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 $\mathbf{0}$-*extension* of $s \in (\mathbb{N}^k \setminus \mathbf{0})^*$ (or s is the $\mathbf{0}$-*contraction* of s') $\stackrel{\text{def}}{\Leftrightarrow}$ s is obtained by removing all $\mathbf{0}$'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 $\mathbf{0}$-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 $(S, s_0, \rightarrow)$ with state set $S \stackrel{\text{def}}{=} (\mathbb{N}^k \setminus \mathbf{0})^*$ and transition relation $\rightarrow \stackrel{\text{def}}{=} \bigcup_{t \in T} \stackrel{t}{\rightarrow}$, where $s \stackrel{t}{\rightarrow} s'$ 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 $\mathbf{0}$-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 $\mathbf{0}$-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 $\mathbf{x} \leq \mathbf{y} \stackrel{\text{def}}{\Leftrightarrow} \forall 0 \leq j < k, \mathbf{x}[j] \leq \mathbf{y}[j]$,

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 $s \leq_* s' \stackrel{\text{def}}{\Leftrightarrow} s = a_1 \cdots a_n$, $s' = x_0 b_1 x_1 \cdots b_n x_n$ 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 $s_1 \stackrel{t}{\rightarrow} s_2$ and $s_1 \leq_* s_3$, then there exists s_4 s.t. $s_2 \leq_* s_4$ and $s_3 \stackrel{t}{\rightarrow} s_4$. 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

$$\alpha = \omega^{\alpha_1} \dot{+} \cdots \dot{+} \omega^{\alpha_p} , \qquad (1)$$

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 $\alpha \dotplus \alpha'$ 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:

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

together with $\omega_n \stackrel{\text{def}}{=} n$. Writing Ω for the ordinal $\omega^{\omega^{\omega^\omega}}$, 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 $F_{\omega^{\omega^\omega}}\left(n + O(1)\right)$.*

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 $F_{\omega^{\omega^\omega}}$ 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 $F_{\omega^{\omega^\omega}}$ 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 $F_{\omega^{\omega^\omega}}$-hard.*

This follows from a reduction from the halting problem for Minsky machines (MM) M with counters bounded by $F_{\omega^{\omega^\omega}}(|M|)$. 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.
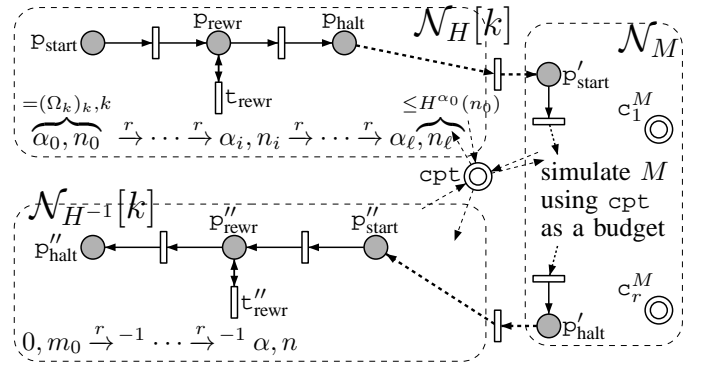

Fig. 1. Schematics for Theorem 4.

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 $H^\Omega(k) = F_{\omega^{\omega^\omega}}(k)$ 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 $h = F_{\omega^{\omega^\omega}}(k)$.

In conclusion, the construction provides a (logspace) many-one reduction from the halting problem for Minsky machines running in space bounded by $F_{\omega^{\omega^\omega}}(k)$ 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* $F_{\omega^{\omega^\omega}}$*-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 $\stackrel{r}{\to}$ 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 ε for the empty sequence in $\mathbb{N}^{k*}$. Then $\alpha(\mathbf{V}) = 0$ iff $\mathbf{V} = \varepsilon$, and $\alpha(\mathbf{V}) = 1$ iff $\mathbf{V} = \mathbf{0}$.

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

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

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

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

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

We say that x is *pure* if each $\mathbf{V}_i$, $i = 1, \ldots, m$, is pure, and if in addition $\mathbf{V}_{\text{rest}} = \varepsilon$. (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

$$pure(\mathbf{V}_1 \# \mathbf{V}_2 \# \cdots \# \mathbf{V}_m \# \mathbf{V}_{\text{rest}}) = \\ pure(\mathbf{V}_1) \# pure(\mathbf{V}_2) \# \cdots \# pure(\mathbf{V}_m) \# \varepsilon . \quad (10)$$

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 $v <_{\text{lex}} v'$, $\alpha(v\,v') = \omega^{\beta(v)} + \omega^{\beta(v')} = \omega^{\beta(v')} = \alpha(v')(\neq \omega^{\beta(v')} \dot{+} \omega^{\beta(v)})$.

