processes

MDPI

Article

Parallel Multiset Rewriting Systems with Distorted Rules

Cristina Sburlan † and Dragoş-Florin Sburlan *,†

Faculty of Mathematics and Informatics, Ovidius University of Constanta, 900527 Constanta, Romania; c_sburlan@univ-ovidius.ro

* Correspondence: dsburlan@univ-ovidius.ro

† These authors contributed equally to this work.

Abstract: Most of the parallel rewriting systems which model (or which are inspired by) natural/artificial phenomena consider fixed, a priori defined sets of string/multiset rewriting rules whose definitions do not change during the computation. Here we modify this paradigm by defining level-t distorted rules—rules for which during their applications one does not know the exact multiplicities of at most $t \in \mathbb{N}$ species of objects in their output (although one knows that such objects will appear at least once in the output upon the execution of this type of rules). Subsequently, we define parallel multiset rewriting systems with t-distorted computations and we study their computational capabilities when level-1 distorted catalytic promoted rules are used. We construct robust systems able to cope with the level-1 distortions and prove the computational universality of the model.

Keywords: multiset rewriting systems; promoters; distorted computation; robust computation; computational universality

Citation: Sburlan, C.; Sburlan, D.-F. Parallel Multiset Rewriting Systems with Distorted Rules. *Processes* **2021**, *9*, 347. https://doi.org/10.3390/pr9020347

Academic Editors: Luis Valencia Cabrera, Mario de Jesús Pérez Jiménez and Agustin Riscos Núñez

Received: 31 December 2020
Accepted: 9 February 2021
Published: 14 February 2021

1. Introduction

Biological, social, and economic life are governed by different levels of uncertainty which usually regard the epistemic cases and facts related to incomplete knowledge due to unknown/faulty information. In general, any partially observable environment can induce uncertainty (as uncertainty reflects quantitative and qualitative aspects of limited information about the subject). Modeling uncertainty and embedding it into formal models is usually done due to lack of knowledge about the studied phenomenon and is often one of the sources of inaccuracy in computational model simulation, analysis, and validation. In this respect, a small local imprecision may determine profound effects during the computation of the model.

When defining a computational model, one might be interested about designing a "fault/deviation"-tolerant system which performs its intended operation even when some part of it is faulty, deviated (distorted from the initially assumed correct form), or one does not know the exact behavior (hence one has to include into the definition presumably true outcomes). Hence, the issue of building rewriting systems that are able to cope with uncertainty (at the level of parallelism of the rewriting, or regarding the execution time of the rewriting rules, or the non-determinism) and solve problems was addressed in several papers (see [1–6]). Yet another approach to model uncertainty was to dynamically construct rules at each step during the computation. In this respect we refer the reader to [7]. Designing a robust system usually means developing a self-stabilizing computation which is able to "recover" from a specified "fault" by being able to detect the "error" and drive towards an error-free state (possibly, starting over).

A *P system* is a computational model inspired by the parallel processes occurring in the living nature, especially by the bio-chemical reactions happening at the level of cells. Since its introduction in [8] many variants were proposed in order to capture various aspects of cellular processes, but their applicability in time proved to be also successful in covering phenomena raging from social, biological, economical, to ecological processes. Here we

considered the simplified version of catalytic P systems with promoters and/or inhibitors at the level of the rules (see [9]) which does not take into account the hierarchical membrane structure—hence we explored the computational capabilities of parallel multiset rewriting systems with catalysts and promoters/inhibitors whose rewriting rules are *distorted* during computations. Intuitively, a distorted multiset rewriting rule expresses the idea that one knows the participants (objects) in the rewriting process but does not know the exact outcome (meaning that one knows which are the resulting objects but does not know their multiplicities). Correspondingly, when defining the computation, one might consider several levels of distortions as the multiplicities of different species of objects in the output of an applied rewriting rule might be different than the initially defined ones. We are interested in defining multiset rewriting systems with 1-distorted computations which are able to generate the same set of numbers regardless the distortions of the rules applied during the computation.

In Section 2 we outline the basic notions and results regarding multisets, register machines, and parallel multiset rewriting systems. Section 3 introduces parallel multiset rewriting systems with promoters/inhibitors and t-distorted computations. Some results regarding their computational power are presented. Finally, in Conclusions we discuss several open problems and future lines of research.

2. Prerequisites

In what follows, we recall the basic notions and notations that will be used in the paper; we recommend [10] for a comprehensive summary of results from the formal language theory and other related fields. We also present parallel multiset rewriting systems as a particular case of P systems (see [11]).

2.1. Multisets, Distorted Multisets, and Related Concepts

An alphabet is a finite set of symbols $\Sigma = \{a_1, a_2, \ldots, a_n\}$, $\Sigma \neq \emptyset$. A string over Σ is a finite sequence of juxtaposed symbols from Σ, i.e., $w = b_1 \ldots b_k$, such that $b_i \in \Sigma$, for all $1 \leq i \leq k$. The empty string is denoted by λ. Σ^* represents the set of all strings over Σ. The Parikh vector associated with a string $w \in \Sigma^*$ is $\Psi_\Sigma(w) = (|w|_{a_1}, \ldots, |w|_{a_n})$. The Parikh image of a language $L \subseteq \Sigma^*$ is denoted by $\Psi_\Sigma(L) = \{\Psi_\Sigma(w) \mid w \in L\}$. Given a family of languages FL, by NFL ($PsFL$, respectively) we denote the family of length sets (Parikh images, respectively) of the languages from FL. We denote by $ET0L$ the family of languages generated by ET0L systems and by RE the family of recursively enumerable languages. It is known that $NET0L \subset NRE$ (see [12]).

