Ordinal Recursive Complexity of Unordered Data Nets

Fernando Rosa-Velardo[⋆]

Technical Report 04/14
Sistemas Informáticos y Computación,
Universidad Complutense de Madrid
fernandorosa@sip.ucm.es

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 problemas have recently been proven complete for the class $\mathbf{F}_{\omega^{\omega^\omega}}$ 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. Using the result by Schmitz and Schnoebelen to bound the length of bad sequences in well-quasi orderings based on finite multisets over tuples of naturals, 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.

1 Introduction

Higman's lemma [14] 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 *ordinal types* [15, 19], 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} [15], while the order type of $X^\oplus$ is only ω^α [24].

The ordinal type of a wqo has recently been used in several works [6, 20, 8, 13] to characterize the ordinal-recursive complexity of (the verification of several problems for) monotonic systems over an underlying wqo, which have been called

⋆ Author partially supported by the MEC Spanish project Strongsoft, and the CAM program PROMETIDOS S2009/TIC-1465.

Well-Structured Transition Systems (WSTS) [2, 11]. Prominent examples of such WSTS are Petri nets/VASS, affine nets [10], Lossy Channel Systems [1, 5] or Data Nets [17]. 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 Q and $k \geq 0$.

Data Nets [17] 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 [4] as arrays or lists of Petri nets (with whole-place operations) communicating by rendezvous and broadcasts. 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.

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. In this case, the state space can be given by some $(\mathbb{N}^k)^\oplus$. On the one hand, unordered Petri Data Nets were proved to have a non-elementary complexity in [17]. On the other hand, (full) Data Nets have been recently proven to be complete for the $\mathbf{F}_{\omega^{\omega^\omega}}$ class in the fast-growing complexity hierarchy. We fill part of the gap in between by proving that (non Petri) *UDN* are complete for $\mathbf{F}_{\omega^\omega}$. This answers positively a question posed in [13].

Thus, the complexity of *UDN* seats at the exact same level of LCS, which were proven to be $\mathbf{F}_{\omega^\omega}$-complete in [6, 20]. Our proof relies on the techniques developed by Schnoebelen and Schmitz, both for the upper bounds as for the hardness result. We first use the techniques in [20] to bound the length of controlled bad sequences in $(\mathbb{N}^k)^\oplus$. Then 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].

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 on the relation between *UDN* and ν-PN. The latter can be seen as a restriction of UDN without broadcasts. Indeed, the construction reducing Data nets to Petri Data Nets (removing whole-place operations and broadcasts) 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. Thus, the exact complexity of ν-PN is still open.

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 define *UDN* and obtain an upper bound for their coverability and termination problems. In Sect. 4 we consider lower bounds. Sect. 5 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. 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.

Multisets. Given a set X, we denote by $X^\oplus$ the set of finite multisets of X, that is, the set of mappings $m : X \to \mathbb{N}$ with a finite support $sup(m) = \{x \in X \mid m(x) \neq 0\}$. 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 $\{x_1, \ldots, x_n\} \leq_\oplus \{x'_1, \ldots, x'_m\}$ if there is an injection $h : \{1, \ldots, n\} \to \{1, \ldots, m\}$ such that $x_i \leq_X x'_{h(i)}$ for each $i \in \{1, \ldots, n\}$.

Words. Given a set X, any $u = x_1 \cdots x_n$ with $n \geq 0$ and $x_i \in X$, for all $i \in \{1, ..., n\}$, 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: $x_1 \ldots x_n \leq_{X^*} x'_1 \ldots x'_m$ if there is a strictly increasing mapping $h : \{1, \ldots, n\} \to \{1, \ldots, m\}$ such that $x_i \leq_X x'_{h(i)}$ for each $i \in \{1, \ldots, n\}$ (Higman's lemma). This is called the embedding order, in contrast with the lexicographic order: $x_1 \ldots x_n <_{lex} x'_1 \ldots x'_m$ iff there is $i \leq min(n, m)$ such that $x_j = x'_j$ for $j < i$ and $x_i < x'_i$, or $x_1 ... x_n = x'_1 ... x'_n$ and $m > n$. Then the reflexive closure $\leq_{lex}$ of $<_{lex}$ is a total (or linear) order, though not necessarily a well order (for instance, $b, ab, aab, ...$ is strictly decreasing). The lexicographic order is a linearization of the embedding order, i.e., $w \leq w'$ implies $w \leq_{lex} w'$.

