

Ordinal Recursive Complexity of Unordered Data Nets

Fernando Rosa-Velardo

*Dpto. de Sistemas Informáticos y Computación, Facultad de Informática
Universidad Complutense de Madrid
C/Profesor José García Santesmases, 9. 28040 Madrid (Spain)*

Abstract

Data Nets are a version of colored Petri nets in which tokens carry data taken from an infinite, linearly ordered and dense domain. This class is interesting because, even though very expressive, their coverability/termination problems remain decidable. Those problems have recently been proven complete for the class $\mathbf{F}_{\omega^\omega}$ in the fast growing complexity hierarchy. In this paper we characterize the exact ordinal-recursive complexity of Unordered Data Nets (*UDN*), a subclass of Data Nets in which the data carried by tokens belong to an *unordered* domain. We adapt the result by Schmitz and Schnoebelen to bound the length of bad sequences in well-quasi orderings based on finite words over tuples of naturals to the case of finite multisets, and we obtain hyper-Ackermannian upper bounds for those lengths, which imply that both problems are in $\mathbf{F}_{\omega^\omega}$. Then we prove that the previous bounds are tight, by constructing *UDN* that weakly compute fast-growing functions and their inverses. Up to our knowledge, this is the first problem that is complete at the $\mathbf{F}_{\omega^\omega}$ level of the fast-growing hierarchy, with an underlying wqo not based on finite words over a finite alphabet.

Key words: Data nets, Ordinal Recursive Complexity,
Hyper-Ackermannian problems, Fast-growing complexity hierarchy,
Multisets over \mathbb{N}^k

Email address: fernandorosa@sip.ucm.es (Fernando Rosa-Velardo)

Author is partially supported by the Spanish projects TRACES TIN2015-67522-C3-3-R and N-Greens S2013/ICE-2731.

1. Introduction

Higman's Lemma [12] is a well-known result that states that whenever (X, \leq) is a well-quasi order (wqo) then the embedding order (X^*, \leq^*) in the set X^* of finite words over X is also a wqo. As a consequence, and because \leq^* is a refinement of the multiset order \leq^\oplus ($s \leq^* s'$ implies $s \leq^\oplus s'$), the order \leq^\oplus over the set of multisets X^\oplus is also a wqo.

However, multisets are intuitively a simpler domain than words. This is witnessed by their *maximal ordinal types* [14, 20], which can be seen as a measure of their size. Indeed, if the order type of X is α then the order type of X^* is ω^{ω^α} [14], while the order type of X^\oplus is only ω^α [24].²

The ordinal type of a wqo has recently been used in several works [6, 21, 8, 11, 16] to characterize the ordinal-recursive complexity of (the verification of several problems for) *monotonic* systems over an underlying wqo, which have been called Well-Structured Transition Systems (WSTS) [2, 9]. Prominent examples of such WSTS are Petri nets/VASS, affine nets [10], Lossy Channel Systems [1, 5] or Data Nets [15]. Lossy Channel Systems (LCS) can be seen as finite state machines communicating over FIFO unreliable channels. Hence, if Γ is the (finite) alphabet of messages, the state space is given by $Q \times (\Gamma^*)^k$ for some finite set of states Q and $k \geq 0$.

Data Nets [15] are a very general (but monotonic) extension of Petri nets in which tokens are taken from a linearly ordered and dense domain, and whole-place operations like transfers or resets are allowed. They can be seen as arrays or lists of Petri nets (with whole-place operations) communicating by rendezvous and broadcasts [4]. Hence, the state space of a Data Net is given by some $(\mathbb{N}^k)^*$. It was shown in [3] that Petri Data Nets (Data Nets in which no whole-place operations or broadcasts are allowed) are already as expressive as Data Nets. Data Nets are used e.g. in [7], to give semantics and prove decidability results of a very expressive formal model of asynchronous event-driven programs, allowing dynamic creation of concurrent tasks, events and threads.

A natural subclass of Data Nets is that of Unordered Data Nets (*UDN*), that is, the subclass of Data Nets in which nets are not in an array of nets, but in a pool of nets. Unlike in Data nets, where the underlying state space is based on finite words of tuples of naturals, in the case of Unordered Data Nets the state space is based on finite multisets of such tuples. Thus, in

²This holds only for $\omega \leq \alpha < \epsilon_0$, which will be the case in the paper.

the unordered case the state space can be given by some $(\mathbb{N}^k)^\oplus$. Unordered Data Nets are very closely related to the class of so-called ν -Petri nets. It can be seen as a restriction of UDN without broadcasts, so that each transition affects only a constant number of names given by the transition. In ν -Petri nets tokens carry pure identifiers, that can only be compared with each other by equality, and can be created fresh (different from all other existing identifiers), which can be used for the modeling of distributed authentication protocols (see for instance [17]). Also, ν -Petri nets and its polyadic version (in which tokens are tuples of identifiers) are in turn closely related to process algebra with name creation (e.g., channel names in the π -calculus) [19].

On the one hand, ν -Petri Nets (UDN without whole-place operations or broadcasts) have been recently proven complete for the $\mathbf{F}_{\omega \cdot 2}$ class of double Ackermannian problems in the fast-growing complexity hierarchy [16]. On the other hand, (full) Data Nets have been recently proven to be complete for the $\mathbf{F}_{\omega^\omega}$ class in the fast-growing complexity hierarchy [11]. We fill part of the gap in between by proving that UDN are complete for $\mathbf{F}_{\omega^\omega}$.

Our proof relies on the techniques developed by Schnoebelen and Schmitz, both for the upper bounds as for the hardness result. For the upper bounds, we adapt the techniques in [21] to bound the length of (controlled) bad sequences in $(\mathbb{N}^k)^\oplus$, that is, sequences s_0, \dots, s_k such that $s_i \not\leq s_j$ for all $i < j$. These upper bounds give us ordinal recursive upper bounds for the coverability and termination problems. This result is general, and could therefore be used for other monotonic systems with a state space based on multisets of tuples of naturals. For the lower bound, we show (i) how we can encode ordinals below ω^{ω^ω} as markings of UDN and (ii) how we can use this encoding to perform weak computations of fast-growing functions and their inverses. This entails the corresponding lower bounds, by using the device presented for instance in [23].

Thus, the complexity of UDN sits at the exact same level as LCS, which were proven to be $\mathbf{F}_{\omega^\omega}$ -complete in [6, 21]. Let us remark that, because the state spaces of $UDN((\mathbb{N}^k)^\oplus)$ and the state spaces of LCS $(Q \times (\Gamma^*)^k)$ are very different (though with comparable order types), it does not seem possible to perform direct reductions from one model to the other, which would yield alternative (and perhaps more direct) proofs of our results. We leave these reductions as open problems.

Finally, let us comment that the construction reducing Data nets to Petri Data Nets (removing broadcasts) [3] is no longer correct in the case of UDN .

We will see that our hardness result heavily relies on broadcast operations that, for instance, empty a given place in all tuples. This is not a coincidence, since *UDN* (hyper Ackermannian) are strictly harder than ν -Petri Nets (double Ackermannian).

The rest of the paper is structured as follows. Section 2 presents some definitions, notations and results we use in the paper. In Sect. 3 we obtain upper bounds for the length of controlled bad sequences in multiset wpos. In Sect. 4 we define *UDN* and obtain an upper bound for their coverability and termination problems. In Sect. 5 we consider lower bounds. Sect. 6 presents our conclusions and some open problems.

2. Preliminaries

Well Orders. (X, \leq_X) is a *quasi-order* (qo) if \leq_X is a reflexive and transitive binary relation on X . For a qo we write $x <_X y$ iff $x \leq_X y$ and $y \not\leq_X x$. A *partial order* (po) is an antisymmetric quasi-order. A po (X, \leq) is *total* (or *linear*) if for any $x, x' \in X$ either $x \leq x'$ or $x' \leq x$. We will shorten (X, \leq_X) to X when the underlying order is obvious. Similarly, \leq will be used instead of \leq_X when X can be deduced from the context.

We say a (finite or infinite) sequence $(x_i)_{i \leq \omega}$ is *good* if there are indices $i < j$ such that $x_i \leq x_j$. Otherwise, we say it is *bad*. A po X is a *well partial order* (wpo) if every bad sequence is finite.

If X_1 and X_2 are wpos, their Cartesian product $X_1 \times X_2$ is well ordered by $(x_1, x_2) \leq_{X_1 \times X_2} (x'_1, x'_2)$ iff $x_1 \leq_{X_1} x'_1$ and $x_2 \leq_{X_2} x'_2$. Their disjoint sum $X_1 + X_2 = (\{1\} \times X_1) \cup (\{2\} \times X_2)$ is well partially ordered by $\langle i, x \rangle \leq_{X_1 + X_2} \langle j, x' \rangle$ iff $i = j$ and $x \leq_{X_i} x'$.

Functions. If X and Y are ordered, a mapping $f : X \rightarrow Y$ is *increasing* (resp. *strictly increasing*) if $x \leq_X y$ implies $f(x) \leq_Y f(y)$ (resp. if $x <_X y$ implies $f(x) <_Y f(y)$); f is an *order embedding* (shortly: embedding) if $f(x) \leq_Y f(x')$ iff $x \leq_X x'$. A bijective order embedding is called an *order isomorphism* (shortly: isomorphism). Two po X and Y are isomorphic if there is an isomorphism between them, in which case we write $X \equiv Y$.

Multisets. Given a set X , we denote by X^\oplus the set of finite multisets of X , that is, the set of mappings $m : X \rightarrow \mathbb{N}$ with a finite support $\text{supp}(m) = \{x \in X \mid m(x) \neq 0\}$. We use the set-like notation for multisets when convenient, with $\{x^n\}$ describing the multiset with n occurrences of x . We use $+$ and $-$ for multiset addition and subtraction, respectively defined by $(m + m')(x) = m(x) + m'(x)$ and $(m - m')(x) = \max(m(x) - m'(x), 0)$. If X is

a wpo then so is X^\oplus ordered by \leq_\oplus defined by $\{x_1, \dots, x_n\} \leq_\oplus \{x'_1, \dots, x'_m\}$ if there is an injection $h : \{1, \dots, n\} \rightarrow \{1, \dots, m\}$ such that $x_i \leq_X x'_{h(i)}$ for each $i \in \{1, \dots, n\}$.

Words. Given a set X , any $u = x_1 \cdots x_n$ with $n \geq 0$ and $x_i \in X$, for all $i \in \{1, \dots, n\}$, is a finite word on X . We denote by X^* the set of finite words on X . If $n = 0$ then u is the empty word, which is denoted by ε . If X is a wpo then so is X^* ordered by \leq_{X^*} which is defined as follows: $x_1 \dots x_n \leq_{X^*} x'_1 \dots x'_m$ if there is a strictly increasing mapping $h : \{1, \dots, n\} \rightarrow \{1, \dots, m\}$ such that $x_i \leq_X x'_{h(i)}$ for each $i \in \{1, \dots, n\}$ (Higman's Lemma). This is called the *embedding order*, in contrast with the *lexicographic order*: $x_1 \dots x_n <_{lex} x'_1 \dots x'_m$ iff there is $i \leq \min(n, m)$ such that $x_j = x'_j$ for $j < i$ and $x_i < x'_i$, or $x_1 \dots x_n = x'_1 \dots x'_n$ and $m > n$. Then the reflexive closure \leq_{lex} of $<_{lex}$ is a total (or linear) order, though not necessarily a well partial order (for instance, b, ab, aab, \dots is strictly decreasing).

WSTS. A *transition system* is a tuple $\mathcal{S} = \langle X, \rightarrow \rangle$ where X is the set of states and $\rightarrow \subseteq X \times X$ is the transition relation. We write $x \rightarrow x'$ instead of $(x, x') \in \rightarrow$ and denote by \rightarrow^* the reflexive and transitive closure of \rightarrow . A *Well Structured Transition System* (WSTS) is a tuple $\mathcal{S} = (X, \rightarrow, \leq)$, where (X, \rightarrow) is a transition system, and \leq is a wpo on X , satisfying the following monotonicity condition: for all $x_1, x_2, x'_1 \in X$, $x_1 \leq x'_1$, $x_1 \rightarrow x_2$ implies the existence of $x'_2 \in X$ such that $x'_1 \rightarrow x'_2$ and $x_2 \leq x'_2$. The *coverability* problem is that of deciding, given x_0 and x , whether $x_0 \rightarrow^* x'$ for some $x' \geq x$. The *termination* problem asks, given x_0 , whether there is an infinite sequence $x_0 \rightarrow x_1 \rightarrow x_2 \dots$.