A *multiset* over Σ is a mapping $M : \Sigma \to \mathbb{N}$. Given $a \in \Sigma$, the number $M(a)$ is called the multiplicity of a in M. The *support* of M is $supp(M) = \{a \in \Sigma \mid M(a) \neq 0\}$. If $supp(M) = \emptyset$ then M is the *empty multiset*. Here, we will represent a multiset M as the string $a_1^{M(a_1)} a_2^{M(a_2)} \ldots a_n^{M(a_n)}$ (any permutation of this string can also represent M). The empty multiset is represented by λ.

If $M_1, M_2 : \Sigma \to \mathbb{N}$ are two multisets then the union of M_1 and M_2, denoted by $M_1 + M_2$, is the multiset $M : \Sigma \to \mathbb{N}$ such that $M(a) = M_1(a) + M_2(a)$, for all $a \in \Sigma$. The relative complement of M_2 in M_1, denoted by $M_1 \setminus M_2$ is the multiset $M : \Sigma \to \mathbb{N}$ such that for any $a \in \Sigma$, $M(a) = M_1(a) - M_2(a)$ if $M_1(a) > M_2(a)$ or $M(a) = 0$, otherwise. We say that M_1 is included in M_2, namely $M_1 \subseteq M_2$, if $M_1(a) \leq M_2(a)$, for all $a \in \Sigma$. Let $w = a_1^{M(a_1)} a_2^{M(a_2)} \ldots a_n^{M(a_n)} \in \Sigma^*$ be a multiset and $k \in \mathbb{N}$. Then we define the scalar multiplication as $k * M = a_1^{k \cdot M(a_1)} \ldots a_n^{k \cdot M(a_n)}$.

Given $w = a_1^{M(a_1)} a_2^{M(a_2)} \ldots a_n^{M(a_n)} \in \Sigma^*$ a multiset and $a_j \in \Sigma$, $1 \leq j \leq n$, then we denote by $\mathtt{rm}_{a_j} = a_1^{M(a_1)} \ldots a_n^{M(a_n)} \setminus a_j^{M(a_j)}$ (remove the species indicated by a_j from w).

Definition 1. *Let* $w = a_1^{M(a_1)} a_2^{M(a_2)} \dots a_n^{M(a_n)}$ *be a multiset and* $1 \leq t \leq card(supp(M))$. *A* level-$t$ strong distortion *of* w *is a multiset*

$$\tilde{w} = a_1^{M(a_1)+j_1} \dots a_n^{M(a_n)+j_n}$$

such that for all $1 \leq i \leq n$ *the following conditions are satisfied:*

- $j_i \in \mathbb{Z}$, $M(a_i) + j_i \geq 0$, *if* $M(a_i) \neq 0$, (1)
- $j_i = 0$, *if* $M(a_i) = 0$,

and at most t *elements from* $\{j_1, \dots, j_n\}$ *are not 0.*

If in the relation (1) the inequality is strict, that is $M(a_i) + j_i > 0$ *if* $M(a_i) \neq 0$, *then the distortion is called* weak.

Remark 1. *A weak distortion is a particular case of a strong one. More precisely, if* w *is a multiset and* $\tilde{w}$ *a level-*t *weak distortion of* w, *then* $supp(w) = supp(\tilde{w})$ *and* $card(supp(w - \tilde{w}) \cup supp(\tilde{w} - w)) \leq t$. *On the other hand, in case of strong distortions, the assertion* $supp(w) = supp(\tilde{w})$ *is not necessarily true.*

Example 1. *Let* $\Sigma = \{a, b, c, d\}$ *be an alphabet and* $w = a^{10}b^{20}c^{30} \in \Sigma^*$ *be a multiset. Then* $supp(w) = \{a, b, c\}$ *and the followings are examples of distorted multisets.*

level-1 weak distortion:	$a^{10}b^{20}c^{30}$, $a^{1}b^{20}c^{30}$, $a^{10}b^{25}c^{30}$
level-2 weak distortion:	$a^{10}b^{20}c^{30}$, $a^{10}b^{1}c^{30}$, $a^{10}b^{1}c^{100}$
level-3 weak distortion:	$a^{10}b^{20}c^{30}$, $a^{1}b^{1}c^{30}$, $a^{1}b^{100}c^{100}$
level-2 strong distortion:	$a^{10}b^{20}c^{30}$, $a^{10}c^{30}$, c^{100}

We denote by $dist_t(w)$ the set of all level-t distortions of w.

Remark 2. *Assuming that* $w \in \Sigma^*$ *is a multiset such that* $card(supp(w)) = k > 0$, *then* $dist_t(w) \subseteq dist_{t+j}(w)$ *for any* $1 \leq t \leq k$ *and* $j \geq 0$ *such that* $t + j \leq k$.

A multiset rewriting rule is a pair (u, v) where u and v are multisets over Σ, $u \neq \lambda$. Typically, such a rule is written as $r : u \to v$, where r is a label through which one can uniquely identify the rule in a set of rules. Given a rule $r : u \to v$, let $left(r) = u$ and $right(r) = v$.

Given a multiset $w \in \Sigma^*$ and a multiset rewriting rule $r : u \to v$, we say that r is *applicable* to w if $left(r) \subseteq w$. Similarly, a multiset of multiset rewriting rules $r_1^{k_1} \dots r_l^{k_l}$ is *applicable* to w if $\sum_{1 \leq i \leq l} k_i * left(r_i) \subseteq w$.

Given a set of multiset rewriting rules R, a multiset $w \in \Sigma^*$, and a multiset of rules $\rho \in R^*$, we say that ρ is *maximally applicable* to w if

- ρ is applicable to w;
- $\rho + r^1$ is not applicable to w for all $r \in R$.

For a multiset of rules $\rho = r_1^{k_1} \dots r_l^{k_l} \in R^*$ we define

- $input(\rho) = \sum_{1 \leq i \leq l} k_i * left(r_i)$;
- $output(\rho) = \sum_{1 \leq i \leq l} k_i * right(r_i)$.