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

Given a WSTS $\mathcal{S} = (X, \to, \leq)$, we say $\mathcal{S}_l = (X, \to_l, \leq)$ is a *lossy* version of $\mathcal{S}$ if $\to_l$ is obtained by adding to $\to$ some transitions $x \to_l x'$ 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. A *normed wpo* is a wpo $(X, \leq)$ endowed with a norm $|\cdot|_X : X \to \mathbb{N}$, 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 $x_0, ..., x_L$ 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) $\alpha = \omega^{\alpha_1} + ... + \omega^{\alpha_n}$, with $\alpha_1 \geq ... \geq \alpha_n$ in CNF, and where the order between two ordinals is defined as follows:

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

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 $\omega^\alpha + ... + \omega^\alpha$ (c times), and use the so called strict form $\alpha = \omega^{\alpha_1} \cdot c_1 + ... + \omega^{\alpha_m} \cdot c_m$, with $\alpha_1 > ... > \alpha_m$ and $c_i > 0$.

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

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

It holds $\lambda_n < \lambda$ for every n, and $\lambda = sup_n \lambda_n$. Fundamental sequences are used to define subrecursive hierarchies: The *Hardy hierarchy* $(H^\alpha : \mathbb{N} \to \mathbb{N})_\alpha$ is defined by

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

The *fast growing hierarchy* $(F_\alpha : \mathbb{N} \to \mathbb{N})_\alpha$ can be defined as $F_\alpha = H^{\omega^\alpha}$.

Given α, the class $\mathscr{F}_\alpha$ 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 $(\mathscr{F}_\alpha)_\alpha$ is called the *extended Grzegorczyck hierarchy.* Instead of directly working with the extended Grzegorcyzck hierarcy, we work with the fast growing complexity hierarchy $(\mathbf{F}_\alpha)_\alpha$, defined for instance in [21], and better suited to establish completeness results.

$$\mathbf{F}_\alpha = \bigcup_{\substack{\beta < \alpha \\ p \in \mathscr{F}_\beta}} \mathit{TIME}(F_\alpha(p(n))$$

Here, $\mathit{TIME}(f(n))$ stands for the class of problems decidable in time $f(n)$.[1] For instance, $\mathbf{F}_3$ is the class of *tower*-like problems with a complexity bounded by the non-elementary function $tower(x) = 2^{\left.\cdot^{\cdot^{2}}\right\}x\ times}$, closed under elementary reductions, $\mathbf{F}_\omega$ is the class of Ackermannian problems, closed under primitive-recursive reductions, and $\mathbf{F}_{\omega^\omega}$ is the class of hyper-Ackermannian problems,

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

closed under multiplive-recursive reductions. For every α, the halting problem for a Minsky machine M with sum of counters bounded by $F_\alpha(|M|)$ is a $\mathbf{F}_\alpha$-complete problem [21].

3 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. 4. Instead of defining *UDN* as a subclass of Data Nets, we do it from scratch (as hinted by in [17]), thus obtaining a definition that is simpler than the one in [17], though equivalent. In the next sections we denote the null tuple $(0, ..., 0) \in \mathbb{N}^k$ (for any k) as $\mathbf{0}$. We fix an infinite set *Var* of variables, and a special variable $\mathcal{R} \in \mathit{Var}$ (that will be used in broadcasts).