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

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

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. $(\pi(\mathbf{V}v\#x))_n = \pi(\mathbf{V}vx) \dot{+} (\omega^{\alpha(\mathbf{V}v)})_n$. We want to define a similar mapping $(.)_n$ from codes to codes s.t. $\pi((x)_n) = (\pi(x))_n$; 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
$$\pi((\mathbf{V}\mathbf{0}\#x)_n) \stackrel{\text{def}}{=} \pi(\mathbf{V}\#^n \mathbf{0}x) \quad (13)$$
$$= \pi(\mathbf{V}\mathbf{0}x) \dot{+} \omega^{\alpha(\mathbf{V})} \cdot n$$
$$= (\pi(\mathbf{V}\mathbf{0}\#x))_n .$$

- 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

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

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

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

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 $H^{\omega^\omega}(n) = H^{\omega^n}(n) < H^{\omega^{n+1}}(n)$. $\square$

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. $\square$

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, $x_0, n_0 \xrightarrow{r}{}^* \varepsilon$, $H^{\pi(x_0)}(n)$ implies $x_0, n_0 \xrightarrow{rt}{}^* \varepsilon$, $H^{\pi(x_0)}(n)$ and $\varepsilon, H^{\pi(x_0)}(n) (\xrightarrow{rt}{}^{-1})^* x_0, n_0$ since ε is $\left(H^{\pi(x_0)}(n)\right)$-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

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

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

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

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 $c_0, \ldots, c_{k-1}$: $v[i]$ is the number of tokens in 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 dp_1 performs the identity updating: `cpt.id` acquires a smaller identity $C' < C$ while `dpt.id` is updated with the previous identity of `cpt.id`, namely C. Then transition 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.
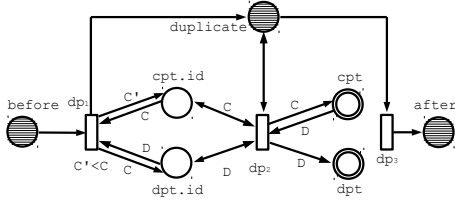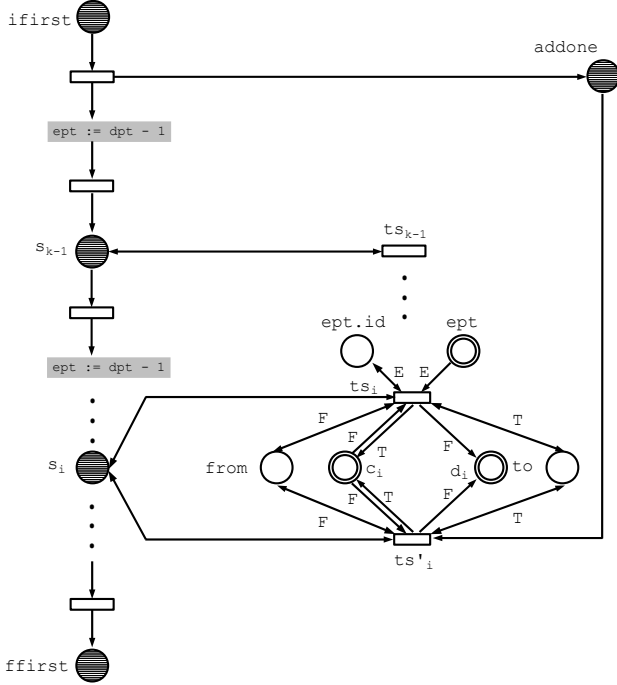
Fig. 2. Duplication of the counter value.

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

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 D' is selected by the guard $\text{C} < \text{D}' < \text{D}$ 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 $\text{d}_0, \ldots, \text{d}_{k-1}$ interpreted as the 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 $\text{d}_0, \ldots, \text{d}_{k-1}$.
- 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 $\text{d}_0, \ldots, \text{d}_{k-1}$. 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
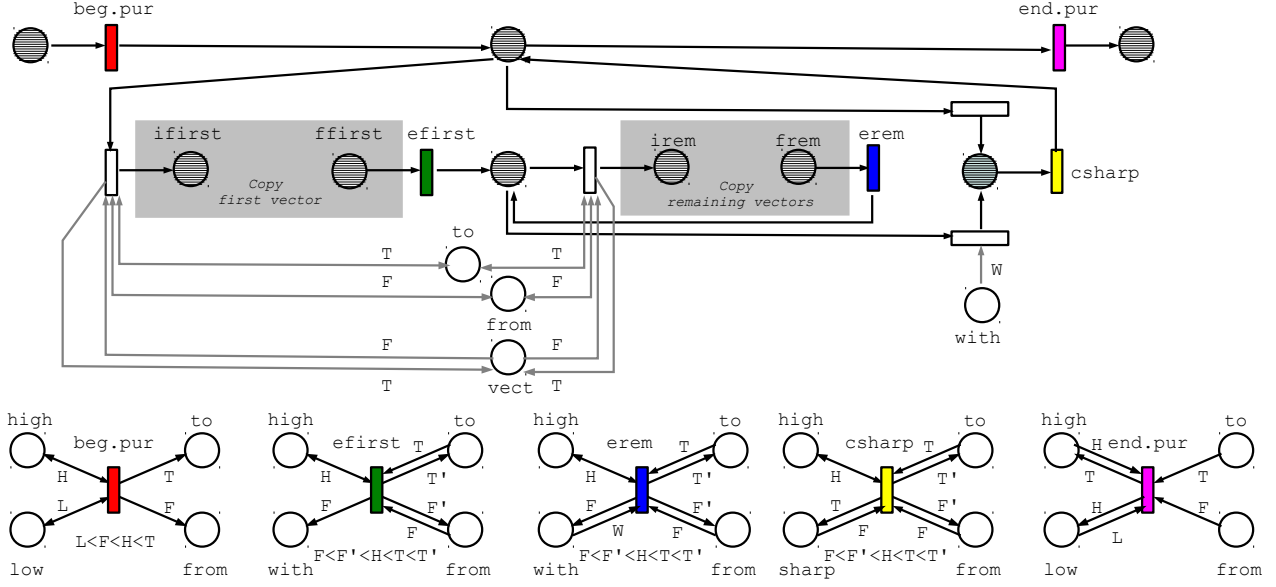
Fig. 4.   Control part of the trimming phase.

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

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 $\mathbb{L}(\text{PDN})$, 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.

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.