2.2. Register Machines

A *register machine* is a tuple $M = (n, \mathcal{P}, l_0, l_h)$ where $n \geq 1$ is the number of *registers* (each register stores a natural number), $\mathcal{P}$ is a finite set of *instructions* bijectively labelled by elements from the set $lab(\mathcal{P}) = \{l_0, \dots, l_{k-1}\}$, $l_0 \in lab(\mathcal{P})$ is the initial label, and $l_h \in lab(\mathcal{P})$ is the final label. There are three types of instructions:

- $l_i : (add(r), l_j, l_k)$ where $l_i \in lab(\mathcal{P}) \setminus \{l_h\}$, $l_j, l_k \in lab(\mathcal{P})$, $1 \leq r \leq n$, increments the value stored by the register r and non-deterministically proceeds to the instruction labelled by l_j or l_k;
- $l_i : (sub(r), l_j, l_k)$ where $l_i \in lab(\mathcal{P}) \setminus \{l_h\}$, $l_j, l_k \in lab(\mathcal{P})$, $1 \leq r \leq n$, if the value stored by register r is 0 then proceeds to the instruction labelled by l_k, otherwise decrements the value stored by register r and proceeds to the instruction labelled by l_j;
- $l_h : halt$ stops the machine.

M is called deterministic if $l_j = l_k$ for all increment instructions $l_i : (add(r), l_j, l_k) \in \mathcal{P}$.

M starts with all registers containing the value 0 and runs the program $\mathcal{P}$, firstly executing the instruction with the label l_0. Considering the content of register 1 for all possible computations of M which end with the execution of the instruction labelled $l_h : halt$, one obtains the set generated by M (denoted here by $N(M)$).

The following result is due to Minsky [13].

Theorem 1. *For any recursively enumerable set $Q \subseteq \mathbb{N}$ there exists a non-deterministic register machine with 3 registers generating Q such that when starting with all registers being empty, M computes and halts with k in register 1, and registers 2 and 3 being empty iff $k \in Q$.*

The result mentioned in Theorem 1 stands true also for deterministic register machines accepting sets of numbers. More precisely, such a register machine starts by having the number $k \in \mathbb{N}$ in register 1 (all the other registers being empty) and accepts k if the computation halts, executing the instruction labelled $l_h : halt$. As above, three registers are sufficient to accept any recursively enumerable set.

2.3. Parallel Multiset Rewriting Systems

A parallel multiset rewriting system (in short, a PMR system) represents a simplification of a membrane system whose hierarchical tree structure of membranes is reduced to one node (region)—see [11,14] for more details. More formally, a *parallel multiset rewriting system with promoters/inhibitors and catalysts* is a construct $\Pi = (\Sigma, \Delta, C, P, I, \mathcal{R}, w_0)$ where Σ is an alphabet of symbols (*objects*), $\Delta \subseteq \Sigma$ is the output alphabet, $C \subseteq \Sigma$ is the set of *catalysts*, $P \subseteq \Sigma$ is the set of *promoters*, $I \subseteq \Sigma$ is the set of *inhibitors*, $\mathcal{R}$ is a set of *multiset rewriting rules* (which can be *catalysed/promoted/inhibited*), and w_0 is the *initial multiset* of objects.

A rule $r \in \mathcal{R}$ is of the form $r : a \to \alpha$ (non-cooperative) or $r : ca \to c\alpha$ (catalytic) where $a \in \Sigma \setminus C$, $\alpha \in (\Sigma \setminus C)^*$, and $c \in C$. The rules of mentioned types can be promoted (or inhibited, respectively), hence one defines promoted non-cooperative rule $r : a \to \alpha|_p$, where $p \in P$ (inhibited non-cooperative rule $r : a \to \alpha|_{\neg i}$, where $i \in I$, respectively) and catalytic promoted rule $r : ca \to c\alpha|_p$, where $p \in P$ (catalytic inhibited rule $r : ca \to c\alpha|_{\neg i}$, where $i \in I$, respectively).

As described in the preamble of the Section 1, a multiset of rules $\rho \in \mathcal{R}^*$ is *applicable* to a multiset of objects $w \in \Sigma^*$ if there are enough objects in w to trigger all the rules from ρ with their corresponding multiplicity and, in case of promoted (inhibited) rules, if the specified promoters (inhibitors) are in w (are not in w). The applicable multiset of rules ρ is *maximal* if there is no applicable multiset of rules $\rho' \in \mathcal{R}^*$ to w and such that $\rho \subset \rho'$ (in the multiset sense).

It is worth mentioning that catalysts inhibits the parallelism of the system (as they are counted as any other objects) while the promoters/inhibitors only guide the computation (as the presence/absence of a promoter/inhibitor in the multiset of objects makes the corresponding promoted/inhibited rules to be available for application as many times as possible).

The system Π evolves by applying a maximal multiset of rules to the initial multiset (configuration), then applying iteratively another maximal multiset of rules to the multiset obtained in the previous step (i.e., the next configuration) and so on. In other words a transition between configurations C_1 and C_2 of Π is determined by an application of a

maximal multiset of rules in parallel. More precisely, if ρ is a maximal multiset of rules on C_1, then $C_2 = C_1 \setminus input(\rho) + output(\rho)$.

A computation of Π is a sequence of configurations such that the first configuration in the sequence is the initial configuration and between each two consecutive configurations there exists a transition (which is determined by an application of a maximal multiset of rules). A computation is successful if this sequence is finite, namely there is no rule applicable to the objects present in the last configuration–in this case, the number or the Parikh vector corresponding to the multiset of objects from Δ present in the last configuration of Π is considered to be the result of the underlying computation. By collecting the results of all successful computation of Π one gets the set of numbers/vectors of numbers generated by Π; they will be denoted by $N(\Pi)$ and $Ps(\Pi)$, respectively.