Definition 1 (Unordered Data Nets). *A k-dimensional* Unordered Data Net *(UDN_k) is a tuple $N = (P, T, F, G, H)$ where:*

- *$P = \{p_1, ..., p_k\}$ 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 $\mathit{Var}(t) \subseteq \mathit{Var}$ with $\mathcal{R} \in \mathit{Var}(t)$ such that:*
  - *$F_t : \mathit{Var}(t) \to \mathbb{N}^k$ is the subtraction function, such that $F_t(\mathcal{R}) = \mathbf{0}$,*
  - *$H_t : \mathit{Var}(t) \to \mathbb{N}^k$ is the addition function,*
  - *$G_t : \mathit{Var}(t) \times \mathit{Var}(t) \to \mathbb{N}^{k \times k}$ 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 $(v_1, ..., v_k)$ represents a name carried by v_i tokens in p_i, for each $i \in \{1, ..., k\}$.

Definition 2 (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* $\mathbf{0}$-expansion *of a marking m' (or m' is a* $\mathbf{0}$-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 $\mathbf{0}$-expansions/contractions.

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

- for each $i \in \{1, ..., n\}$, $x_i \in \mathit{Var}(t)$ 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 $\mathbf{0}$-expansions. Some other names/tuples $v_{n+1}, ..., v_{n'}$ 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 $i \in \{1, ..., n\}$,
– whole-place operations are done via the matrices G_t:
  - 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 $G_t(\mathcal{R}, x_j)$ 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, 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 3. *Let m_1 be a marking of N and $t \in T$ with $Var(t) = \{x_1, ..., x_n\}$. A marking m_2 can be reached from m_1 by firing t as follows:*

1. *Let $m'_1 = \{v_1, ..., v_n, v_{n+1}, ..., v_{n'}\}$ 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 $\{v'_1, ..., v'_{n'}\}$.*

In figures, we will use standard conventions: places are represented as circles and transitions by squares. For each $x \in Var(t)$, with $F_t(x) = (n_1, ..., n_k)$, we draw an arrow from p_i to t labelled 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. 4 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 **Id** and **0** the identity and the null matrices, respectively. A transition t in an *UDN* can perform transfers or resets among a selected tuple $x \in Var(t)$, simply by setting $G_t(y, z) = \mathbf{0}$ for every $y \neq z$, $G_t(y, y) = \mathbf{Id}$ for every $y \neq x$, and $G_t(x, x)$ accordingly. In case a place p is reset, it is enough to set $G_t(x, x)$ as the identity matrix except for the p-row, which is null. Similarly, we can transfer all the x-tokens from p to q, or copy all the x-tokens in p to q.[2] In figures, we denote by double arrows these

[2] These operations are exactly the whole-place operations in affine nets [10].

whole-place operations, labelled with variables. More specifically, an x-labelled double arrow from p to t, together with an x-labelled double arrow from t to q, represents the transfer of all the x-tokens from p to q (see transtion $st(eq)$ in Fig. 2). A double arrow labelled by x from p to t, such that there is no x-labelled outgoing arrow, represents the reset of the x-tokens in p (see for instance transition $end(neq)$ in Fig. 5). Finally, if there is also an x-labelled arrow from t to p, they represent the copy of the x-tokens in p to q (e.g., transition $st(neq)$ in Fig. 4). So far, we are assuming $G_t(x, y)$ is the null matrix if $x \neq y$. We can also copy the x-tokens in p as y-tokens in q, by setting $G_t(x, y)(p, q) = 1$. In figures, this operation will be represented by having an x-labelled double arc from p to t, an x-labelled double arc from t to p, and a y-labelled double arc from t to q (see transition $st(R22)$ in Fig. 7).