Given a WSTS $\mathcal{S} = (X, \rightarrow, \leq)$, we say $\mathcal{S}_l = (X, \rightarrow_l, \leq)$ is a *lossy* version of \mathcal{S} if \rightarrow_l is obtained by adding to \rightarrow some transitions $x \rightarrow_l x'$ with $x > x'$. It is easy to see that the termination and coverability problems in \mathcal{S} and \mathcal{S}_l are equivalent. We will use this fact throughout the paper by considering lossy versions of WSTS, in which extra lossy transitions are introduced.

Controlled bad sequences. We say $g : \mathbb{N} \rightarrow \mathbb{N}$ is *smooth* if $g(x) > x$ and $g(x + y) \geq g(x) + g(y)$ for all $x, y \in \mathbb{N}$. This implies that $g(x + 1) \geq g(x) + 1 \geq x + 2$ and $g(x \cdot y) \geq g(x) \cdot y$. A *normed wpo* is a wpo (X, \leq) endowed with a norm $|\cdot|_X : X \rightarrow \mathbb{N}$, such that for any $n \in \mathbb{N}$, the set $X_{<n} = \{x \in X \mid |x|_X < n\}$ is finite. Given a normed wpo, a sequence x_0, \dots, x_L of elements in X is *g, n -controlled* if $|x_i|_X < g^i(n)$ for every i . We define $L_{g,X}(n)$ as the length of the longest g, n -controlled bad sequence in X (which exists because of König's Lemma).

Fundamentals on ordinals. We only work with ordinals $\alpha < \epsilon_0$. These can be represented in Cantor Normal Form (CNF) $\alpha = \omega^{\alpha_1} + \dots + \omega^{\alpha_n}$, with $\alpha_1 \geq \dots \geq \alpha_n$ in CNF, and where the order between two ordinals is defined as follows:

$$\omega^{\alpha_1} + \dots + \omega^{\alpha_n} < \omega^{\beta_1} + \dots + \omega^{\beta_m} \Leftrightarrow \alpha_1 \dots \alpha_n <_{lex} \beta_1 \dots \beta_m$$

An ordinal of the form $\alpha + 1$ is called a *successor*. Otherwise, it is a *limit*, and we use λ to denote limit ordinals. We sometimes write $\omega^\alpha \cdot c$ instead of $\omega^\alpha + \dots + \omega^\alpha$ (c times), and use the so called strict form $\alpha = \omega^{\alpha_1} \cdot c_1 + \dots + \omega^{\alpha_m} \cdot c_m$, with $\alpha_1 > \dots > \alpha_m$ and $c_i > 0$. The natural sum $\alpha \oplus \alpha'$ and natural product $\alpha \otimes \alpha'$ of two ordinals are defined as:

$$\sum_{i=1}^m \omega^{\alpha_i} \oplus \sum_{j=1}^n \omega^{\alpha'_j} = \sum_{k=1}^{m+n} \omega^{\beta_k} \quad \sum_{i=1}^m \omega^{\alpha_i} \otimes \sum_{j=1}^n \omega^{\alpha'_j} = \bigoplus_{i=1}^m \bigoplus_{j=1}^n \omega^{\alpha_i \oplus \alpha'_j}$$

where $\beta_1 \geq \dots \geq \beta_{m+n}$ is a reordering of $\alpha_1, \dots, \alpha_m, \alpha'_1, \dots, \alpha'_n$.

Ordinal recursive complexity. Given a limit ordinal λ we define the *fundamental sequence* $(\lambda_n)_{n < \omega}$, as follows:

$$(\gamma + \omega^{\alpha+1})_n = \gamma + \omega^\alpha \cdot n, \quad (\gamma + \omega^\lambda)_n = \gamma + \omega^{\lambda_n} \quad (1)$$

It holds $\lambda_n < \lambda$ for every n , and $\lambda = \sup_n \lambda_n$. Given α and $x \in \mathbb{N}$, we define the x -predecessor $P_x(\alpha) < \alpha$ of α as follows:

$$P_x(\alpha + 1) = \alpha, \quad P_x(\lambda) = P_x(\lambda_x)$$

Fundamental sequences are used to define subrecursive hierarchies: The *Hardy functions* $(h^\alpha : \mathbb{N} \rightarrow \mathbb{N})_\alpha$ are defined by

$$h^0(n) = n, \quad h^{\alpha+1}(n) = h^\alpha(h(n)), \quad h^\lambda(n) = h^{\lambda_n}(n)$$

The *fast growing hierarchy* $(F_\alpha : \mathbb{N} \rightarrow \mathbb{N})_\alpha$ can be defined as $F_\alpha = H^{\omega^\alpha}$, where $H(n) = n + 1$.

Given α , the class \mathcal{F}_α is the least class containing F_β for all $\beta \leq \alpha$ and all constants, and closed under addition, projection, substitution and limited recursion. The hierarchy $(\mathcal{F}_\alpha)_\alpha$ is called the *extended Grzegorzczuk hierarchy*. Instead of directly working with the extended Grzegorzczuk hierarchy, we

work with the fast growing complexity hierarchy $(\mathbf{F}_\alpha)_\alpha$, defined by Schmitz for instance in [22], and better suited to establish completeness results.

$$\mathbf{F}_\alpha = \bigcup_{\substack{\beta < \alpha \\ p \in \mathcal{F}_\beta}} \text{TIME}(H^{\omega_\alpha}(p(n))) \quad (2)$$

Here, $\text{TIME}(f(n))$ stands for the class of problems decidable in time $f(n)$.³ For instance, \mathbf{F}_3 is the class of *tower*-like problems with a complexity bounded by the non-elementary function $\text{tower}(x) = 2^{\cdot^{\cdot^{\cdot^2}}}$ x times, closed under elementary reductions, \mathbf{F}_ω is the class of Ackermannian problems, closed under primitive-recursive reductions, and $\mathbf{F}_{\omega^\omega}$ is the class of hyper-Ackermannian problems, closed under multiply-recursive reductions. For every α , the halting problem for a Minsky machine M with sum of counters bounded by $F_\alpha(|M|)$ is an \mathbf{F}_α -complete problem [22].

It is worth noting here that the definition of the fast-growing complexity classes above is robust in two senses: On the one hand, one can take slight variations in the definition of fundamental sequences. For instance, [22] defines them so that $\omega_n = n + 1$ (instead of $\omega_n = n$, as in the definition here), though these variations define the same fast-growing complexity classes [22, Cor. 4.5]. On the other hand, one can use different Hardy functions h^β obtained from different *control functions* h in Eq. 2. In particular, Schmitz proves in [22, Cor. 4.3] that for $\alpha \geq \omega$ one can use any increasing primitive recursive function h to define \mathbf{F}_α . We will use this fact to obtain our upper bounds in Th. 2.

3. Upper bounds in multiset wpos

In this section we obtain upper bounds for the length of bad sequences in wpos involving multisets of tuples of naturals. We work with normed wpos that can be obtained by the following grammar:

$$A ::= \emptyset \mid \mathbf{Unit} \mid (\mathbb{N}^d)^\oplus \mid A + A \mid A \times A \quad (3)$$

where \emptyset denotes the empty wpo, \mathbf{Unit} denotes the singleton wpo (with the trivial order), $(\mathbb{N}^d)^\oplus$ is the wpo of multisets of tuples of arity d (for some

³The choice of using nondeterministic/deterministic bounds, or space/time bounds, is irrelevant in classes beyond exponential.

$d \geq 0$). Each A is endowed with its standard order. In this algebra we can also have finite sets (as sums of **Unit**) and naturals (taking $\mathbb{N}^0 = \mathbf{Unit}$, so that $\mathbb{N} \equiv (\mathbb{N}^0)^\oplus$). We write Q_m for the finite set with m elements ordered by equality, so that $Q_0 \equiv \emptyset$ and $Q_1 \equiv \mathbf{Unit}$. Each A is also endowed with a norm $|\cdot|_A$, given by:

- $|unit|_{\mathbf{Unit}} = 0$ and $|n|_{\mathbb{N}} = n$,
- $|a|_{A+A'} = |a|_A$ if $a \in A$ (analogously if $a \in A'$),
- $|(a, a')|_{A \times A'} = \max(|a|_A, |a'|_{A'})$,
- $|\{a_1, \dots, a_n\}|_{A^\oplus} = \max(n, |a_1|_A, \dots, |a_n|_A)$.

If we mean \emptyset by the empty sum and **Unit** by the empty product, any A can be canonically written as

$$A \equiv \sum_{i=1}^m \prod_{j=1}^{k_i} (\mathbb{N}^{d_{ij}})^\oplus$$

for some $m, k_i, d_{ij} \geq 0$. We next recall two important tools to bound the length of controlled bad sequences: reflections and residuals.

Definition 1 (Reflections). *Let A, B be two wpos. A normed reflection is any mapping $h : A \rightarrow B$ such that $h(a_1) \leq h(a_2)$ implies $a_1 \leq a_2$ for all $a_1, a_2 \in A$ (it is a reflection) and $|h(a)|_B \leq |a|_A$ for all $a \in A$.*

We write $A \hookrightarrow B$ if there is a normed reflection from A to B . It holds (see [21]) that $A \hookrightarrow B$ implies $L_{g,A}(n) \leq L_{g,B}(n)$.

Definition 2 (Residuals). *Let A be a normed wpo and $a \in A$. We define the set of residuals of a in A as the set $A/a = \{b \in A \mid a \not\leq b\}$.*

Residuals can be used to bound the length of bad sequences, thanks to the following recurring equation, called the *Descent Equation*:

$$L_{g,A}(n) = \max_{a \in A_{<n}} \{1 + L_{g,A/a}(g(n))\}$$

Proposition 1. *The following reflections or isomorphisms hold:*

1. $(A + B)/a \equiv (A/a) + B$ if $a \in A$ (analogously for $b \in B$).
2. $(A \times B)/(a, b) \hookrightarrow ((A/a) \times B) + (A \times (B/b))$.
3. $\mathbf{Unit}/unit \equiv \emptyset$.
4. $A^\oplus/\{x_1, \dots, x_k\} \hookrightarrow A^{k-1} \times [(A/x_1)^\oplus + \dots + (A/x_k)^\oplus]$
5. $(\mathbb{N}^d)^\oplus/\{x_1, \dots, x_k\} \hookrightarrow \mathbb{N}^{d \cdot (k-1)} \times [(\mathbb{N}^{d-1} \cdot d \cdot |x_1|)^\oplus + \dots + (\mathbb{N}^{d-1} \cdot d \cdot |x_k|)^\oplus]$

PROOF. The cases for sum and product are from [21], and the case for **Unit** is trivial. The last case is an instantiation of (4) for $A = \mathbb{N}^d$. For the 4th case, we first prove the following lemma.

$$A^\oplus/(m + \{x\}) \hookrightarrow (A/x)^\oplus + (x \uparrow \times (A^\oplus/m))$$

where $x \uparrow = \{y \mid x \leq y\}$. Let $m' \in A^\oplus/(m + \{x\})$, i.e., $m' \in A^\oplus$ and $m + \{x\} \not\leq m'$. Then, either $m' \in (A/x)^\oplus$ (it is a multisets of elements *not* above x), or $m' = m'' + \{y\}$ with $x \leq y$ but $m \not\leq m''$. This prompts us to the following definition:

$$h : A^\oplus/(m + \{x\}) \rightarrow (A/x)^\oplus + (x \uparrow \times (A^\oplus/m))$$

$$h(m') = \begin{cases} \langle 1, m' \rangle & \text{if } m' \in (A/x)^\oplus \\ \langle 2, (y, m' - \{y\}) \rangle & \text{if } m' - \{y\} \not\leq m \text{ and } y \geq x \end{cases}$$

In the second case above, the particular choice of y is irrelevant. Let us see that h is a normed reflection. Assume that $h(m'_1) \leq h(m'_2)$. Then, we are in one of the following two cases: (i) $h(m'_i) = \langle 1, m'_i \rangle$, in which case $m'_1 \leq m'_2$; (ii) $h(m'_i) = \langle 2, (y_i, m'_i - \{y_i\}) \rangle$, with $y_1 \leq y_2$ and $m'_1 - \{y_1\} \leq m'_2 - \{y_2\}$, in which case we can also conclude that $m'_1 \leq m'_2$.