The familly of all sets of numbers (vectors) of numbers generated by PMR systems with a list of features $f \in \{ncoo, cat_k, pro, inh\}$ (indicating the usage of non-cooperative rules, catalytic rules with at most k catalysts, promoters, and inhibitors, respectively) is denoted by $NOP(f)$ (or $PsOP(f)$, respectively).

It is known that $NOP(ncoo, pro) = NOP(ncoo, inh) = NET0L$ and $NOP(cat, pro) = NOP(cat, inh) = NRE$ (see [15]).

3. Towards "Distorted" Computations

Given a non-cooperative multiset rewriting rule $r : a \to \alpha$, a t-level distortion of r is a multiset rewriting rule $\tilde{r} : a \to \tilde{\alpha}$ where $\tilde{\alpha} \in dist_t(\alpha)$. Similarly, given a catalytic multiset rewriting rule $r : ca \to c\alpha$, a t-level distortion of r is a multiset rewriting rule $\tilde{r} : ca \to c\tilde{\alpha}$ where $\tilde{\alpha} \in dist_t(\alpha)$. Distorted non-cooperative or catalytic rules can be promoted or inhibited as described above.

A *parallel multiset rewriting system with promoters and/or inhibitors with a t-distorted computation*, where $t > 0$ (in short a PMRD$_t$ system) $\Pi_t = (\Sigma, \Delta, C, P, I, \mathcal{R}, w_0)$ has all the components defined as for a PMR system but with the semantics of the computation modified in the following way. To compute a step leading to a configuration C_{j+1} from a configuration C_j (that is, to perform a transition between configurations), Π_t non-deterministically selects a maximal applicable multiset of rules ρ and for each instance r of a rule in ρ, Π_t applies a t-level strong distortion of r but such that at least one such application is weak for any given species of rules in ρ.

For example, let the selected multiset of rules be $\rho = r_1^{k_1} \dots r_j^{k_j}$. Accordingly, for the species of rules r_i, $1 \leq i \leq j$, there will be k_i applications of the (distorted) rule r_i that Π_t has to perform. Among these applications, at least one of them has to be a weak t-level distortion of r_i (while the rest of them might be weak or strong distortions of r_i).

As defined for a PMR system, Π_t starts from the initial configuration and performs transitions between configurations (consequently, yielding a sequence of configurations). The computation produces an output if this sequence of configurations is finite (that is, the last configuration in the sequence is the halting configuration in which no rule is applicable anymore).

By $N(\Pi_t)$ (and $Ps(\Pi_t)$, respectively) we denote the set of numbers (or vectors, respectively) of numbers generated by Π_t. By $NOP_{d=t}(f)$ (or $PsOP_{d=t}(f)$, respectively) we denote the family of all sets of numbers (vectors) of numbers generated by PMRD systems with a list of features $f \in \{ncoo, cat_k, pro, inh\}$ and t-distorted computations.

A PMRD system Π_t, $t \geq 1$, is called to be *permissive to distortion* if, in any given configuration during the computation, no matter how a t-level distortion $\tilde{r}$ of each instance of a rule r in the multiset of applicable rules ρ is selected, Π_t generates the same set of numbers/vectors.

Example 2. *In order to better explain the functioning of a PMRD$_1$ system $\Pi_1 = (\Sigma, \Delta, C, P, I, \mathcal{R}, w_0)$, assume that $\mathcal{R} = \{r_1 : a \to bd, r_2 : ca \to ce\}$ and Π_1 is in configuration $C = ca^3$. Then one can notice that there exist two maximal applicable multisets of rules to C, namely r_1^3 and $r_1^2 r_2$.*

The system Π_1 is "highly non-deterministic" as it has to apply level-1 distorted rules. Here we present few (among many others) next configurations obtained by applying the multiset of rules r_1^3 on C (recall that in this case we have to apply 3 times the level-1 distorted rule $\tilde{r}_1$; at least one of them has to be a weak level-1 distortion of r_1 while the rest of them can be strong): cbd, cbd^3, cb^3d, $cb^{10}d^3$, $cb^{10}d^{11}$, etc.

Similarly, if on configuration C is applied the multiset of rules $r_1^2 r_2$ then some examples of next configurations are: $cebd$, $cebd^{10}$, $ce^{10}b^5d^3$, etc. Recall that in this case a single weak level-1 distortion of r_2 was applied; additionally, it was applied one weak level-1 distortion of r_1 and one strong level-1 distortion of r_1.

Example 3. *We sketch an example of a distorted P System with promoters and one catalyst which is able to generate the non-semilinear set $\{2^n \mid n \geq 1\}$. A common approach to generate this set of numbers is to consider several rules whose iterative applications determine the system to double the number of an object (at each iteration); in addition, the constructed system has to provide the rules to break out the cycle, hence to end the computation.*

As some of the techniques employed in what follows were also used to prove the equivalence between distorted P systems with mentioned features and register machines, here we only detail the rules that, starting from a configuration $cXYa^{2^k}$, determine the system to reach the configuration $cX^{l_1}\overline{a}^{2^{k+1}}$, where $l_1 \geq 1$. The general idea employed in this example is to use the catalyst c to sequentially double the number of symbols a into $\overline{a}$, checking at the same time if the distortion does not affect the number of target objects $\overline{a}$ produced during the application of the rules. The rules are grouped according to the moment of their execution.