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. We take $Var(t) = \{x, \mathcal{R}\}$, so that the tuple to which x is instantiated instructs the rest to do a reset in place p. We take $F_t(\mathcal{R}) = \mathbf{0}$, $G_t(x, x) = \mathbf{Id}$, $G_t(x, \mathcal{R}) = G_t(\mathcal{R}, x) = \mathbf{0}$ and $G_t(\mathcal{R}, \mathcal{R})$ be the identity matrix, except that its p-row is null (we leave $F_t(x)$ and $H_t(x)$ unspecified). Similarly, we can have full reset broadcasts (in which not only one place, but every place is reset) or transfer broadcasts, in which every tuple transfers the tokens from one place to another.

In figures, a double arrow from p to t, together with a double arrow from t to q (without any label), represents the (broadcast) transfer of all the tokens from p to q (see transition *start* in Fig. 1). If there are no such outgoing arrows, it represents the reset of place p (like in transition *end* in Fig. 1).

(Lossy) name creation. As discussed in [17], *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 $y \in Var(t)$. 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 $\nu \in Var(t)$, only appearing in post-arcs, to represent lossy name creations (see transition *new* in Fig. 6).Notice that lossy name creation can already be achieved in Unordered Data Nets without broadcasts (we do not make use of $\mathcal{R}$).

The state space of a d-dimensional *UDN* is $A = (\mathbb{N}^d)^{\oplus}$, with $o(A) = \omega^{\omega^d}$. Since sequences are controlled by a function g which is primitive recursive (it is actually linear for termination and exponential for coverability [17]), we can apply [22, Theorem 3.3] together with [21, Corollary 4.3] to conclude that $L_{g,A}$

is bounded by some function in $\mathbf{F}_{\omega^d}$. This gives us combinatorial algorithms that work in nondeterministic space bounded by $\mathbf{F}_{\omega^d}$ (see e.g. [20] for details).

Theorem 1. *Termination and coverability for UDN_d are in $\mathbf{F}_{\omega^d}$. Those problems for UDN are in $\mathbf{F}_{\omega^\omega}$.*

4 Hyper-Ackermannian Lower bound

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 $v = (v[0], ..., v[k-1])$. 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 $(v[i], ..., v[j])$, and $\min(v, v') = (\min(v[0], v'[0]), ..., \min(v[k-1], v'[k-1]))$. Also, we identify each ordinal α with $\{\beta \mid \beta < \alpha\}$.

Definition 4. *We define $\mathbf{v} : \omega^k \to \mathbb{N}^k$ as $\mathbf{v}(\omega^{k-1} \cdot c_{k-1} + ... + \omega^0 \cdot c_0) = (c_0, ..., c_{k-1})$, and $\mathbf{C} : \omega^{\omega^k} \to (\mathbb{N}^k)^\oplus$ as $\mathbf{C}(\omega^{\beta_1} + ... + \omega^{\beta_l}) = \{\mathbf{v}(\beta_1), ..., \mathbf{v}(\beta_l)\}$.*

We also define the converse mappings.

Definition 5. *We define $\beta : \mathbb{N}^k \to \omega^k$ as $\beta(v) = \omega^{k-1} \cdot v[k-1] + ... + \omega^0 \cdot v[0]$ and $\alpha : (\mathbb{N}^k)^\oplus \to \omega^{\omega^k}$ as $\alpha(\{v_1, ..., v_l\}) = \omega^{\beta_1} + ... + \omega^{\beta_l}$, where $\beta_1 \geq ... \geq \beta_l$ is a reordering of $\beta(v_1), ..., \beta(v_l)$.*

Proposition 1. *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 loosing a summand in the CNF of α, and decreasing some value in v means loosing a summand in the (non-strict) CNF of $\beta(v)$. □

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 1. *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:

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

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

Proposition 2. *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 $\mathbf{v}(\beta) = \mathbf{0} \in \mathbf{C}(\alpha)$. □

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

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

Lemma 3. *Let $m = m' + \{v\}$ with $v = (c_0, ..., c_{k-1}) = \min_{lex} m$, be the encoding of a limit ordinal $\alpha = \gamma + \omega^{\beta+1} < \omega^{\omega^k}$ and let $n > 0$. Then $\mathbf{C}(\alpha_n) = m' + \{(c_0 - 1, c_1, ..., c_{k-1})^n\}$.*

