rule does not require to copy the code in order to be simulated. It is sufficient to select an identity between `low` and `high` corresponding to a #, to update the `low` identity with this identity (implicitly deleting the prefix #) and to increment the counter. This is performed by a single transition depicted in Figure 5.

2) Weak Forward Rule 3: Let us recall this rule:

$$rvs\#s'\#t, n \xrightarrow{d} \mathbf{V}\#^{n''}\mathbf{0}\#t', n' \tag{D3}$$

where $\mathbf{V} \leq_* r$ is pure, $t' \leq_* t$, and $n'' \leq n' \leq n$. In the perfect case, these orderings are equalities, v is $\mathbf{0}$, and s, s' are ε.

a) First Stage: It consists in duplicating the counter, which corresponds to the following transition.

$$x, n \to x, n' \text{ where } n' \leq n$$

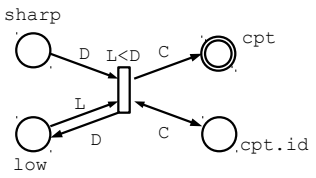


Fig. 5. Implementation of (D1).

The subsequent stages consist of an appropriate copy with update of the code.

b) Second Stage: It consists in extracting a non empty vector sequence and in copying with purification a vector sequence with the last vector (v) which is not copied. This can be performed by a net similar to (and in fact simpler than) the purification net of Figure 4. It corresponds to the following *partial* transition. The up arrows are "lower bounds" of the identities of `from` and `to`.

$$rv_{\uparrow}x', n' \rightarrow \mathbf{V}_{\uparrow} \text{ where } \mathbf{V} \leq_* r, \mathbf{V} \text{ pure and } n' \leq n$$

c) Third Stage: It consists in picking two $\#$ in the remaining code (x') and in copying, with the help of the duplicated counter, at most n' $\#$ symbols followed by the null vector and a $\#$. This is performed by the net of Figure 6. Transition $\texttt{mult}_1$ checks that `from` contains the identity of a $\#$ symbol and adds such a symbol to the new code. Transition $\texttt{mult}_2$, which can be performed at most n' times, also adds a $\#$ at the end of the new code. Finally, transition $\texttt{mult}_3$ adds a null vector to the new code and "moves forward" the identity contained in `from`. All transitions also move forward the token in `to`. Creating a null vector is easy since it simply consists in adding a `vect` identity to the new code. We have not represented the concatenation of the $\#$ which simply consists in adding a token in place `sharp` with identity `to` and in increasing the identity.

This stage corresponds to the partial transition:

$$rvs\#s'\#_{\uparrow}t, n' \rightarrow \mathbf{V}\#^{n''}\mathbf{0}\#_{\uparrow}$$

where $\mathbf{V} \leq_* r$, $\mathbf{V}$ pure and $n'' \leq n' \leq n$

d) Fourth Stage: The last stage consists in copying the remaining code t in a way similar to that of the weak trimming net.

3) Weak Forward Rule 7: Let us recall this rule:

$$rvs\#s'\#t, n \xrightarrow{d} \mathbf{V}v_0 \dots v_{n''}\#(v_{n''} + \mathbf{1}_0)\#t', n' \qquad \text{(D7)}$$

where $\mathbf{V} \leq_* r$, $v_0 + \mathbf{1}_0 \leq v$, $\mathbf{V}(v_0 + \mathbf{1}_0)$ pure, $\forall 0 \leq i \leq n$, $v_i \leq v_i$, $t' \leq_* t$, and $n'' \leq n' \leq n$

We have already presented all the gadgets necessary to simulate this rule. First we duplicate the counter which yields a new value $n' \leq n$. Then we extract and copy a non empty pure vector sequence, the last vector containing at least a token

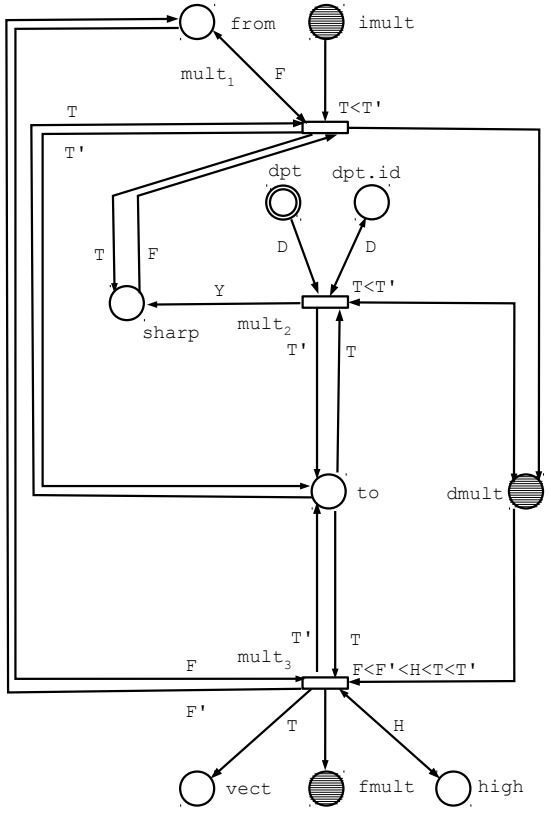


Fig. 6. Multiplying the tally symbol

in the 0-component; this sequence is $\mathbf{V}(v_0 + \mathbf{1}_0)$. We delete this token and extend it to a sequence of n non increasing vectors $v_0 \dots v_{n''}\#v_{n''}$ (inserting a $\#$ before and after writing the last vector). This can be done by a modified version of the purification net where the duplicated counter controls the length of the sequence (as for the duplication of $\#$ in rule D2). We add a token to the last vector. Finally, we copy the remaining code t at the end of the new code.