step 1	$X \to X_1 \mid a$	*step 3*	$cb \to co \mid \overline{b}$	*step 5*	$X_3 \to \# \mid \overline{\overline{b}}$
	$ca \to c\overline{a}\overline{a}b \mid X$		$\overline{b} \to \overline{\overline{b}}$		$\overline{\overline{b}} \to \lambda$
	$Y \to Y_1$		$X_2 \to X_3$		
step 2	$cb \to c\overline{b} \mid X_1$	*step 4*	$X_3 \to XY \mid o$		$\# \to \#$
	$X_1 \to X_2$		$\overline{\overline{b}} \to \overline{\overline{b}}$		
	$X \to t \mid Y_1$		$o \to \lambda$		
	$Y_1 \to \lambda$				

In the first step, the distorted rule $ca \to c\overline{a}\overline{a}b \mid X$ is applied only once because the current multiset contains only one copy of object c. In this case, the level-1 distortion can affect either the number of output objects $\overline{a}$ or the number of output objects b. Using level-1 distorted catalytic rules and promoters, one can effectively check if the number of objects b is greater than 2 (hence one can determine if b was actually the subject of distortion). This is done in the steps 2 and 3 by the rules $cb \to c\overline{b} \mid X_1$ and $cb \to co \mid \overline{b}$ (the last rule being applied only if there were more objects b present in the second configuration, hence implying that b was the subject of distortion). It follows that, in case the object o is generated, then the promoters X and Y are produced, hence the rewriting of another object a can start over. Proceeding in this way for all the objects from the initial multiset $cXYa^{2^k}$ one gets in the last configuration 2^{k+1} copies of the object $\overline{a}$. It is important to highlight that in case all the objects a were rewritten, in the first step only the rule $Y \to Y_1$ is executed; in the second step all the object Y_1 are deleted by the rule $Y_1 \to \lambda$ and the computation halts.

Finally, in order to generate the set $\{2^n \mid n \geq 1\}$ one has to provide the rules that rewrite all the objects $\overline{a}$ in their corresponding counterparts a. This can be done using a similar technique to the one presented above (that is, by sequentially rewriting $\overline{a}$ into a by using a level-1 distorted promoted catalytic rule and checking if the distortion did not affect the multiplicity of the output objects a).

The following result shows that the class of sets of numbers generated by PMRD systems with level-1 distorted non-cooperative multiset rewriting rules is at most equal to the class of sets of numbers generated by ET0L systems.

Theorem 2. $NOP_{d=1}(ncoo, pro) \subseteq NET0L$.

Proof. Let us consider a PMRD$_1$ system $\Pi_1 = (\Sigma, \Delta, C, P, I, \mathcal{R}, w_0)$ with promoted non-cooperative multiset rewriting rules ($C = I = \emptyset$) and with 1-distorted computation. Without loss of generality we may assume that all the rules of Π_1 are promoted (recall that a non-cooperative multiset rewriting rule $a \to \beta \in \mathcal{R}$ can be written as $a \to \beta|a$). Hence, let

$$\begin{aligned}\mathcal{R} = \{ \quad & r_1 : b_1 \to b_{(1,1)}^{s_{(1,1)}} \dots b_{(1,k_1)}^{s_{(1,k_1)}}|_{p_1}, \\ & r_2 : b_2 \to b_{(2,1)}^{s_{(2,1)}} \dots b_{(2,k_2)}^{s_{(2,k_2)}}|_{p_2}, \\ & \dots \\ & r_n : b_n \to b_{(n,1)}^{s_{(n,1)}} \dots b_{(n,k_n)}^{s_{(n,k_n)}}|_{p_n} \quad \}\end{aligned}$$

As mentioned in Section 2 we know that $NOP(ncoo, pro) = NET0L$, hence it is sufficient to construct a PMR system $\overline{\Pi} = (\overline{\Sigma}, \overline{\Delta}, \overline{C}, \overline{P}, \overline{I}, \overline{\mathcal{R}}, \overline{w_0})$ with promoted non-cooperative multiset rewriting rules ($\overline{C} = \overline{I} = \emptyset$) that simulates the computation of Π_1.

Let us consider the finite sets $\Sigma' = \{\overline{a} \mid a \in \Sigma\}$, $\Sigma'' = \{\overline{\overline{a}} \mid a \in \Sigma\}$.

We define

$$\begin{aligned}\overline{\Sigma} &= \Sigma \cup \Sigma' \cup \Sigma'' \\ &\cup \{r_i, r_{(i,j)} \mid 1 \le i \le n, 1 \le j \le k_i\} \cup \{t, S, X, \overline{X}, Y, \overline{Y}\}\end{aligned}$$

Also, let the following morphisms be:

- $h' : \Sigma \to \Sigma'$, defined by $h'(a) = \overline{a}$, for all $a \in \Sigma$;
- $h'' : \Sigma \to \Sigma''$, defined by $h''(a) = \overline{\overline{a}}$, for all $a \in \Sigma'$;

The set of rules $\overline{P}$ is defined as follows:

- for $1 \le i \le n$, the following rules (type 1) are added to $\overline{P}$:

$$b_i \to h'(b_i) t r_i|p_i$$

- for $1 \le i \le n$, $1 \le j \le k_i$, the following rules (type 2) are added to $\overline{P}$:

$$r_i \to r_{(i,j)}|t$$

- for $1 \le i \le n$, the following rules (type 3) are added to $\overline{P}$:

$$\begin{aligned}& h'(b_i) \to h''\left(\mathtt{rm}_{b_{(i,1)}}(b_{(i,1)}^{s_{(i,1)}} \dots b_{(i,k_i)}^{s_{(i,k_i)}})\right) t|r_i \\ & \dots \\ & h'(b_i) \to h''\left(\mathtt{rm}_{b_{(i,k_i)}}(b_{(i,1)}^{s_{(i,1)}} \dots b_{(i,k_i)}^{s_{(i,k_i)}})\right) t|r_i\end{aligned}$$