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

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

Definition 6. *We define R as the following rewriting system over $(\mathbb{N}^k)^\oplus$:*

(R1) $\{\mathbf{0}\}, n \to \emptyset, n + 1$

(R21) $\{(c_0, c_1, ..., c_{k-1})\}, n \to \{(c_0 - 1, c_1, ..., c_{k-1})^n\}, n$ if $c_0 > 0$

(R22) $\{(0, ..., 0, c_i, ..., c_{k-1})\}, n \to \{(0, ..., n, c_i - 1, ..., c_{k-1})\}, n$ if $i = \min\{j \mid c_j > 0\} > 0$

(Min) $\{v\}, n \to M', n' \Rightarrow M, n \to (M - \{v\}) + M', n'$ 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 1).

Proposition 3. *If $M, n \to_R M', n'$ then $H^{\alpha(M)}(n) = H^{\alpha(M')}(n')$.*

In particular, since $\alpha(\emptyset) = 0$, if $\mathbf{C}(\alpha), n \to_R^* \emptyset, n'$ then $H^\alpha(n) = n'$. We denote by (R^{-1}) the rewriting system obtained by inversing the rules of (R).

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

$$\frac{M_1, n_1 \geq M_1', n_1' \to_X M_2', n_2' \geq M_2, n'}{M_1, n_1 \to_{X_l} M_2, n_2}$$

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 7 (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.[3]

Lemma 4. *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 $H^{\omega^i \cdot (c+1)}(n) \geq H^{\omega^i \cdot (c+1)}(n)$, 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 $d_1, ..., d_{k-1} \in \mathbb{N}$, and by iterating the previous argument $d_1 + ... + d_{k-1}$ times we conclude that $H^{\beta(v_1)}(n) \leq H^{\beta(v_2)}(n)$. Similarly, for the general case it is enough to see that $H^{\alpha(m+\{0\})}(n) = H^{\alpha(m)+1}(n) = H^{\alpha(m)}(n+1) \geq H^{\alpha(m)}(n)$.

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 4. *The following two conditions hold:*

- *If $M, n \to_{R_l} M', n'$ then $H^{\alpha(M)}(n) \geq H^{\alpha(M')}(n')$*
- *If $M, n \to_{R_l^{-1}} M', n'$ 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. 3 and Lemma 4.

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

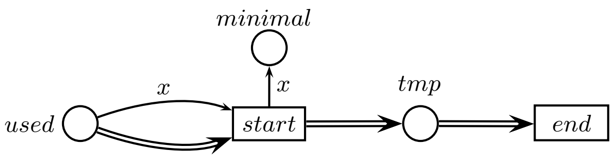

Fig. 1. Start and end of the selection of the minimal

Unordered Data Nets for weak Hardy computations

Let us define $\mathcal{N}$ and $\mathcal{N}^{-1}$, *UDN*s 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 $c_0, ..., c_{k-1}$ to represent such tuples. We also have a place p_n ir 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 $a \in used$,[4] we write $M(a)$ to denote the tuple $(v_0, ..., v_{k-1})$ provided there are v_i a-tokens in c_i, for $0 \leq i < k$. Hence, any marking M represents the multiset of markings $\{M(a_1), ..., M(a_n)\}$, provided *used* contains exactly the set $\{a_1, ..., a_n\}$.

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 $\{(0, ..., 0, k)\}, k$, and $\{(0, ..., 0, k)\}$ is in turn the encoding of $\omega^{\omega^{k-1} \cdot k}$. Notice that $H^{\omega^{\omega^{\omega}}}(k) = H^{\omega^{\omega^{k}}}(k) = H^{\omega^{\omega^{k-1} \cdot k}}(k)$.

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 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.

Minimal selection. In the following, for $j \in \{0, ..., k-1\}$ 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 $j \in \{0, ..., k-1\}$. The intuitive key idea to select the minimal tuple with respect 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.