Let us now see that $|h(m')| \leq |m'|$. Let $m' = \{y_1, \dots, y_k\}$, so that $|m'| = \max(|y_1|, \dots, |y_k|, k)$. If $h(m') = \langle 1, m' \rangle$ then $|h(m')| = |m'|$. Otherwise, without loss of generality $h(m') = \langle 2, (y_1, m' - \{y_1\}) \rangle$, so that

$$|h(m')| = \max(|y_1|, \max(|y_2|, \dots, |y_k|, k-1)) = \max(|y_1|, \dots, |y_k|, k-1) \leq |m'|$$

Now we prove the result, by induction on k . If $k = 0$ then $A^\oplus/\emptyset \equiv \emptyset$, and $\emptyset \hookrightarrow X$ for any wpo X . For $k = 1$, $A^\oplus/\{x\} \equiv (A/x)^\oplus \equiv A^0 \times (A/x)^\oplus$. Let $k > 1$.

$$\begin{aligned} A^\oplus/\{x_1, \dots, x_k\} &\hookrightarrow (A/x_1)^\oplus + (x_1 \uparrow \times A^\oplus/\{x_2, \dots, x_k\}) && \text{(lemma)} \\ &\hookrightarrow (A/x_1)^\oplus + (A \times (A^{k-2} \times \sum_{j=2}^k (A/x_j)^\oplus)) && \text{(ind. hyp, } x_1 \uparrow \hookrightarrow A) \\ &\hookrightarrow A^{k-1} \times (A/x_1)^\oplus + (A^{k-1} \times \sum_{j=2}^k (A/x_j)^\oplus) && \text{(\mathbf{Unit} } \hookrightarrow A^{k-1}) \\ &\equiv A^{k-1} \times \sum_{j=1}^k (A/x_j)^\oplus \end{aligned}$$

□

In particular, the previous reflections can be used to obtain $Q_m/q \equiv Q_{m-1}$ and $\mathbb{N}/n \hookrightarrow Q_n$.

Reflecting residuals into ordinals

Any ordinal $\alpha < \omega^{\omega^\omega}$ can be written as

$$\alpha = \bigoplus_{i=1}^n \bigotimes_{j=1}^{k_i} \omega^{\omega^{d_{ij}}}$$

for some $n, k_i, d_{ij} \geq 0$. Thus, we can define for every such α the wpo

$$C(\alpha) = \sum_{i=1}^n \prod_{j=1}^{k_i} (\mathbb{N}^{d_{ij}})^\oplus$$

For instance, $C(w \cdot 2 + 1) = \mathbb{N} + \mathbb{N} + \mathbf{Unit}$, and $C(\omega^{\omega^2 + \omega} + 2) = (\mathbb{N}^2)^\oplus \times \mathbb{N}^\oplus + Q_2$. Conversely, we can map wpos to ordinals via their order type [14, 20, 24], given by:

- $o(Q_m) = m$,
- $o(A \times B) = o(A) \otimes o(B)$,
- $o(A + B) = o(A) \oplus o(B)$,
- $o(A^\oplus) = \omega^{o(A)}$.

As a consequence, we have $o(\mathbb{N}) = \omega$, $o(\mathbb{N}^d) = \omega^d$ and $o((\mathbb{N}^d)^\oplus) = \omega^{\omega^d}$. For two wpos A and B it holds that $A \hookrightarrow B$ implies $o(A) \leq o(B)$ [14, 20].

Lemma 1. *C and o are bijective inverses, compatible with sums and products.*

Let us look again at Eq. 5 in Prop. 2. This equation can be further refined when $|\{x_1, \dots, x_m\}| < n$ (for some $n > 1$), which implies $m < n$ and $|x_i| < n$. In that case we have

$$(\mathbb{N}^d)^\oplus / \{x_1, \dots, x_m\} \hookrightarrow \mathbb{N}^{d \cdot (n-2)} \times (\mathbb{N}^{d-1} \cdot d \cdot (n-1))^\oplus \cdot (n-1)$$

Notice that if $n = 1$ then $m = 0$, and $(\mathbb{N}^d)/\emptyset \equiv \emptyset$. The ordinal type of the right hand-side of the previous reflection can be computed as follows:

$$o(\mathbb{N}^{d \cdot (n-2)} \times (\mathbb{N}^{d-1} \cdot d \cdot (n-1))^\oplus \cdot (n-1)) = \omega^{\omega^{d-1} \cdot d \cdot (n-1) + d \cdot (n-2)} \cdot (n-1) \quad (4)$$

This, together with the residuals in Prop. 1, prompts us to the following definition:

Definition 3 (Derivatives). Given $n > 1$ we define:

$$D_n(\omega^{\omega^d}) = \begin{cases} \omega^{\omega^{d-1} \cdot d \cdot (n-1) + d \cdot (n-2)} \cdot (n-1) & \text{if } d > 0 \\ n-1 & \text{if } d = 0 \end{cases}$$

$$D_n(\omega^{\omega^{d_1} + \dots + \omega^{d_k}}) = \bigoplus_{j=1}^k (D_n(\omega^{\omega^{d_j}}) \otimes \bigotimes_{l \neq j} \omega^{\omega^{d_l}})$$

$$\partial_n \left(\sum_{i=1}^m \omega^{\beta_i} \right) = \{ D_n(\omega^{\beta_i}) \oplus \sum_{l \neq i} \omega^{\beta_l} \mid i = 1, \dots, m \}$$

As an example, we have $D_n(1) = 0$, $\partial_n(m) = \{m-1\}$, $D_n(\omega) = n-1$ and $\partial_n(\omega) = \{n-1\}$.

Proposition 2. Let $n > 1$ and $x \in A_{<n}$ for some A . There is $\alpha' \in \partial_n o(A)$ such that $A/x \hookrightarrow C(\alpha')$.

PROOF. Let $A \equiv \sum_{i=1}^m \prod_{j=1}^{k_i} (\mathbb{N}^{d_{ij}})^{\oplus}$.

Case 1: $k_i = 0$ for every $i \in \{1, \dots, m\}$, i.e., $A \equiv Q_m$. Analogous to [21, Th. 4.1].

Case 2: $m = 1 = k_1$, i.e., $A \equiv (\mathbb{N}^d)^{\oplus}$. If $d = 0$ then $A \equiv \mathbb{N}$, and we have $o(A) = \omega$, so that $\partial_n(o(A)) = \{n-1\}$. Since $\mathbb{N}/x \hookrightarrow Q_x$, $Q_x \hookrightarrow Q_{n-1}$ (since $x < n$) and $C(n-1) = Q_{n-1}$ we are done.

Assume now $d > 0$, so that $o(A) = \omega^{\omega^d}$ and $\partial_n(o(A)) = \{\alpha'\}$ with $\alpha' = \omega^{\omega^{d-1} \cdot d \cdot (n-1) + d \cdot (n-2)} \cdot (n-1)$. By Eq. 4 we have that $C(\alpha') = \mathbb{N}^{d \cdot (n-2)} \times (\mathbb{N}^{d-1} \cdot d \cdot (n-1))^{\oplus} \cdot (n-1)$. By Prop. 1 we know that $A/x \hookrightarrow C(\alpha')$, and we are done.

Case 3: $m = 1 < k_1$, i.e., A is a product $\prod_{i=1}^k (\mathbb{N}^{d_i})^{\oplus}$. Analogous to [21, Th. 4.1].

Case 4: $m > 1$, i.e., $A \equiv \sum_{i=1}^m A_i$, where A_i is a product (in case 3). Analogous to [21, Th. 4.1]. \square

Definition 4. Let $\alpha < \omega^{\omega^\omega}$ and g a smooth function. If $\alpha = 0$ we define $M_{g,\alpha}(n) = 0$. Otherwise:

$$M_{g,\alpha}(n) = \max_{\alpha' \in \partial_n \alpha} \{1 + M_{g,\alpha'}(g(n))\}$$

Proposition 3. $L_{g,A}(n) \leq M_{g,o(A)}(n)$.

PROOF. Analogous to [21], using the descent equation and Prop. 2. \square

We now bound $M_{g,\alpha}$, using the *Cichoń* hierarchy $(h_\alpha : \mathbb{N} \rightarrow \mathbb{N})_\alpha$ for the control function $h : \mathbb{N} \rightarrow \mathbb{N}$, given by

$$h_0(x) = 0, \quad h_{\alpha+1}(x) = 1 + h_\alpha(h(x)), \quad h_\lambda(x) = h_{\lambda_x}(x)$$

Cichoń's functions measure the length of Hardy computations, meaning that the norm of the elements in an h, n -controlled sequence of length bounded by $h_\alpha(n)$ is in turn bounded by $h^\alpha(n)$. Another property of Cichoń's functions that we will use is the following [21]: For $\alpha > 0$

$$h_\alpha(x) = 1 + h_{P_x(\alpha)}(h(x)) \quad (5)$$

We say α is k -lean if $\alpha = \omega^{\beta_1} \cdot c_1 + \dots + \omega^{\beta_m} \cdot c_m$ with $c_i \leq k$ and β_i is k -lean for every $i \in \{1, \dots, m\}$.

Proposition 4. *Let $\alpha < \omega^{\omega^{d+1}}$ be k -lean. If $\alpha' \in \partial_n \alpha$ then α' is knd -lean.*

PROOF. We first prove that if $\beta < \omega^{d+1}$ is k -lean then $D_n(\omega^\beta)$ is $kd \cdot (n-1)$ -lean. Let $\beta = \sum_{i=1}^m \omega^{d_i} \cdot c_i$ with $d_i \leq d$ and $c_i \leq k$. By Def. 3 we have

$$\begin{aligned} D_n(\omega^\beta) &= \bigoplus_{i=1}^m \left(D_n(\omega^{\omega^{d_i}}) \cdot c_i \otimes \omega^{\omega^{d_i} \cdot (c_i-1)} \otimes \bigotimes_{l \neq i} \omega^{\omega^{d_l} \cdot c_l} \right) = \\ &= \bigoplus_{i=1}^m \left(\omega^{\omega^{d_i-1} \cdot d_i \cdot (n-1) + d_i \cdot (n-2)} \cdot (n-1) \cdot c_i \otimes \omega^{\omega^{d_i} \cdot (c_i-1)} \otimes \bigotimes_{l \neq i} \omega^{\omega^{d_l} \cdot c_l} \right) = \\ &= \bigoplus_{i=1}^m (\omega^{\beta_i} \cdot c_i \cdot (n-1)) \end{aligned}$$

with $\beta_i = (\omega^{d_i-1} \cdot d_i \cdot (n-1) + d_i \cdot (n-2)) \oplus \omega^{d_i} \cdot (c_i-1) \otimes \bigotimes_{l \neq i} \omega^{d_l} \cdot c_l$.

The coefficients in $D_n(\omega^\beta)$ are $c_i \cdot (n-1) \leq k \cdot (n-1) \leq kd \cdot (n-1)$. Moreover, each β_i is also $kd \cdot (n-1)$ -lean. Indeed, the maximal coefficient is obtained when $d_i = 1$ and $d_m = 0$, which is $d_i \cdot (n-1) + d_i \cdot (n-2) + c_m < 2d_i \cdot (n-1) + k \leq kd \cdot (n-1)$ (for $k, d \geq 2$).