- for $1 \le i \le n$, $1 \le j \le k_i$, the following rules (type 4) are added to $\overline{P}$:

$$\begin{aligned}& r_{(i,j)} \to h''(b_{(i,j)}) r_{(i,j)} t \\ & r_{(i,j)} \to h''(b_{(i,j)}) r_i t\end{aligned}$$

- the following rules (type 5) are added to $\overline{P}$:

$$S \to SXY|t$$
$$X \to \overline{X}$$
$$Y \to \lambda|t$$
$$\overline{X} \to \lambda$$
$$Y \to \overline{Y}|\overline{X}$$

- the following rules (type 6) are added to $\overline{P}$:

$$h''(a) \to at|\overline{Y} \text{ for all } a \in \Sigma$$
$$t \to \lambda$$

Rules of type 5 indicate a "control" branch in the computation of $\overline{\Pi}$ that is executed in parallel with the main branch responsible for the simulation of the level-1 distorted rules applied by Π. The cycle $S \to SXY|t$ produces at each step the objects X and Y, but only if an object t is present in the respective configuration of $\overline{\Pi}$. The object t is produced during the applications of the rules from the main branch (also, in the main branch, the objects from Σ are rewritten programmatically into their corresponding counterparts by the morphisms h' and h''). It is worth mentioning that in the case that the object t is not produced (recall that if present in a configuration, t is deleted by the type 6 rule $t \to \lambda$) the cycle $S \to SXY|t$ is interrupted, hence $\overline{Y}$ will be produced (the rule $Y \to \lambda|t$ cannot run because the object t is missing; this allows the execution of the rule $Y \to \overline{Y}|\overline{X}$). The object $\overline{Y}$will determine the rewriting of the type-6 rules $h''(a) \to at|\overline{Y}$ for all the symbols $a \in \Sigma$ from the current configuration (hence the simulation can start over).

The main branch simulates the applications of the (distorted) rules of Π as follows. Firstly, the rules of type $b_i \to h'(b_i)tr_i|p_i$ are applied. Their role is

- to "paint" the objects that trigger the application of a multiset of rules of Π in a given configuration;
- to produce a "witness" object t indicating that a simulation of an application of a rule is in progress; the object t will forbid the generation of $\overline{Y}$ in the control branch;
- to produce a "selector" object r_i– an object that will be used to select which distorsion of the simulated rule has to be applied.

Consequently the selector r_i is rewritten by the type 2 rules into $r_{(i,1)}, \ldots, r_{(i,k_i)}$. An object $r_{(i,j)}$ is used as a promoter to indicate the species of objects that will be affected by the distortion in rule i (namely $b_{(i,j)}$)

Correspondingly, the rule of type 3 is executed:

$$h'(b_i) \to h''\left(\mathtt{rm}_{b_{(i,j)}}(b_{(i,1)}^{s_{(i,1)}} \ldots b_{(i,k_i)}^{s_{(i,k_i)}})\right)t|r_{(i,j)}$$

As described before, the "witness" object t prevents the generation of $\overline{Y}$ in the control branch.

Next, the rules of type 4 are executed. The rule $r_{(i,j)} \to h''(b_{(i,j)})r_{(i,j)}t$ represents a cycle which generates objects $h''(b_{(i,j)})$. Due to non-determinism, the cycle can be interrupted by the rule $r_{(i,j)} \to h''(b_{(i,j)})r_it$. In this way, after the end of applications of sequences of rules for all possible cases (recall that one has to simulate a multiset of distorted rules of Π) the resulting configuration will contain the "painted" objects by morphism h'' (and no object $r_{(i,j)}$). At that time, because the object t is not generated anymore, the control cycle produces object $\overline{Y}$. This will determine the execution of the rules $h''(a) \to at|\overline{Y}$ for all $a \in \Sigma$, hence the simulation can start over.

The simulation continues in the same manner until Π_1 stops (in the last configuration of Π_1 no rule can be applied). Correspondingly, $\overline{\Pi}$ stops as well because the rules of type 1 cannot be triggered.

Consequently we conclude that $NOP_{d=1}(ncoo, pro) \subseteq NET0L$. □

The following result proves that parallel multiset rewriting systems with level-1 distorted rules are Turing universal.

Theorem 3. $NOP_{d=1}(cat_1, pro) = NRE$.

Proof. We prove the result by double inclusion. $NRE \supseteq NOP_{d=1}(cat_1, pro)$ is assumed true by Turing-Church Thesis. The opposite inclusion, namely $NOP_{d=1}(cat_1, pro) \supseteq NRE$, is proved by simulating a register machine $M = (n, \mathcal{P}, l_0, l_h)$. Starting from the definition of M one can construct a PMRD$_1$ system $\Pi_1 = (\Sigma, \Delta, C, P, I, \mathcal{R}, w_0)$ as follows:

$$\begin{aligned}
\Sigma &= \{a_i \mid 1 \leq i \leq n\} \cup \{l, \bar{l}, A_l, B_l, \mid l \in lab(\mathcal{P})\} \\
&\cup \{X_l, \overline{X_l}, \overline{\overline{X_l}}, Y_l, \overline{Y_l}, \overline{\overline{Y_l}} \mid l \in lab(\mathcal{P})\} \cup \{\#\}; \\
\Delta &= \{a_i \mid 1 \leq i \leq n\}; \\
C &= \{c\}; \\
P &= \{B_{l_i}, X_{l_i}, \overline{X_{l_i}}, \overline{\overline{X_{l_i}}}, \overline{\overline{Y_{l_i}}} \mid l_i \in lab(\mathcal{P}), l_i \text{ label of } add \text{ instruction}\} \\
&\cup \{l_i, X_{l_i}, Y_{l_i} \mid l_i \in lab(\mathcal{P}), l_i \text{ label of } sub \text{ instruction}\}; \\
I &= \emptyset; \\
w_0 &= cl_0.
\end{aligned}$$