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

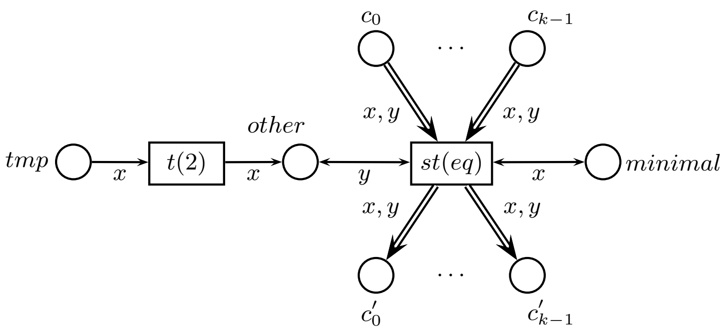

Fig. 2. Minimal selection: (2) and start of (3eq)

Example 1. 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.

Formally, 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. 1). 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. 2). 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. 2, that transfers all the tokens of *minimal* and *other* from places $c_0, ..., c_{k-1}$ to places $c'_0, ..., c'_{k-1}$. Then fire the transitions t_i (for all $i \in \{0, ..., k-1\}$ in

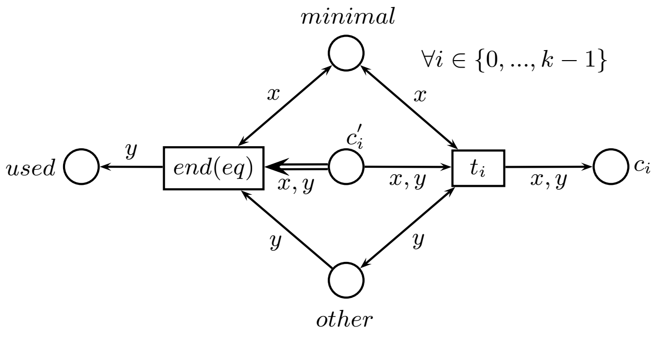

Fig. 3. Minimal selection: end of (3eq)

Fig. 3 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. 3, 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 $j \in \{0, ..., k-1\}$. Then set

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

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

This can be done as follows: first select any $j \in \{0, ..., k-1\}$ (choice not shown in the figures). Then fire $st(neq)$ in Fig. 4. 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. 5 any number of times, which has the analogous role of transitions t_i in Fig. 3. 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 2 and Fig. 3), but only for each $i \in \{j+1, ..., k-1\}$ (instead of $i \in \{0, ..., k-1\}$). Finally, we fire transition $end(neq)$ in Fig. 5, which empties c_j'. Then either go to 2 or go to 4.

(4) Reset *tmp* (transition *end* in Fig. 1).

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 5. $v_l = \min_{lex}\{x_1', ..., x_l', v_l\}$.

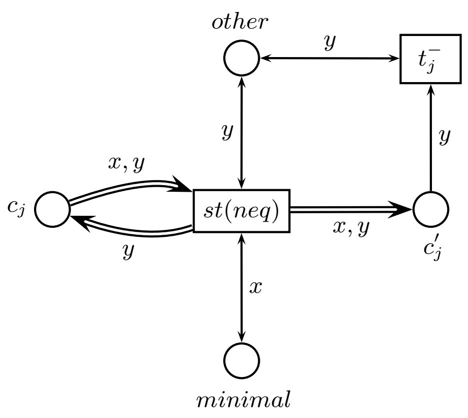

Fig. 4. Minimal selection: start of (3neq)