Now we see that if $\alpha' \in \partial_n \alpha$ then α' is knd -lean. Assume $\alpha = \gamma \oplus \omega^\beta$, so that $\alpha' = \gamma \oplus D_n(\omega^\beta)$. On the one hand, γ is k -lean because α is. We have just seen that $D_n(\omega^\beta)$ is $kd \cdot (n-1)$ -lean. Then $\gamma \oplus D_n(\omega^\beta)$ is $k + kd \cdot (n-1)$ -lean, and since $k + kd \cdot (n-1) \leq kdn$, we are done. \square

Proposition 5. *Let $\alpha, \alpha' < \omega^{\omega^{d+1}}$ and h smooth. If α is k -lean and $\alpha' \in \partial_n \alpha$ then for all $x \geq knd$, $h_{\alpha'}(x) \leq h_{P_{knd}(\alpha)}(x)$.*

PROOF. Analogous to [21, Cor. B3], using Prop. 4. \square

Corollary 1. *Let $\alpha < \omega^{\omega^{d+1}}$. If α is k -lean then $M_{g,\alpha}(n) \leq h_\alpha(knd)$, where $h(x) = x \cdot g(x)$.*

PROOF. We proceed by induction on α . If $\alpha = 0$ then $M_0(n) = 0 = h_0(knd)$. Assume now $\alpha > 0$. By Def. 4 there is $\alpha' \in \partial_n \alpha$ such that $M_\alpha(n) = 1 + M_{\alpha'}(g(n))$. Since α is k -lean, by Prop. 4 we know that α' is knd -lean. Also $\alpha' < \alpha$, so we can apply the induction hypothesis.

$$\begin{aligned}
M_{g,\alpha}(n) &= 1 + M_{g,\alpha'}(g(n)) \\
&\leq 1 + h_{\alpha'}(knd \cdot d \cdot g(n)) && \text{(ind. hyp.)} \\
&\leq 1 + h_{\alpha'}(knd \cdot g(knd)) && (h_{\alpha'} \text{ monotonic, } g \text{ smooth}) \\
&= 1 + h_{\alpha'}(h(knd)) && \text{(def. of } h) \\
&\leq 1 + h_{P_{knd}(\alpha)}(h(knd)) && \text{(prev. result, } h(knd) \geq knd) \\
&= h_\alpha(knd) && \text{(Eq. 5)}
\end{aligned}$$

□

By Prop. 3, as a corollary, we obtain the following:

Theorem 1. *Let A be a normed wpo obtained by the grammar in Eq. 3 with maximal order type α . If $\alpha < \omega^{\omega^{d+1}}$ and α is k -lean then $L_{g,A}(n) \leq h_\alpha(kdn)$, where $h(x) = x \cdot g(x)$.*

4. Unordered Data Nets

Now we define Unordered Data Nets (UDN) and show some computations that can be done with them, which will be used later in Sect. 5. Instead of defining UDN as a subclass of Data Nets, we do it from scratch (as hinted by in [15]), thus obtaining a definition that is simpler than the one in [15], though equivalent. In the next sections we denote the null tuple $(0, \dots, 0) \in \mathbb{N}^k$ (for any k) as $\mathbf{0}$. We fix an infinite set Var of variables, and a special variable $\mathcal{R} \in Var$ (that will be used in broadcasts).

Definition 5 (Unordered Data Nets). *A k -dimensional Unordered Data Net (UDN_k) is a tuple $N = (P, T, F, G, H)$ where:*

- $P = \{p_1, \dots, p_k\}$ is a finite set of places,
- T is a finite set of transitions, with $P \cap T = \emptyset$,

- For each $t \in T$, there is a finite set $Var(t) \subseteq Var$ with $\mathcal{R} \in Var(t)$ such that:
 - $F_t : Var(t) \rightarrow \mathbb{N}^k$ is the subtraction function, such that $F_t(\mathcal{R}) = \mathbf{0}$,
 - $H_t : Var(t) \rightarrow \mathbb{N}^k$ is the addition function,
 - $G_t : Var(t) \times Var(t) \rightarrow \mathbb{N}^{k \times k}$ is the whole-place operations function, assigning a $k \times k$ -matrix to each (x, y) .

A marking m of a PDN can be seen as a mapping m that maps every place p to a multiset of identities. This will be the intuition that will guide our graphical notations. However, in the technical developments we represent markings as multisets of tuples. Intuitively, each tuple (v_1, \dots, v_k) represents a name carried by v_i tokens in p_i , for each $i \in \{1, \dots, k\}$.

Definition 6 (Markings, 0-expansions/contractions). A marking of a UDN_k is any element of $(\mathbb{N}^k \setminus \{\mathbf{0}\})^\oplus$. We say $m \in (\mathbb{N}^k)^\oplus$ is a **0-expansion** of a marking m' (or m' is the **0-contraction** of m) if $m = m' + \{\mathbf{0}^l\}$ for some $l \geq 0$.

In order to deal with identities that are not present in m , or identities that are removed due to the firing of t , we introduce/remove null vectors where needed, via **0-expansions/contractions**.

Next, we define the semantics of UDN . The firing of a transition $t \in T$ with $Var(t) = \{x_1, \dots, x_n\}$ happens in the following steps:

- for each $i \in \{1, \dots, n\}$, $x_i \in Var(t)$ is instantiated to a name/tuple v_i so that $F_t(x_i) \leq v_i$ (there are enough tokens carrying that name). Some of the selected tuples may be $\mathbf{0}$ thanks to **0-expansions**. Some other names/tuples $v_{n+1}, \dots, v_{n'}$ are not selected. We will refer to v_i as the i -selected tuple when $i \leq n$, or as the non-selected i -tuple, when $i > n$,
- tokens are removed from each selected tuple, obtaining $v_i - F_t(x_i)$ for each $i \in \{1, \dots, n\}$,
- whole-place operations are done via the matrices G_t . Intuitively, $G_t(x, y)$ specifies how x -tokens contribute to the presence of new y -tokens. More precisely:

- In the case of the j -th selected name, for each i -selected token in p_k we consider $G_t(x_i, x_j)(k, l)$ j -tokens in p_l , and for each non-selected token in p_k we consider $G_t(\mathcal{R}, x_j)(k, l)$ j -tokens in p_l .
- In the case of the non-selected j -tuple, for each i -selected token in p_k we consider $G_t(x_i, \mathcal{R})(k, l)$ j -tokens in p_l , and for each j -token in p_k we consider $G_t(\mathcal{R}, \mathcal{R})(k, l)$ j -tokens in p_l .
- $H_t(x_i)$ i -tokens are added, and for each non-selected j -name we add $H_t(\mathcal{R})$ j -tokens,
- finally, $\mathbf{0}$ -tuples are removed.

Let us see the definition of the semantics of *UDN* formally.

Definition 7. Let m_1 be a marking of N and $t \in T$ with $\text{Var}(t) = \{x_1, \dots, x_n\}$. A marking m_2 can be reached from m_1 by firing t as follows:

1. Let $m'_1 = \{v_1, \dots, v_n, v_{n+1}, \dots, v_{n'}\}$ be a $\mathbf{0}$ -expansion of m_1 , such that:
 - $F_t(x_i) \leq v_i$ for $0 < i \leq n$, and
 - $v_i \neq \mathbf{0}$ for $n < i \leq n'$.
2. If we let $x_j = \mathcal{R}$ for $n < j \leq n'$, take
 - $v'_j = \sum_{i=1}^{n'} (v_i - F_t(x_i)) * G_t(x_i, x_j) + H_t(x_j)$ for $0 < j \leq n$,
 - $v'_j = \sum_{i=1}^n (v_i - F_t(x_i)) * G_t(x_i, x_j) + v_j * G_t(\mathcal{R}, \mathcal{R}) + H_t(\mathcal{R})$ for $n < j \leq n'$,
3. Then, m_2 is the $\mathbf{0}$ -contraction of $\{v'_1, \dots, v'_{n'}\}$.

Example 1. Let us illustrate the previous definitions by means of an example of a *UDN* N with a single transition t and two places $\{p_1, p_2\}$. In order to specify N we need to define the functions F_t , G_t and H_t . We define $\text{Var}(t) = \{x_1, x_2, \mathcal{R}\}$ and take:

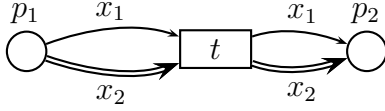
$$\begin{array}{lll}
F_t(x_1) = (1, 0) & F_t(x_2) = (0, 0) & F_t(\mathcal{R}) = \mathbf{0} \\
H_t(x_1) = (0, 0) & H_t(x_2) = (1, 0) & H_t(\mathcal{R}) = (0, 1) \\
G_t(x_1, x_1) = \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix} & G_t(x_1, x_2) = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} & G_t(x_1, \mathcal{R}) = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \\
G_t(x_2, x_1) = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} & G_t(x_2, x_2) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & G_t(x_2, \mathcal{R}) = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \\
G_t(\mathcal{R}, x_1) = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} & G_t(\mathcal{R}, x_2) = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} & G_t(\mathcal{R}, \mathcal{R}) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}
\end{array}$$

Notice that $F_t(\mathcal{R}) = \mathbf{0}$ is forced by Def. 7 (otherwise the obtained transition system would not be monotonic). Let us see how the firing of t works for a marking $m_1 = \{(2, 0), (1, 0)\}$. Intuitively, this marking has three tokens in p_1 of two different identities. In order to fire t we consider the $\mathbf{0}$ -expansion of m_1 , $m'_1 = \{v_1, v_2, v_3\}$, where $v_1 = (1, 0)$, $v_2 = (0, 0)$ and $v_3 = (0, 1)$. Instead of directly using the two equations in Def. 7 we split them obtaining intermediate tuples v''_i and v'''_i for each $i \in \{1, 2, 3\}$. We first consider v''_1 , v''_2 and v''_3 , obtained respectively from v_1 , v_2 and v_3 by removing tokens according to F_t .

$$\begin{aligned}
v''_1 &= v_1 - F_t(x_1) = (1, 0) \\
v''_2 &= v_2 - F_t(x_2) = (0, 0) \\
v''_3 &= v_3 - F_t(\mathcal{R}) = (1, 0)
\end{aligned}$$

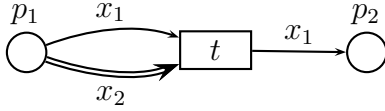
Second, we consider the whole-place operations specified by G_t :

$$\begin{aligned}
v'''_1 &= v''_1 * G_t(x_1, x_1) + v''_2 * G_t(x_2, x_1) + v''_3 * G_t(\mathcal{R}, x_1) = \\
& (1, 0) * \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix} + (0, 0) * \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} + (1, 0) * \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} = (0, 0) \\
v'''_2 &= v''_1 * G_t(x_1, x_2) + v''_2 * G_t(x_2, x_2) + v''_3 * G_t(\mathcal{R}, x_2) = \\
& (1, 0) * \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} + (0, 0) * \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + (1, 0) * \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} = (1, 0) \\
v'''_3 &= v''_1 * G_t(x_1, \mathcal{R}) + v''_2 * G_t(x_2, \mathcal{R}) + v''_3 * G_t(\mathcal{R}, \mathcal{R}) = \\
& (1, 0) * \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} + (0, 0) * \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} + (1, 0) * \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = (1, 0)
\end{aligned}$$



$$\begin{aligned}
F_t(x_1) &= (1, 0), F_t(x_2) = (0, 0), F_t(\mathcal{R}) = (0, 0) \\
H_t(x_1) &= (0, 1), H_t(x_2) = (0, 0), H_t(\mathcal{R}) = (0, 0) \\
G_t(x_1, x_1) &= \mathbf{Id}, G_t(x_1, x_2) = G_t(x_1, \mathcal{R}) = \mathbf{0} \\
G_t(x_2, x_1) &= \mathbf{0}, G_t(x_2, x_2) = \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix}, G_t(x_2, \mathcal{R}) = \mathbf{0} \\
G_t(\mathcal{R}, x_1) &= G_t(\mathcal{R}, x_2) = \mathbf{0}, G_t(\mathcal{R}, \mathcal{R}) = \mathbf{Id}
\end{aligned}$$

Figure 1: Selective transfer



$$\begin{aligned}
F_t(x_1) &= (1, 0), F_t(x_2) = (0, 0), F_t(\mathcal{R}) = (0, 0) \\
H_t(x_1) &= (0, 1), H_t(x_2) = (0, 0), H_t(\mathcal{R}) = (0, 0) \\
G_t(x_1, x_1) &= \mathbf{Id}, G_t(x_1, x_2) = G_t(x_1, \mathcal{R}) = \mathbf{0} \\
G_t(x_2, x_1) &= \mathbf{0}, G_t(x_2, x_2) = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}, G_t(x_2, \mathcal{R}) = \mathbf{0} \\
G_t(\mathcal{R}, x_1) &= G_t(\mathcal{R}, x_2) = \mathbf{0}, G_t(\mathcal{R}, \mathcal{R}) = \mathbf{Id}
\end{aligned}$$

Figure 2: Selective reset

Finally, we add tokens according to H_t :

$$\begin{aligned}
v'_1 &= v'''_1 + H_t(x_1) = (0, 0) \\
v'_2 &= v'''_2 + H_t(x_2) = (2, 0) \\
v'_3 &= v'''_3 + H_t(\mathcal{R}) = (1, 1)
\end{aligned}$$

and take m_2 as the $\mathbf{0}$ -contraction of $\{v'_1, v'_2, v'_3\}$, that is, $m_2 = \{(2, 0), (1, 1)\}$. This marking can be intuitively seen as one with four tokens, two of them carrying a given name, twice in p_1 , and the other two carrying another name, one in p_1 and one in p_2 .