The set of rules $\mathcal{R}$ is defined in the following way:

• for each increment instruction $l_i : (add(r), l_j, l_k)$ we add the following rules to $\mathcal{R}$:

$$\begin{array}{ll}
l_i \to A_{l_i} B_{l_i} & \bar{l}_i \to l_j | \overline{\overline{X_{l_i}}} \\
cA_{l_i} \to ca_r \bar{l}_i X_{l_i} | B_{l_i} & \bar{l}_i \to l_k | \overline{\overline{X_{l_i}}} \\
B_{l_i} \to \lambda & Y_{l_i} \to \overline{Y_{l_i}} \\
A_{l_i} \to \lambda | X_{l_i} & \overline{Y_{l_i}} \to \overline{\overline{Y_{l_i}}} \\
cX_{l_i} \to c\overline{X_{l_i}} Y_{l_i} & \overline{\overline{Y_{l_i}}} \to \lambda \\
cX_{l_i} \to c\overline{\overline{X_{l_i}}} | \overline{X_{l_i}} & \bar{l}_i \to \# | \overline{\overline{Y_{l_i}}} \\
\# \to \# &
\end{array}$$

• for each decrement instruction $l_i : (sub(r), l_j, l_k)$ we add the following rules to $\mathcal{R}$:

$$\begin{array}{ll}
ca_r \to cX_{l_i} | l_i & Y_{l_i} \to \overline{Y_{l_i}} \\
l_i \to \bar{l}_i Y_{l_i} & \bar{l}_i \to l_k | \overline{Y_{l_i}} \\
\bar{l}_i \to l_j | X_{l_i} & \overline{Y_{l_i}} \to \lambda \\
X_{l_i} \to \lambda &
\end{array}$$

• for the halt instruction $l_h : halt$ the rule $cl_h \to c$ is added to $\mathcal{R}$.

In this simulation, the number stored by register i, $1 \leq i \leq n$, of M will be represented as the multiplicity of the object a_i. The computation starts from configuration cl_0 (as the register machine starts its computation having all the registers empty).

In general, a configuration of Π_1 (which corresponds to a configuration of M) is a multiset of objects of type $cl_i^k a_1^{s_1} \ldots a_n^{s_n}$. The object l_i represents the label of the next rule to be executed by M, hence that have to be simulated by Π_1 (recall that the multiplicity of l_i can be greater than one as Π_1 runs with level-1 distorted rules).

In case the instruction that has to be simulated is the increment instruction $l_i : (add(r) l_j, l_k)$, then Π_1 will execute in the first step the rule $l_i \to A_{l_i} B_{l_i}$ (which will rewrite all the existing objects l_i) and in the second step the rules $cA_{l_i} \to ca_r \bar{l}_i X_{l_i} | B_{l_i}$ and $B_{l_i} \to \lambda$ (this is done to enforce the execution of the rule $cA_{l_i} \to ca_r \bar{l}_i X_{l_i} | B_{l_i}$ exactly one time in a simulation of an instruction of M).

In the next step, rule $A_{l_i} \to \lambda | X_{l_i}$ is used to clean-up all the remaining (if any) symbols A_{l_i}. Meanwhile, one has to check if the object a_r was not affected by the distortion during the application of the level-1 distorted rule $cA_{l_i} \to ca_r\overline{l_i}X_{l_i}|B_{l_i}$. This is done by using the "dummy" object X_{l_i}; more precisely, if Π_1 detects that in the current multiset there are several objects X_{l_i} then this means that the distorted rule $cA_{l_i} \to ca_r\overline{l_i}X_{l_i}|B_{l_i}$ produced exactly one symbol a_r (that is, the distortion affected the multiplicity of X_{l_i}) hence the system has to proceed to the generation of the next label l_j or l_k. Correspondingly, if Π_1 executes in consecutive steps the rules

$$cX_{l_i} \to c\overline{X_{l_i}}Y_{l_i}$$
$$cX_{l_i} \to c\overline{\overline{X_{l_i}}}|\overline{X_{l_i}}$$

it follows that the object $\overline{\overline{X_{l_i}}}$ is produced and from here on the generation of objects l_j or l_k will take place (as one of the rules $\overline{l_i} \to l_j|\overline{\overline{X_{l_i}}}$ or $\overline{l_i} \to l_k|\overline{\overline{X_{l_i}}}$ will run; it is also worth to mention that although there is exactly one object $\overline{l_i}$, after applying the above mentioned rules there might exist multiple copies of objects l_j or l_k, respectively).

However, due to distortion it might happen that $cX_{l_i} \to c\overline{\overline{X_{l_i}}}|\overline{X_{l_i}}$ is not executed (meaning that other symbol than X_{l_i} was the subject of distortion while $cA_{l_i} \to ca_r\overline{l_i}X_{l_i}|B_{l_i}$ was applied or the distortion on the species X_{l_i} did not produce strictly more than one object). If this is the case, then the object Y_{l_i} will be rewritten (in several steps) into $\overline{\overline{Y_{l_i}}}$ and it will be used as a promoter in rule $\overline{l_i} \to \#|\overline{\overline{Y_{l_i}}}$. Once the object # is generated the rule # $\to$ # will run forever, consequently yielding no result.

If the instruction that has to be simulated is the decrement instruction $l_i : (sub(r)l_j, l_k)$ then Π_1 executes the rule $l_i \to \overline{l_i}Y_{l_i}$ and possibly $ca_r \to cX_{l_i}Y_{l_i}|l_i$. If the current configuration contains at least one object a_r, then one instance of the object a_r is deleted by the last mentioned rule. It is worth to point out that in the first step all the objects l_i are rewritten by the rule $l_i \to \overline{l_i}Y_{l_i}$, hence $ca_r \to cX_{l_i}Y_{l_i}|l_i$ can be executed only once (as being promoted by l_i). Next, in the second step, the rules $\overline{l_i} \to l_j|X_{l_i}$, $X_{l_i} \to \lambda$ and $Y_{l_i} \to \overline{Y_{l_i}}$ are executed, hence the object l_j (corresponding to the label of the next register machine instruction) is generated. The object $\overline{Y_{l_i}}$ will be deleted later on by the rule $\overline{Y_{l_i}} \to \lambda$.