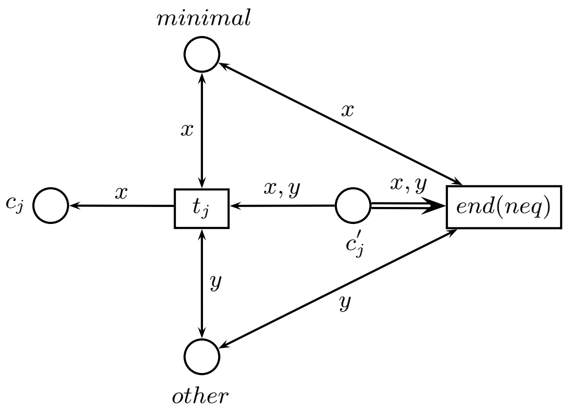

Fig. 5. Minimal selection: end of (3neq)

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 $\leq$ 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 $m = \{v, x_1, ..., x_l\} + \overline{m}$ we may end up in any marking $m' = \{v_l, x_1', ..., x_l'\}$, and $m' \leq m$. However, it can be a perfect step whenever $\overline{m} = \emptyset$, $v_l = v$ 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. 1, the net could choose $(0, 0, 0)$ as the minimal tuple, and compare it with every other tuple in the right way: e.g., $(0, 0, 0) <_1 (0, 1, 2)$.

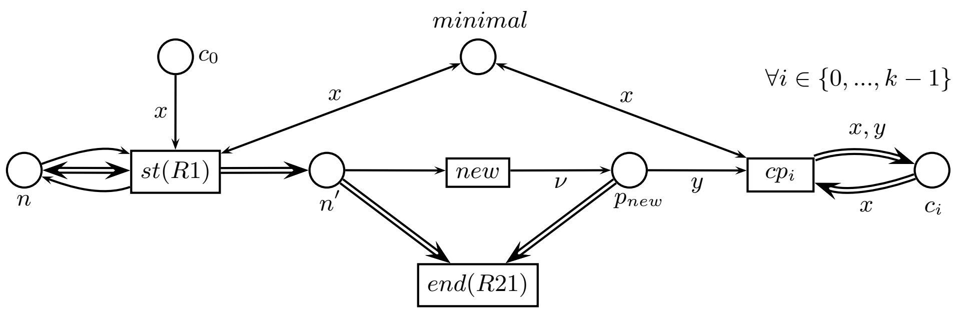

Fig. 6. Simulation of (R21)

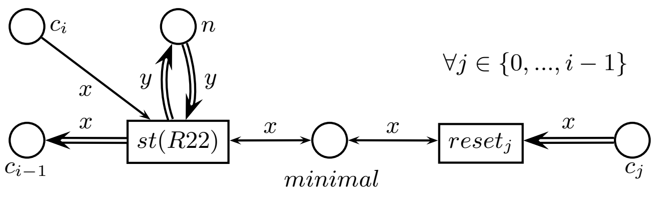

Fig. 7. Simulation of (R22)

Next we show how the rules in both rewriting systems can be simulated using *UDN* transitions.

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.

Simulation of (R21). The procedure, depicted in Fig. 6, 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. 6).
(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 $end(R21)$).

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.

Simulation of (R22). The procedure, depicted in Fig. 7, is as follows:

(1) Select the minimal tuple v, as previously explained.
(2) Select non-deterministically $i \in \{1, ..., k-1\}$ 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. 7).
(4) Set $v[i-1] = n$ and decrease $v[i]$ in one unit (transition $st(R22)$ in Fig. 7).

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.

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. 3). Since this name does not appear anywhere in the net, it represents the **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.

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 $(c_0 - 1, c_1, ..., c_{k-1})$ (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 transfering 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.

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 $i \in \{0, ..., k-1\}$ 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 lexicographic order, that is, proceed from step (1) in the selection of the minimal, 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 5. *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 [20] 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 [20] 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.

Theorem 2. *Coverability and termination for UDN are* $\mathbf{F}_{\omega^\omega}$*-complete.*

5 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.