In figures, we will use standard conventions: places are represented as circles and transitions by squares. For each $x \in \text{Var}(t)$, with $F_t(x) = (n_1, \dots, n_k)$, we draw an arrow from p_i to t labeled by n_i occurrences of x (no arrow is drawn if every such n_i is null for every x). The representation of H_t is analogous. The graphical conventions for whole-place operations will be discussed next. Actually, it is enough for our purposes in Sect. 5 to perform operations like transfers, resets, copies (among the selected tuples or not), or name creation, that can be done as follows.

Selective reset/transfer/copies.

Next we denote as \mathbf{Id} and $\mathbf{0}$ the identity and the null matrices, respectively. A transition t in an UDN can perform whole-place operations dealing

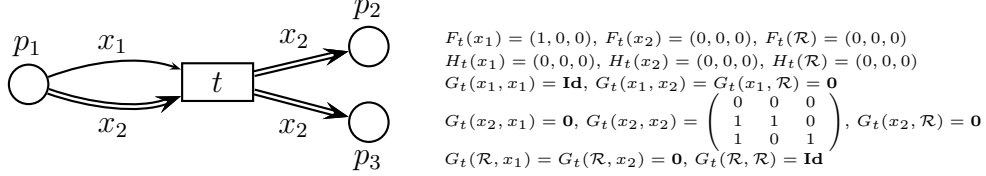


Figure 3: Selective copy

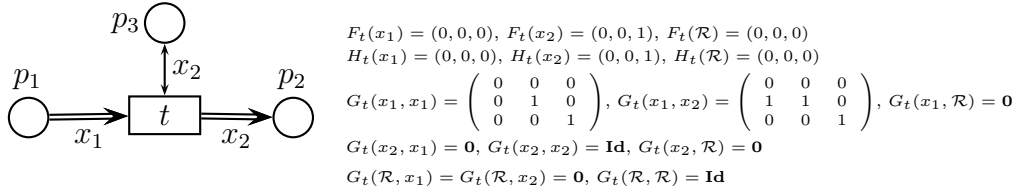


Figure 4: Selective transfer (II): transferring x_1 -tokens as x_2 -tokens

with one of the selected tuples.⁴ In general, to perform whole-place operations only for x -tokens (tokens carrying the name to which x is instantiated), it is enough to set $G_t(y, z) = \mathbf{0}$ for every $y \neq z$, $G_t(y, y) = \mathbf{Id}$ for every $y \neq x$, and $G_t(x, x)$ accordingly (depending on the specific whole-place operation). For instance, it can transfer all the x -tokens from one place to another, as shown in the *UDN* in Fig. 1; remove all the x -tokens from a given place, as shown in Fig. 2; or copying them in different places, as shown in Fig. 3.

In figures, we denote by double arrows these whole-place operations, labeled with variables. More specifically, an x -labeled double arrow from p to t , together with an x -labeled double arrow from t to q , represents the transfer of all the x -tokens from p to q (see Fig. 1 or transition $st(eq)$ in Fig. 8). If there is also an x -labeled double arrow from t to a different place q' , they represent the copy of the x -tokens in p both to q and q' (see Fig. 3 or transition $st(neq)$ in Fig. 10). A double arrow labeled by x from p to t , such that there is no x -labelled outgoing double arrow, represents the reset of the x -tokens in p (see Fig. 2 or transition $end(neq)$ in Fig. 11).

So far, we are assuming that whole-place operations deal only with one specific name, so that $G_t(x, y)$ is the null matrix if $x \neq y$. We can also copy the x_1 -tokens in p as x_2 -tokens in q , by setting $G_t(x_1, x_2)(p, q) = 1$. Fig. 4 depicts a *UDN* that transfers all the x_1 -tokens in p_1 to p_2 as x_2 -tokens. In figures, this operation will be represented by having an x_1 -labelled double arc

⁴These operations correspond to the whole-place operations in affine nets [10].

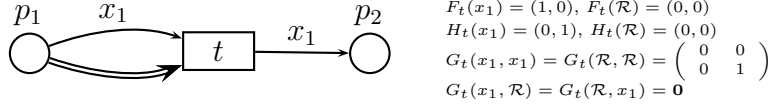


Figure 5: Reset broadcast

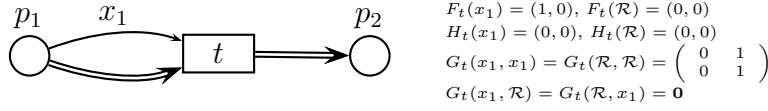


Figure 6: Transfer broadcast

from p to t , together with an x_2 -labelled double arc from t to q (see Fig. 4 or transition $st(R22)$ in Fig. 13).

Reset/transfer broadcast.

Let us see how we can implement an operation in which a process/tuple instructs the rest of processes to reset a given place. Figure 5 depicts a *UDN* that can reset place p_1 whenever there is any token in p_1 and moves that token to p_2 . Similarly, we can have transfer broadcasts, in which every tuple transfers the tokens from one place to another. Fig. 6 shows a *UDN* that whenever there is a token in p_1 can transfer all the tokens in p_1 (except the one to which x_1 is instantiated) to p_2 .

In figures, a double arrow from p to t , together with a double arrow from t to q (without any label), represents the transfer broadcast of all the tokens from p to q (see Fig. 6 or transition $start$ in Fig. 7). If there are no such outgoing arrows, it represents the reset of place p (like in Fig. 5 or transition end in Fig. 7).

(Lossy) name creation.

As discussed in [15], *UDN* do not have a primitive for name/tuple creation. Actually, [3] defines an extension of Petri Data Nets that does contain that primitive. We here show that name creation can already be obtained in *UDN*, though in a lossy way, thanks to selective resets, which is enough for our purposes.

Name creation can be achieved by a transition t in a lossy way, as follows: Let $F_t(x) = \mathbf{0}$, $H_t(x) \neq \mathbf{0}$ and $G_t(y, x) = \mathbf{0}$ for all $y \in \text{Var}(t)$. Then, either (i) x is instantiated to a new tuple $\mathbf{0}$ in the $\mathbf{0}$ -expansion, in which case a new tuple $H_t(x)$ is created, or (ii) x is instantiated to an already existing tuple, but this tuple is first reset due to G_t . In figures we will use a special variable $\nu \in \text{Var}(t)$, only appearing in post-arcs, to represent lossy name creations (see transition *new* in Fig. 12). Notice that lossy name creation can already be achieved in Unordered Data Nets without broadcasts (we do not make use of \mathcal{R}).

Upper bounds for Unordered Data Nets.

The state space of a d -dimensional *UDN* is $A = (\mathbb{N}^d)^\oplus$, with $o(A) = \omega^{\omega^d}$, which is d -lean. Since sequences are controlled by a function g which is primitive recursive (it is actually linear for termination and exponential for coverability [15]), we can apply Theorem 1 in the previous section, to conclude that $L_{g,A}(n)$ is bounded by $h_{\omega^{\omega^d}}(d^2 \cdot n)$. Therefore, the size of the elements in that sequence is bounded by $h^{\omega^{\omega^d}}(d^2 \cdot n)$. This gives us combinatorial algorithms that work in nondeterministic space bounded by $h^{\omega^{\omega^d}}$ (see e.g. [23] for details). Since $g(x)$ is primitive recursive so is $h(x) = x \cdot g(x)$, so that we can use h in Eq. 2 to define the \mathbf{F}_α classes. Summing up, we can conclude the following.

Theorem 2. *Termination and coverability for UDN_d are in \mathbf{F}_{ω^d} . Those problems for UDN are in $\mathbf{F}_{\omega^\omega}$.*

5. Hyper-Ackermannian lower bound

In this section we obtain $\mathbf{F}_{\omega^\omega}$ lower bounds for termination and coverability of *UDN*, by providing (weak) computations of $F_{\omega^\omega} = H^{\omega^{\omega^\omega}}$ and its inverse, using *UDN*. Let us first define the encoding of ordinals below ω^{ω^ω} using *UDN* markings. In the encoding we fix $k \in \mathbb{N}$. For a tuple $v \in \mathbb{N}^k$ we write $v = (v[0], \dots, v[k-1])$. We compare tuples in \mathbb{N}^k using the lexicographic order \leq_{lex} (identifying them with words of length k), the rightmost component being the most significant. For $0 \leq i \leq j < k$ we write $v[i..j]$ to denote the tuple $(v[i], \dots, v[j])$, and $\min(v, v') = (\min(v[0], v'[0]), \dots, \min(v[k-1], v'[k-1]))$. Also, we identify each ordinal α with $\{\beta \mid \beta < \alpha\}$.

Definition 8. *We define $\mathbf{v} : \omega^k \rightarrow \mathbb{N}^k$ as $\mathbf{v}(\omega^{k-1} \cdot c_{k-1} + \dots + \omega^0 \cdot c_0) = (c_0, \dots, c_{k-1})$, and $\mathbf{C} : \omega^{\omega^k} \rightarrow (\mathbb{N}^k)^\oplus$ as $\mathbf{C}(\omega^{\beta_1} + \dots + \omega^{\beta_l}) = \{\mathbf{v}(\beta_1), \dots, \mathbf{v}(\beta_l)\}$.*

We also define the converse mappings.

Definition 9. We define $\beta : \mathbb{N}^k \rightarrow \omega^k$ as $\beta(v) = \omega^{k-1} \cdot v[k-1] + \dots + \omega^0 \cdot v[0]$ and $\alpha : (\mathbb{N}^k)^\oplus \rightarrow \omega^{\omega^k}$ as $\alpha(\{v_1, \dots, v_l\}) = \omega^{\beta_1} + \dots + \omega^{\beta_l}$, where $\beta_1 \geq \dots \geq \beta_l$ is a reordering of $\beta(v_1), \dots, \beta(v_l)$.

Proposition 6. The functions \mathbf{C} and α are inverse bijections. Moreover, the encoding is robust, i.e., $m_1 \leq m_2$ implies $\alpha(m_1) \leq \alpha(m_2)$.

PROOF. Clearly \mathbf{C} and α are inverse bijections, because \mathbf{v} and β are. The encoding is robust because dropping a tuple from $\alpha(m)$ means losing a summand in the CNF of α , and decreasing some value in v means losing a summand in the (non-strict) CNF of $\beta(v)$. \square

In order to compute the elements in the fundamental sequence of limit ordinals, we need to process (the encoding of) the minimal ordinal that appears as exponent. This ordinal is represented by the minimal tuple, according to the lexicographic order.

Lemma 2. Let $\beta_1, \beta_2 < \omega^k$. Then $\beta_1 < \beta_2$ iff $\mathbf{v}(\beta_1) <_{lex} \mathbf{v}(\beta_2)$.

Let us see how Hardy computations are mimicked under this encoding of ordinals. The tail recursive definitions of the Hardy hierarchy can be seen as the following rewrite system:

$$(H1) \quad \alpha + 1, n \rightarrow_H \alpha, n + 1 \qquad (H2) \quad \lambda, n \rightarrow_H \lambda_n, n$$

Clearly, $\alpha, n \rightarrow_H \alpha', n'$ implies $H^\alpha(n) = H^{\alpha'}(n')$ and, in particular, if $\alpha, n \rightarrow_H^* 0, n'$ then $H^\alpha(n) = n'$. The next result tells us the form of encodings of successor and limit ordinals.

Proposition 7. Let $\beta < \omega^k$ and $\alpha < \omega^{\omega^k}$.

- β is a successor ordinal iff $\mathbf{v}(\beta)[0] > 0$.
- α is a successor ordinal iff $\mathbf{0} \in \mathbf{C}(\alpha)$.