4) Weak Forward Rule 11: Let us recall this rule:

$$rvs\#s'\#t, n \xrightarrow{d} \mathbf{V}v''\#v't', n' \qquad \text{(D11)}$$

where $\mathbf{V} \leq_* r$, $v' \leq v$, $\exists i > 0\ v'(i) > 0 \wedge \forall j < i\ v'(j) = 0$, $\mathbf{V}v'$ pure, $v'' \leq (v')_{n''}$, $t' \leq_* t$ and $n'' \leq n' \leq n$

We simulate the rule in four stages. First we duplicate the counter. Then we simulate the following partial copy:

$$rv_{\uparrow}x' \rightarrow \mathbf{V}v'\#v'$$

It consists in extracting and copying a non empty pure vector sequence $\mathbf{V}v'\#v'$. The copy of the last vector is particular since

- we arbitrarily choose some component index i, such that $v'(i) > 0$; we memorize this choice by marking place $\texttt{deriv}_i$.
- we set $v'(j) = 0$ for all $j < i$;

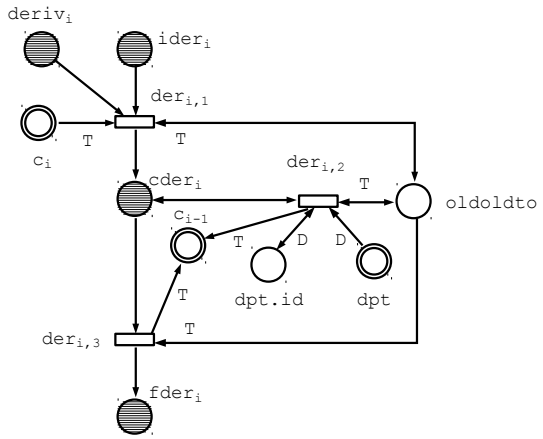


Fig. 7. Derivating a vector (case i)

- we duplicate v' in the new code and separate them by a $\#$. This can be done by moving forward `to` three times and keeping in auxiliary places `oldto` and `oldoldto` the identities that precede the current one. Then the copy is done simultaneously using `to` and `oldoldto` for the two copies of the vector, while a $\#$ is inserted with identity `oldto`.

The third stage consists in deriving it, i.e. in deleting a token in $\mathtt{c}_i$ and adding at most n' tokens in $\mathtt{c}_{i-1}$. This is done by the net of Figure 7 for some case i. Transition $\mathtt{deriv}_{i,1}$ starts the derivation by subtracting a token in $\mathtt{c}_i$. Then there are at most n applications of $\mathtt{deriv}_{i,2}$ adding tokens in $\mathtt{c}_{i-1}$. Finally, $\mathtt{deriv}_{i,3}$ stops the derivation by adding one more token in $\mathtt{c}_{i-1}$.

The last stage consists in copying the remaining code $\#t$ at the end of the new code.

We end this subsection by presenting the copy of remaining vectors of a sequence during the weak trimming in figure 8.

Additional References

[24] E. A. Cichoń and E. Tahhan Bittar, "Ordinal recursive bounds for Higman's Theorem," *Theor. Comput. Sci.*, vol. 201, pp. 63–84, 1998.

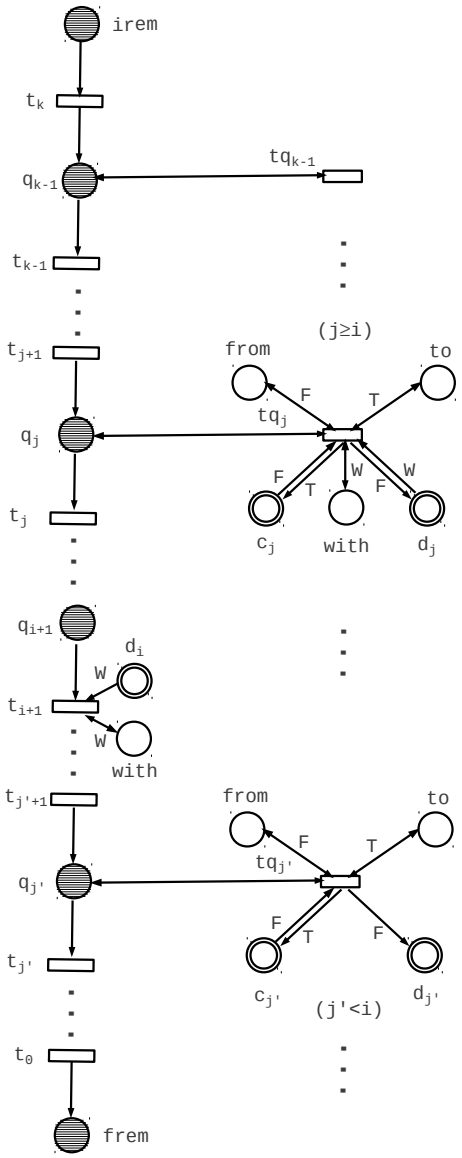


Fig. 8. Weak Trimming: Copying the remaining vectors.