The complexity of other restrictions of Data nets are still open: *UDN* without broadcasts, called ν-PN in [18] or Unordered Petri Data Nets, which are unordered Data Nets without broadcasts, whole-place operations or fresh name creation. In the case of ν-PN, they are known to be $\mathbf{F}_\omega$-hard and in $\mathbf{F}_{\omega^\omega}$. Unordered Petri Data nets were proven to have a non-elementary complexity in [17], but no currently known upper bound is also the generic $\mathbf{F}_{\omega^\omega}$. Actually, the decidability of reachability is also an open problem in this model.

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 Language-Based Comparison of Extensions of Petri Nets with and without Whole-Place Operations. LATA'09, LNCS vol. 5457, pp. 71-82. Springer, 2009.
4. R. Bonnet. Well-structured Petri Nets extensions with data. Master Computer Science, EPFL, Lausanne, Switzerland, March 2010.
5. G. Cécé, A. Finkel, S. P. Iyer. Unreliable Channels are Easier to Verify Than Perfect Channels. Inf. Comput. 124(1): 20-31 (1996)
6. P. Chambart, and Ph. Schnoebelen. The Ordinal Recursive Complexity of Lossy Channel Systems. LICS 2008: 205-216.
7. A. Finkel, R. Bonnet, S. Haddad, F. Rosa-Velardo. Comparing Petri Data Nets and Timed Petri Nets. LSV Research Report 10-23 2010.
8. D. Figueira, S. Figueira, S. Schmitz, and Ph. Schnoebelen. Ackermann and Primitive-Recursive Bounds with Dickson's Lemma. CoRR abs/1007.2989: (2010)
9. A. Finkel. A generalization of the procedure of karp and miller to well structured transition systems. 14th Int. Colloquium on Automata, Languages and Programming, ICALP'87, LNCS vol. 267, pp. 499-508. Springer, 1987.
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. A. Finkel, and Ph. Schnoebelen. Well-structured transition systems everywhere! Theor. Comput. Sci. 256(1-2): 63-92 (2001)

12. G. Geeraerts, J. Raskin, and L. Van Begin. Well-structured languages. Acta Informatica, 44:249-288. Springer, 2007.
13. S. Haddad, S. Schmitz, P. Schnoebelen. The Ordinal-Recursive Complexity of Timed-arc Petri Nets, Data Nets, and Other Enriched Nets. LICS 2012: 355-364
14. G. Higman. Ordering by Divisibility in Abstract Algebras. Proc. London Math. Soc. (1952) s3-2 (1): 326-336
15. D. H. J. de Jongh, and R. Parikh. Well partial orderings and hierarchies. Indagationes Mathematicae (Proceedings), vol. 80, p. 195-207, 1977
16. O. Kouchnarenko, and Ph. Schnoebelen. A Formal Framework for the Analysis of Recursive-Parallel Programs. In PaCT'97, LNCS 1277, pages 45-59. Springer, 1997
17. R. Lazic, T.C. Newcomb, J. Ouaknine, A.W. Roscoe, and J. Worrell. Nets with Tokens Which Carry Data. Fund. Informaticae 88(3):251-274. IOS Press, 2008.
18. F. Rosa-Velardo, and D. de Frutos-Escrig. Decidability and complexity of Petri nets with unordered data. Theor. Comput. Sci. 412(34): 4439-4451 (2011)
19. D. Schmidt. Well-partial orderings and their maximal order types. Fakultat fur Mathematik der Ruprecht-Karl-Universitat Heidelberg. Habilitationsscrift, 1979.
20. S. Schmitz, and Ph. Schnoebelen. Multiply-Recursive Upper Bounds with Higman's Lemma. ICALP (2) 2011: 441-452
21. S. Schmitz. Complexity Hierarchies Beyond Elementary. http://arxiv.org/abs/1312.5686, 2013.
22. S. Schmitz. Complexity Bounds for Ordinal-Based Termination. In RP'14, LNCS 8762, pages 1-19. Springer, 2014.
23. Ph. Schnoebelen. Revisiting Ackermann-Hardness for Lossy Counter Machines and Reset Petri Nets. MFCS 2010: 616-628
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.