PROOF. For the first item, $\beta = \sum_{i=1}^{k-1} \omega^i \cdot c_i$ is a successor iff $c_0 = \mathbf{v}(\beta)[0] > 0$ (since c_0 is the coefficient of ω^0). For the second item, $\alpha = \gamma + \omega^\beta$ is a successor iff $\beta = 0$ iff $\mathbf{v}(\beta) = \mathbf{0} \in \mathbf{C}(\alpha)$. \square

Most of the work in doing Hardy computations is in the computations of the elements in the fundamental sequence. The next two lemmas show how to do such computations under our encoding.

Lemma 3. *Let $v = (0, \dots, 0, c_i, \dots, c_{k-1})$ with $i = \min\{j \mid c_j > 0\} > 0$, be the encoding of a limit ordinal $\beta < \omega^k$. Then for any $n > 0$ we have $\mathbf{v}(\beta_n) = (0, \dots, n, c_i - 1, c_{i+1}, \dots, c_{k-1})$. As a consequence, if $m = m' + \{v\}$ is the encoding of $\alpha = \gamma + \omega^\beta$ (with $v = \min_{lex} m$) then $\mathbf{C}(\alpha_n) = m' + \{(0, \dots, n, c_i - 1, c_{i+1}, \dots, c_{k-1})\}$.*

PROOF. Let $\beta = \gamma + \omega^i$, where $\gamma = \omega^{k-1} \cdot c_{k-1} + \dots + \omega^{i+1} \cdot c_{i+1} + \omega^i \cdot (c_i - 1)$. By Eq. 1 (left), $\beta_n = \gamma + \omega^{i-1} \cdot n = \omega^{k-1} \cdot c_{k-1} + \dots + \omega^{i+1} \cdot c_{i+1} + \omega^i \cdot (c_i - 1) + \omega^{i-1} \cdot n$, and the thesis follows. The last sentence follows from Eq. 1 (right). \square

Lemma 4. *Let $m = m' + \{v\}$ with $v = (c_0, \dots, c_{k-1}) = \min_{lex} m$, be the encoding of a limit ordinal $\alpha = \gamma + \omega^{\beta+1} < \omega^{\omega^k}$ and let $n > 0$. Then $\mathbf{C}(\alpha_n) = m' + \{(c_0 - 1, c_1, \dots, c_{k-1})^n\}$.*

PROOF. By hypothesis, $\mathbf{C}(\gamma) = m'$ and $\mathbf{v}(\beta + 1) = (c_0, \dots, c_{k-1})$, so that $\mathbf{v}(\beta) = (c_0 - 1, c_1, \dots, c_{k-1})$. Then, $\alpha_n = (\gamma + \omega^{\beta+1})_n = \gamma + \omega^\beta + \dots + \omega^\beta$, so that $\mathbf{C}(\alpha_n) = \mathbf{C}(\gamma) + \{\mathbf{v}(\beta), \dots, \mathbf{v}(\beta)\} = m' + \{\mathbf{v}(\beta)^n\}$, and we are done. \square

The two previous lemmas justify the definition of the following rewriting system.

Definition 10. *We define R as the following rewriting system over $(\mathbb{N}^k)^\oplus$:*

- (R1) $\{\mathbf{0}\}, n \rightarrow \emptyset, n + 1$
- (R21) $\{(c_0, c_1, \dots, c_{k-1})\}, n \rightarrow \{(c_0 - 1, c_1, \dots, c_{k-1})^n\}, n$ if $c_0 > 0$
- (R22) $\{(0, \dots, 0, c_i, \dots, c_{k-1})\}, n \rightarrow \{(0, \dots, n, c_i - 1, \dots, c_{k-1})\}, n$
if $i = \min\{j \mid c_j > 0\} > 0$
- (Min) $\{v\}, n \rightarrow M', n' \Rightarrow M, n \rightarrow_R (M - \{v\}) + M', n'$ if $v = \min_{lex} M$

(R1) is the counterpart of rule (H1), dealing with successor ordinals. (R21) and (R22) are both the counterpart of rule (H2): (R21) deals with limit ordinals $\gamma + \omega^\beta$ when β is a successor, and (R22) is the rule for limit ordinals when β is also a limit. The last rule (Min) states that rewritings must happen for tuples representing minimal ordinals, that is, for tuples that are minimal for the lexicographic order (see Lemma 2).

Proposition 8. *If $M, n \rightarrow_R M', n'$ then $H^{\alpha(M)}(n) = H^{\alpha(M')}(n')$.*

In particular, since $\alpha(\emptyset) = 0$, if $\mathbf{C}(\alpha), n \rightarrow_R^* \emptyset, n'$ then $H^\alpha(n) = n'$. We denote by (R^{-1}) the rewriting system obtained by inverting the rules of (R) . In the rest of the section we will implement the rules of (R) and (R^{-1}) using Unordered Data Nets. This actually cannot be done (we would need a Turing-complete formalism) but it can be done in a weak (or lossy) sense. For that purpose, we next define lossy versions of (R) and (R^{-1}) .

Lossy rewriting system

We now consider lossy versions of (R) and (R^{-1}) , that we denote as (R_l) and (R_l^{-1}) , respectively. Both are obtained with the rule

$$\frac{M_1, n_1 \geq M'_1, n'_1 \rightarrow_X M'_2, n'_2 \geq M_2, n'}{M_1, n_1 \rightarrow_{X_l} M_2, n_2}$$

where X is either R or R^{-1} , and $M, n \geq M', n'$ simply means $M \geq M'$ and $n \geq n'$.

Let us see that the lossy rewriting systems perform Hardy computations in a weak sense. First, a definition.

Definition 11 (Structural order). *We define the structural order \sqsubseteq between ordinals in ω^{ω^k} as $\alpha \sqsubseteq \alpha'$ iff $\mathbf{C}(\alpha) \leq \mathbf{C}(\alpha')$.*

The following result states that Hardy functions are monotonic with respect its ordinal parameter, when considering the structural order.⁵

Lemma 5. *If $\alpha \sqsubseteq \alpha'$ then $H^\alpha(n) \leq H^{\alpha'}(n)$.*

PROOF. It follows from the fact that $H^{\alpha_1 + \alpha_2}(n) = H^{\alpha_1}(H^{\alpha_2}(n))$ and each H^α is monotonic. First, consider $v_1, v_2 \in \mathbb{N}^k$ with $v_2 = v_1 + e_i$, where e_i is the null vector, except for a 1 in the i -th component. In this case, there are $c \in \mathbb{N}$ and $\beta_1, \beta_2 < \omega^k$ such that $\beta(v_1) = \beta_1 + \omega^i \cdot c + \beta_2$ and $\beta(v_2) = \beta_1 + \omega^i \cdot (c+1) + \beta_2$. Then, $H^{\beta(v_2)} = H^{\beta_1}(H^{\omega^i \cdot (c+1)}(H^{\beta_2}(n)))$, and since every H^α is monotonic, it is enough to prove that $H^{\omega^i \cdot (c+1)}(n) \geq H^{\omega^i \cdot c}(n)$, which is true because $H^{\omega^i \cdot (c+1)}(n) = H^{\omega^i \cdot c}(H^{\omega^i}(n)) \geq H^{\omega^i \cdot c}(n)$ (since always $H^\alpha(n) \geq n$). Then, for every $v_1 \leq v_2$ we have $v_2 = v_1 + \sum_{j=1}^{k-1} d_j \cdot e_j$ for some $d_1, \dots, d_{k-1} \in \mathbb{N}$,

⁵Remember that $\alpha \leq \alpha'$ does not imply $H^\alpha(n) \leq H^{\alpha'}(n)$ in general.

and by iterating the previous argument $d_1 + \dots + d_{k-1}$ times we conclude that $H^{\beta(v_1)}(n) \leq H^{\beta(v_2)}(n)$. Similarly, for the general case it is enough to consider that $H^{\alpha(m+\{0\})}(n) = H^{\alpha(m)+1}(n) = H^{\alpha(m)}(n+1) \geq H^{\alpha(m)}(n)$. \square

Equivalently stated, the previous lemma says that if $m_1 \leq m_2$ then $H^{\alpha(m_1)}(n) \leq H^{\alpha(m_2)}(n)$. Now we can prove that indeed, the lossy rewriting systems weakly compute H and its inverse.

Proposition 9. *The following two conditions hold:*

- If $M, n \rightarrow_{R_l} M', n'$ then $H^{\alpha(M)}(n) \geq H^{\alpha(M')}(n')$
- If $M, n \rightarrow_{R_l^{-1}} M', n'$ then $H^{\alpha(M)}(n) \geq H^{\alpha(M')}(n')$

PROOF. By definition of (R_l) and (R_l^{-1}) , it is enough to apply Prop. 8 and Lemma 5. \square

Unordered Data Nets for weak Hardy computations

Let us define \mathcal{N} and \mathcal{N}^{-1} , UDNs that compute (R_l) and (R_l^{-1}) , respectively. We need to be able to represent (multisets of) tuples of arity k . For that purpose, in both cases we have places c_0, \dots, c_{k-1} to represent such tuples. We also have a place p_n in order to represent the value of n , and a place *used*, that at all time will contain a single copy of all the used tokens (i.e., those representing tuples in the encoding). We will also use other auxiliary places, that we will describe later.

For any marking M and $a \in \text{used}$,⁶ we write $M(a)$ to denote the tuple (v_0, \dots, v_{k-1}) provided there are v_i a -tokens in c_i , for $0 \leq i < k$. Hence, any marking M represents the multiset of markings $\{M(a_1), \dots, M(a_n)\}$, provided *used* contains exactly the tokens in the set $\{a_1, \dots, a_n\}$.

The initial marking of N contains a single token in *used*, k occurrences of this token in c_{k-1} , and k tokens in p_n . This marking is the representation of $\{(0, \dots, 0, k)\}, k$, and $\{(0, \dots, 0, k)\}$ is in turn the encoding of $\omega^{\omega^{k-1} \cdot k}$. Notice that $H^{\omega^{\omega}}(k) = H^{\omega^{\omega^k}}(k) = H^{\omega^{\omega^{k-1} \cdot k}}(k)$.

Next we explain how to simulate each of the rules of (R) and (R^{-1}) in a lossy way. Since these rules must be applied to tuples that are minimal with respect to the lexicographic order, we first show the way we can select such

⁶We are abusing notation, meaning that according to M , there is an a -token in *used*.

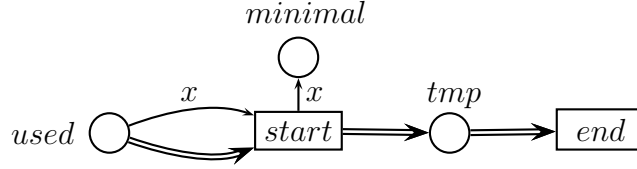


Figure 7: Start and end of the selection of the minimal tuple

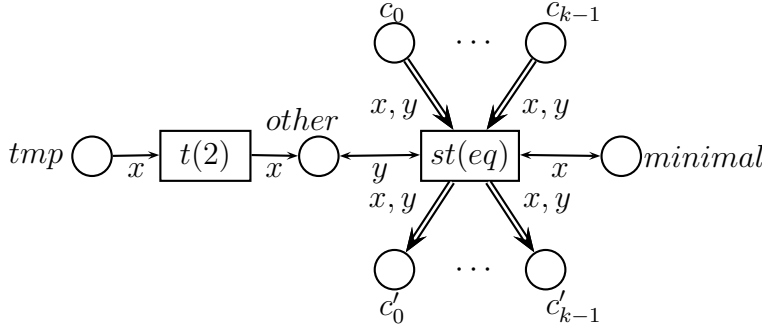


Figure 8: Minimal tuple selection: (2) and start of (3eq)

minimal tuple. Notice that this must be done even if tuples lie in a pool of unordered tuples, and no comparison can be done between the number of tokens in two places (this is actually equivalent to a zero-test). In some of the figures, we will omit some control places that guarantee that transitions fire only in the specified order.

5.1. Minimal selection

In the following, for $j \in \{0, \dots, k-1\}$ we write $v <_j v'$ when $v[i] = v'[i]$ for $i > j$ and $v[j] < v'[j]$. Then, $v \leq_{lex} v'$ iff $v = v'$ or $v <_j v'$ for some $j \in \{0, \dots, k-1\}$. The intuitive key idea to *lossily* select the minimal tuple with respect to the lexicographic order is to select any tuple v in a nondeterministic way, and consider an arbitrary number of other tuples v' , *forcing* them to be greater or equal than v , either forcing them to be equal or guessing j so that $v <_j v'$. Finally, we remove every other tuple.