On other hand, if the rule $ca_r \to cX_{l_i}Y_{l_i}|l_i$ is not executed because there are no objects a_r in the current configuration, then in the first step only the rule $l_i \to \overline{l_i}Y_{l_i}$ will be executed. In the second step the rule $Y_{l_i} \to \overline{Y_{l_i}}$ will run. Finally, in the third step the rules $\overline{l_i} \to l_k|\overline{Y_{l_i}}$ and $\overline{Y_{l_i}} \to \lambda$ will generate the object l_k (corresponding to the label of the next register machine instruction).

Finally we are pointing out that in case the object l_h is produced then Π_1 stops.

We also emphasize that the construction of the $PMRD_1$ system presented in the proof of Theorem 3 can be easily modified to simulate an arbitrary deterministic register machine which accepts a set of numbers. Moreover, the constructed system Π_1 is permissive to 1-level distortions.

Consequently we conclude that $NOP_{d=1}(cat_1, pro) = NRE$. □

4. Conclusions

Many mathematical models proposed to capture natural/artificial phenomena can be represented by using discrete input–output maps. In this paper we defined multiset rewriting systems with t-distorted computations. These types of P systems were proposed to reflect the uncertainty related with non-exact knowledge of the parallel rewriting processes (which, in particular, can model bio-chemical, physical, economical sociological, or other types of phenomena) for which one knows the inputs but does not know the exact outputs. To this aim we defined level-t distorted multiset rewriting rules which assumes that, if applied in a derivation step of a PMRD system, the multiplicities of at most t species (non-deterministically selected) of the objects in the output are not known. We succeeded to find an upper bound for

the computational power of PMRD systems with level-1 distorted non-cooperative promoted rules, namely we succeeded to prove that $NOP_{d=1}(ncoo, pro) \subseteq NET0L$. On the other hand, when level-1 distorted catalytic promoted rules (with one catalyst) are used, the model proves to be computationally universal.

There are several problems left open. A promising research line regards the usage of other rewriting formalisms and types of rules. For example, although it was mentioned in this paper, the case of PMRD systems with inhibited rules was not investigated. However, for this types of PMRD systems we conjecture the existence of similar results (namely $NOP_{d=1}(ncoo, inh) \subseteq NET0L$ and $NOP_{d=1}(cat_1, inh) = NRE$). Moreover, the cases when $d > 1$ are also left open. Yet another research topic regards an extensive study of systems which are permissive to distortions. In this respect one might also consider developing ultimately confluent systems with distorted rules (see [9]).

Author Contributions: C.S.: Conceptualization, formal analysis; D.-F.S.: Conceptualization, investigation, writing. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Alhazov, A.; Freund, R.; Ivanov, S.; Pan, L.; Song, B. Time-freeness and Clock-freeness and Related Concepts in P systems. *Theor. Comput. Sci.* **2020**, *805*, 127–143. [CrossRef]
2. Cavaliere, M.; Sburlan, D. Time and Synchronization in Membrane Systems. *Fundam. Inform.* **2005**, *64*, 65–77.
3. Csuhaj-Varjú, E.; Freund, R.; Sburlan, D. Modeling Dynamical Parallelism in Bio-systems. *Lect. Notes Comput. Sci.* **2006**, *4361*, 330–351.
4. Freund, R. Asynchronous P Systems and P Systems Working in the Sequential Mode. *Lect. Notes Comput. Sci.* **2005**, *3365*, 36–62.
5. Sburlan, D. Clock-free P systems. In Proceedings of the Fifth Workshop on Membrane Computing, Milan, Italy, 14–16 June 2004; pp. 372–383.
6. Song, B.; Pérez-Jiménez, M.J.; Pan, L. An efficient time-free solution to SAT problem by P systems with proteins on membranes. *J. Comput. Syst. Sci.* **2016**, *82*, 1090–1099. [CrossRef]
7. Alhazov, A.; Freund, R.; Ivanov, S. P Systems with Randomized Right-hand Sides of Rules. *Theor. Comput. Sci.* **2020**, *805*, 144–160. [CrossRef]
8. Paun, G. Computing with Membranes. *TUCS Tech. Rep. Turku Cent. Comput. Sci.* **1998**, *208*, 1–39.
9. Alhazov, A.; Sburlan, D. Ultimately Confluent Rewriting Systems. Parallel Multiset–Rewriting with Permitting or Forbidding Contexts. *Lect. Notes Comput. Sci.* **2005**, *3365*, 178–189.
10. Rozenberg, G.; Salomaa, A. (Eds.) *Handbook of Formal Languages*; Springer: Berlin, Germany, 1997.
11. Paun, G.; Rozenberg, G.; Salomaa, A. (Eds.) *The Oxford Handbook of Membrane Computing*; Oxford University Press: London, UK, 2009.
12. Rozenberg, G.; Salomaa, A. *The Mathematical Theory of L Systems*; Academic Press: New York, NY, USA, 1980.
13. Minsky, M. *Computation: Finite and Infinite Machines*; Prentice Hall: Upper Saddle River, NJ, USA, 1967.
14. The P Systems Website. Available online: http://ppage.psystems.eu/ (accessed on 26 December 2020).
15. Sburlan, D. Further Results on P Systems with Promoters/Inhibitors. *Int. J. Found. Comput. Sci.* **2006**, *17*, 205–221. [CrossRef]