Example 2. Consider the marking

$$m = \{(1, 2, 3), (2, 3, 2), (0, 3, 2), (0, 1, 2), (1, 2, 1), (2, 0, 0), (0, 0, 0)\}$$

We show a possible outcome of the selection of the minimal. Assume we select $v = (1, 2, 1)$ as the minimal one. Now we compare it to other tuples:

- We first compare it to $x = (0, 3, 2)$, forcing them to be equal. We can do this (in a lossy way) by setting both to $\min(v, x) = (0, 2, 1)$.
- Then, we compare it to $x = (0, 1, 2)$, forcing $v <_1 x$ (notice that $j = 1$ is also selected non-deterministically). This is done by setting $v = (0, 0, 1)$ and $x = (0, 1, 1)$ (we leave the 0-component as it is, we equal the 2-component as in the previous step, and for the 1-component we set $v[1] = \min(v[1], x[1] - 1)$ and $x[1] = \min(v[1], x[1])$).
- Next, we compare v to $x = (2, 3, 2)$, forcing $v <_0 x$, leaving v as it is and setting $x = (2, 0, 1)$, similarly as in the previous case.
- Finally, we remove every other tuple.

The result is the marking $m' = (2, 0, 1), (0, 2, 1), (0, 1, 1), (0, 0, 1)$, where the tuple $v = (0, 0, 1)$ is correctly selected as the minimal.

More precisely, the procedure is as follows:

- (1) Move an arbitrary token from *used* to a place *minimal* and transfer all tokens from *used* to a place *tmp* (transition *start* in Fig. 7). This amounts to selecting an arbitrary tuple $v = v_0$ that will be forced to be the minimal one.
- (2) Move an arbitrary token from *tmp* to *used* (transition $t(2)$ in Fig. 8). This amounts to selecting an arbitrary tuple x_l . Go to (3eq) or (3neq) (choice not shown in figures).
- (3eq) Set both x and v to $\min(v, x)$. This is done as follows: first fire $st(eq)$ in Fig. 8, that transfers all the tokens of *minimal* and *other* from places c_0, \dots, c_{k-1} to places c'_0, \dots, c'_{k-1} . Then fire the transitions t_i (for all $i \in \{0, \dots, k-1\}$ in Fig. 9 any number of times. Notice that at any time the number of tokens of *minimal* and *other* in each c_i coincides. Finally, fire $end(eq)$ in Fig. 9, thus emptying the c'_i places and moving the token in *other* to *used*.
Then either go to 2 or go to 4.
- (3neq) Take any $j \in \{0, \dots, k-1\}$. Then set

$$x = (x[0..j-1], x[j], \min(v[j+1..k-1], x[j+1..k-1]))$$

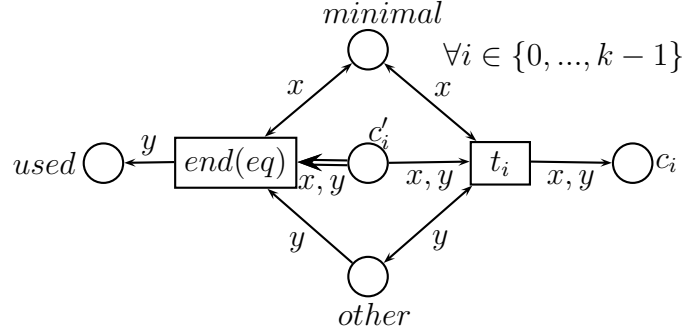


Figure 9: Minimal selection: end of (3eq)

$$v = (v[0 \dots j-1], \min(v[j], x[j] - 1), \min(v[j+1 \dots k-1], x[j+1 \dots k-1]))$$

This can be done as follows: first select any $j \in \{0, \dots, k-1\}$ (choice not shown in the figures). Then fire $st(neq)$ in Fig. 10. This transfers the tokens of *minimal* from c_j to c'_j , and *copies* the tokens of *other* from c_j to c'_j . Then fire t_j^- once (deadlock if it is not possible) and fire transition t_j in Fig. 11 any number of times, which has the analogous role of transitions t_i in Fig. 9. After these firings, the number of tokens of *minimal* in c_j is less or equal than the number of token of *other* minus one (hence, it is strictly smaller).

Regarding the components from $j+1$ to $k-1$ we proceed as in (3eq) (and Fig 8 and Fig. 9), but only for each $i \in \{j+1, \dots, k-1\}$ (instead of $i \in \{0, \dots, k-1\}$). Finally, we fire transition $end(neq)$ in Fig. 11, which empties c'_j . Then either go to 2 or go to 4.

- (4) Reset *tmp* (transition end in Fig. 7).

After the previous procedure, the token in place *minimal* identifies the minimal tuple, with respect to the lexicographic order. In order to prove that the previous construction is correct, we introduce the following notations. Let v_l denote the value of the tuple selected as *minimal* in the l -step (hence, the value selected initially is v_0). Let also x_l be the tuple selected as *other* in the l -step, and x'_l the final value of this tuple (after it has been processed). Then we have the following:

Lemma 6. $v_l = \min_{lex}\{x'_1, \dots, x'_l, v_l\}$.

PROOF. In the first place, we have $v_i \geq v_{i-1}$ for every i , since v_i is obtained from v_{i-1} by possibly removing some tokens. Since \leq_{lex} is a linearization of

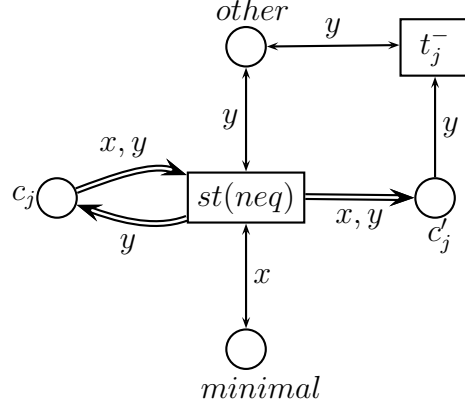


Figure 10: Minimal selection: start of (3neq)

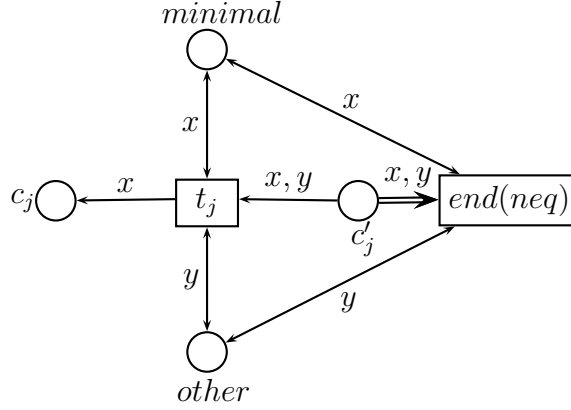


Figure 11: Minimal selection: end of (3neq)

\leq we have $v_i \geq_{lex} v_{i-1}$ for all i . Moreover, if (3eq) is chosen, then $x'_i = v_i$, so that $v_i \leq_{lex} x'_i$; if (3neq) is chosen with some j , then we have $v_i <_j x'_i$ (it cannot be $x'_i[j] = 0$, or it would have deadlocked). Thus, in any case we have $v_i \leq_{lex} x'_i$. Putting both things together, we get $v_l \leq_{lex} x'_i$ for every i , and we are done. \square

The previous selection step is a lossy step. In general, if we start from some $m = \{v, x_1, \dots, x_l\} + \bar{m}$ we may end up in any marking $m' = \{v_l, x'_1, \dots, x'_l\}$, and $m' \leq m$. However, it can be a perfect step whenever $\bar{m} = \emptyset$, $v_l = v$ and $x'_i = x_i$ for every i , which can happen when (i) v is chosen as the actual minimal tuple; (ii) (3eq) is chosen when $x_i = v_i$, and v_{i+1} is taken equal to both; (iii) (3neq) is chosen when $x_i \neq v_i$, with the proper j , and (iv) step 4 is not taken until tmp is empty. In Ex. 2, the net could choose $(0, 0, 0)$ as

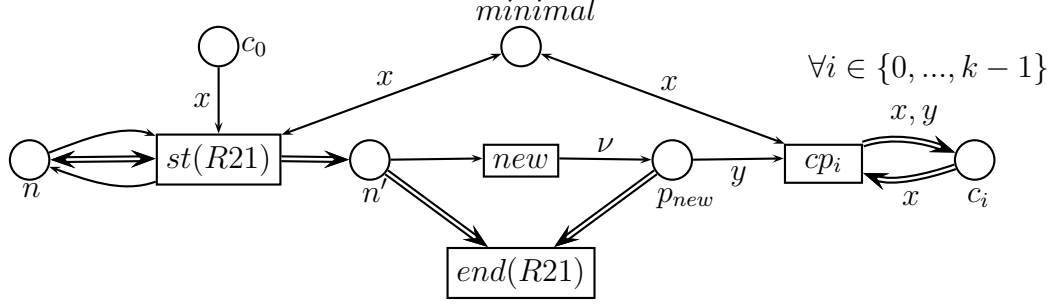


Figure 12: Simulation of (R21)

the minimal tuple, and compare it with every other tuple in the right way: e.g., $(0, 0, 0) <_1 (0, 1, 2)$.

So far, we have seen how to compute using a *UDN* the minimal tuple (according to the lexicographic order) in a multiset of tuples. Next we show how the rules in both rewriting systems can be simulated using *UDN* transitions (for which, in some cases, we will need to compute the minimal tuple).

5.2. Simulation of (R1)

In this case, since $\mathbf{0}$ is minimal according to \leq_{lex} , we do not need to start the simulation with the selection of the minimal. It is therefore enough to remove any token from *used*, and increase the token count in *n*. If the removed tuple is $\mathbf{0}$ then this is a perfect step. If not, it is a lossy step.

5.3. Simulation of (R21)

The procedure, partially depicted in Fig. 12, is as follows:

- (1) Select the minimal tuple, as previously explained.
- (2) Remove one token of the minimal tuple from c_0 and set $n' = n - 1$ (see transition $st(R21)$ in Fig. 12).
- (3) Create $m \leq n'$ copies of the selected tuple. This is done by firing $m' \leq n'$ times the transition new , thus creating m' fresh names in place p_{new} , followed by the firing $m \leq m'$ times of each transition cp_i , that does the actual copy.
- (4) Reset n' and p_{new} (transition $end(R21)$).

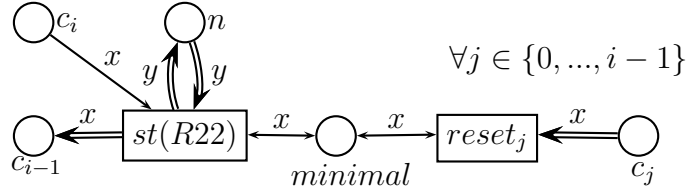


Figure 13: Simulation of (R22)

In this case, a perfect step is simulated whenever the minimal tuple is chosen in a non-lossy way, $m = n'$, every name creation is non-lossy and every cp_i is fired exactly n' times.

5.4. Simulation of (R22)

The procedure, partially depicted in Fig. 13, is as follows:

- (1) Select the minimal tuple v , as previously explained.
- (2) Select non-deterministically $i \in \{1, \dots, k-1\}$ with $v[i] > 0$, or deadlock if there is no such i (this choice is not shown in the figure).
- (3) Set $v[j] = 0$ for $0 \leq j < i$ (transitions reset_j in Fig. 13).
- (4) Set $v[i-1] = n$ and decrease $v[i]$ in one unit (transition $st(R22)$ in Fig. 13).

This simulation is perfect when the minimal tuple is chosen in a non-lossy way and the chosen i is exactly $\min\{j \mid c_j > 0\} > 0$. Now we proceed with the definition of \mathcal{N}^{-1} , that is, with the simulation of backwards computations. In the three cases we need to guarantee (possibly in a lossy way) that the considered marking is in the range of the rewriting system R .

5.5. Simulation of $(R1^{-1})$

It is enough to decrement the token count in n and put a fresh name in *used* (as explained in Sect. 4). Since this name does not appear anywhere in the net, it represents the $\mathbf{0}$ tuple. If the name creation is lossy (the selected tuple already existed) then this is a lossy simulation. Otherwise, we are simulating $(R1^{-1})$ perfectly.

5.6. Simulation of $(R21^{-1})$

Now we do not start by selecting the minimal tuple, but we guarantee (similarly as in the minimal tuple selection) that the distinguished tuple we are working with is minimal in the left handside of the rule. Intuitively, we start by selecting an arbitrary tuple v , which plays the role of the tuple $(c_0 - 1, c_1, \dots, c_{k-1})$ (one of them) in the right handside of the rule. We need to (i) guarantee that there are other $n - 1$ tuples equal to v (if not, we need to decrement n), (ii) those n tuples are minimal and (iii) there are no other tuples equal to them. We can do this as follows:

- (1) Select an arbitrary tuple v by putting it in *minimal* and transfer every other name in *used* to *tmp*.
- (2) Set $n' = n - 1$ and reset n . This is done by transferring the tokens from p_n to $p_{n'}$ and removing a token from $p_{n'}$ (deadlock if it is not possible).
- (3) Select an arbitrary tuple x and *force* it to be equal to v , as in the step (3eq) of the minimal selection. Decrement n' and increment n in one unit (moving a token from $p_{n'}$ to p_n). Remove x (remove its name from *tmp*) and repeat this step or go to (4).
- (4) Set $n' = 0$ (i.e., reset $p_{n'}$) and increment $v[0]$.
- (5) Select an arbitrary tuple x (by selecting an arbitrary name from *tmp*) and force it to be greater or equal than v (as in steps 2, 3eq and 3neq in the minimal selection). Move that name from *tmp* to *used*. Repeat this step or go to (6).
- (6) Reset *tmp* and move the name in *minimal* to *used*.

This is therefore a lossy simulation of $(R21^{-1})$. Notice that n is decremented whenever there are less than $n - 1$ tuples equal to v . The simulation becomes perfect when (i) the selected tuple v is actually minimal, (ii) step (3) is repeated $n - 1$ times, and always with a tuple x that is equal to v , and (iii) step (5) processes every other tuple (and in the correct way with respect to the minimal selection), so that the reset of *tmp* is only done when it is empty.

5.7. Simulation of $(R22^{-1})$

Similarly as in the previous case, the procedure is not preceded by a minimal tuple selection step, but this selection step is intertwined with the rest of the procedure.

- Select a tuple v and $i \in \{0, \dots, k-1\}$ such that $v[i-1] > 0$.
- Set $v[j] = 0$ for $j < i$ and increase by one the value of $v[i]$.
- Force v to be the minimal tuple according to the lexicographic order, that is, proceed from step (1) in the selection of the minimal in Sect. 5.1, once v has been selected as minimal.

This concludes the construction of \mathcal{N} and \mathcal{N}^{-1} . If we write $M, n \rightarrow_{\mathcal{N}} M', n'$ to denote that from the marking representing M, n we can reach a marking representing M', n' in \mathcal{N} (and analogously for \mathcal{N}^{-1}), we have the following result.

Proposition 10. *The following holds:*

- If $H^\alpha(n) = n'$ then $\mathbf{C}(\alpha), n \rightarrow_{\mathcal{N}}^* \emptyset, n'$ and $\emptyset, n' \rightarrow_{\mathcal{N}^{-1}}^* \mathbf{C}(\alpha), n$.
- If $M, n \rightarrow_{\mathcal{N}}^* M', n'$ then $n' \leq H^{\alpha(M)}(n)$.
- If $M, n \rightarrow_{\mathcal{N}^{-1}}^* M', n'$ then $n \geq H^{\alpha(M')}(n')$.

Once we have proved that UDN can weakly compute $F_{\omega^\omega} = H^{\omega^{\omega^\omega}}$ and its inverse, we can make use of the device presented for instance in [21] in order to obtain hyper-Ackermannian lower bounds, by reducing the acceptance problem for a $\mathbf{F}_{\omega^\omega}$ -bounded Minsky machine M to coverability/termination for UDN . We refer the reader to [21] for details, but we sketch the key ideas below.

We first put the Minsky machine M on a budget, meaning that we consider an extra counter b that contains the remaining budget. When we increase a counter we decrease the budget, and viceversa, so that the sum of all counters remains constant.

Then, we simulate the Minsky machine using UDN . Since this simulation can only be done in a weak sense (since UDN are not Turing-complete), we use the budget, together with a coverability condition, to witness that the simulation is correct. More precisely, given a Minsky machine M of size

$k \in \mathbb{N}$ we build the *UDN* \mathcal{N} that (weakly) computes $F_{\omega\omega}(k)$, the budget of M . Then we simulate M by a *UDN* \mathcal{N}_M (in a standard way), replacing tests for zero by a reset. When a test for zero is incorrectly executed, the counter actually decreases, which is witnessed thanks to the budget, since the sum of the counters has also decreased. Since the machine must end in a configuration in which every counter (but the budget) is null, it holds that the value of the budget is $b \leq F_{\omega\omega}(k)$, and only equals $F_{\omega\omega}(k)$ when (i) $F_{\omega\omega}(k)$ is computed correctly, and (ii) the simulation is correct. Finally, we use the *UDN* \mathcal{N}^{-1} , that only in that case can (weakly) compute $k = F_{\omega\omega}^{-1}(b)$ and check with a coverability condition that we have indeed computed it. Thus, M accepts iff a certain marking in the *UDN* $\mathcal{N}^{-1} \circ \mathcal{N}_M \circ \mathcal{N}$ can be covered. The case for termination is analogous, so that we can finally conclude the following, using Theorem 2 for the upper bound.

Theorem 3. *Coverability and termination for UDN are $\mathbf{F}_{\omega\omega}$ -complete.*

6. Conclusions and open problems

We have proved that the coverability and termination problems are complete for the $\mathbf{F}_{\omega\omega}$ class in the fast-growing complexity hierarchy. Up to our knowledge, this is the first problem that is complete at this level of the hierarchy with a state space that does not rely on words over a finite alphabet. This narrows the gap between *UDN* without broadcasts, called ν -PN in [18], which are complete for the $\mathbf{F}_{\omega.2}$ class of double Hyper-Ackermannian problems, and Data Nets, which are complete for $\mathbf{F}_{\omega\omega\omega}$.

Up to our knowledge, the exact complexity of Unordered Petri Data Nets, which are unordered Data Nets without broadcasts, whole-place operations or fresh name creation, is still open. They were proven to have a non-elementary complexity in [15], but no upper bound is currently known, other than the generic $\mathbf{F}_{\omega.2}$. There have been important recent advances on the study of this class of Petri Nets [13], yielding in particular the decidability (in hyper Ackermannian time) of place boundedness (the problem of deciding whether a given place is bounded). This is interesting because the decidabilities of place boundedness and reachability are usually coupled. Actually, the decidability of reachability is still open in this model.

Acknowledgments.

I would like to thank Ranko Lazic and Sylvain Schmitz for helpful discussions and the revision of an early version of this paper. I would also like to thank the anonymous referees for their insightful comments and suggestions for improvement.

References

- [1] P. A. Abdulla, and B. Jonsson. Verifying Programs with Unreliable Channels. *Inf. Comput.* 127(2): 91-101 (1996)
- [2] P. A. Abdulla, K. Cerans, B. Jonsson, Y. Tsay. Algorithmic Analysis of Programs with Well Quasi-ordered Domains. *Inf. Comput.* 160(1-2): 109-127 (2000)
- [3] P.A. Abdulla, G. Delzanno, and L. Van Begin. A classification of the expressive power of Well-Structured Transition Systems. *Inf. Comput.* 209(3): 248-279 (2011)
- [4] R. Bonnet. Well-Structured Petri Nets Extensions with Data. Master of Computer Science, LSV ENS Cachan, 2010.
- [5] G. Cécé, A. Finkel, S. P. Iyer. Unreliable Channels are Easier to Verify Than Perfect Channels. *Information and Computation* 124(1): 20-31 (1996)
- [6] P. Chambart, and Ph. Schnoebelen. The Ordinal Recursive Complexity of Lossy Channel Systems. 23rd Annual IEEE Symposium on Logic in Computer science, LICS 2008, pp. 205-216. IEEE Comp. Soc. Press, 2008.
- [7] M. Emmi, P. Ganty, R. Majumdar, F. Rosa-Velardo. Analysis of Asynchronous Programs with Event-Based Synchronization. 24th European Symposium on Programming, ESOP 2015. LNCS vol. 9032, pp. 535-559. Springer, 2015.
- [8] D. Figueira, S. Figueira, S. Schmitz, and Ph. Schnoebelen. Ackermann and Primitive-Recursive Bounds with Dickson's Lemma. 26th Annual IEEE Symposium on Logic in Computer science, LICS 2011, pp. 269-278. IEEE Comp. Soc. Press, 2011.

- [9] A. Finkel, and Ph. Schnoebelen. Well-Structured Transition Systems Everywhere! *Theoretical Computer Science* 256(1-2):63-92 (2001)
- [10] A. Finkel, P. McKenzie, and C. Picaronny. A well-Structured Framework for Analysing Petri Net Extensions. *Information and Computation* 195(1-2):1-29 (2004).
- [11] S. Haddad, S. Schmitz, P. Schnoebelen. The Ordinal-Recursive Complexity of Timed-arc Petri Nets, Data Nets, and Other Enriched Nets. 27th Annual IEEE Symposium on Logic in Computer science, LICS 2012, pp. 355-364. IEEE Comp. Soc. Press, 2012.
- [12] G. Higman. Ordering by Divisibility in Abstract Algebras. *Proc. London Math. Soc.* s3-2(1):326-336 (1952)
- [13] P. Hofman, S. Lasota, R. Lazic, J. Leroux, S. Schmitz, and P. Totzke. Coverability Trees for Petri Nets with Unordered Data. 19th Int. Conference on Foundations of Software Science and Computation Structures, FOSSACS 2016. LNCS vol. 9634, pp. 445-461. Springer, 2016.
- [14] D. H. J. de Jongh, and R. Parikh. Well Partial Orderings and Hierarchies. *Indagationes Mathematicae* 80:195-207 (1977)
- [15] R. Lazic, T.C. Newcomb, J. Ouaknine, A.W. Roscoe, and J. Worrell. Nets with Tokens Which Carry Data. *Fundamenta Informaticae* 88(3):251-274. IOS Press (2008)
- [16] R. Lazic, and S. Schmitz. The Complexity of Coverability in ν -Petri Nets. 31st Annual IEEE Symposium on Logic in Computer science, LICS 2016, pp. 467-476. IEEE Comp. Soc. Press, 2016.
- [17] F. Rosa-Velardo, D. de Frutos-Escrig, O. Marroquín-Alonso. On the Expressiveness of Mobile Synchronizing Petri Nets. International Workshop on Security and Concurrency, SecCo 2005. ENTCS vol. 180(1), pp. 77-94. Elsevier, 2007.
- [18] F. Rosa-Velardo, and D. de Frutos-Escrig. Decidability and Complexity of Petri Nets with Unordered Data. *Theoretical Computer Science* 412(34): 4439-4451 (2011)

- [19] F. Rosa-Velardo, and M. Martos-Salgado. Multiset rewriting for the verification of depth-bounded processes with name binding. *Information and Computation* 215: 68-87 (2012)
- [20] D. Schmidt. Well-Partial Orderings and their Maximal Order Types. Fakultät für Mathematik der Ruprecht-Karl-Universität Heidelberg. Habilitationsschrift, 1979.
- [21] S. Schmitz, and Ph. Schnoebelen. Multiply-Recursive Upper Bounds with Higman's Lemma. 38th International Colloquium on Automata, Languages and Programming - Part II, ICALP(2) 2011. LNCS vol. 6756, pp. 441-452. Springer, 2011.
- [22] S. Schmitz. Complexity Hierarchies Beyond Elementary. *ACM Transactions on Computation Theory* 8(1):3 (2016)
- [23] Ph. Schnoebelen. Revisiting Ackermann-Hardness for Lossy Counter Machines and Reset Petri Nets. 35th International Symposium on Mathematical Foundations of Computer Science, MFCS 2010. LNCS vol. 6281, pp. 616-628. Springer, 2010.
- [24] A. Weiermann. A Computation of the Maximal Order Type of the Term Ordering on Finite Multisets. *Mathematical Theory and Computational Practice, 5th Conference on Computability in Europe, CiE 2009*. LNCS vol. 5635, pp. 488-498. Springer, 2009.