Natural Computing
https://doi.org/10.1007/s11047-024-09986-1

RESEARCH

On the spectrum between reaction systems and string rewriting

Artiom Alhazov[1] · **Rudolf Freund**[2] · **Sergiu Ivanov**[3]

Accepted: 21 March 2024

Abstract
Reaction systems are a model of computing aiming to formalize biochemistry by capturing the qualitative relations between the species, and explicitly discarding any accounts of multiplicity. From the point of view of the formal language theory, this situates them in the realm of set rewriting. In this work, we propose a series of extensions of reaction systems to use strings. These extensions form a spectrum in the sense that all of them honor the hallmark features of the original model: the threshold principle and the non-permanency principle. We thoroughly discuss the details of the structure and the behavior of these variants, and commence studying their expressive power by comparing them to some classic models of computing.

Keywords Reaction systems · String rewriting · Computational power

Mathematics Subject Classification 68Q07 · 68Q10 · 68Q42 · 68Q45

1 Introduction

Reaction systems are a model of computing aiming to abstract the operation of biochemical reactions in a formal framework (Brijder et al. 2011; Ehrenfeucht and Rozenberg 2007). Rather than delving in quantitative details of biochemistry, reaction systems focus on the higher-level qualitative information that can be derived from pre-existing biological knowledge, and are particularly suited to expressing qualitative relations between entities. Starting from this postulate, reaction systems are defined to only qualify the entities of interest as present or absent, omitting the representation of any kind of multiplicity. Furthermore, when an entity is available in the system, it is considered to be available in unlimited amounts—this is called the *threshold principle*. This in particular excludes competition for resources, since the presence of any entity is always sufficient for all interactions it is implicated in. Thus entities in reaction systems resemble Boolean variables, as discussed already in the seminal paper (Brijder et al. 2011), and also in Alhazov et al. (2023).

While reaction systems set aside any quantitative data and competition for resources, they still aim to stay close to biochemical systems by imposing mandatory degradation of the species—if an entity is not explicitly sustained (reproduced), it must disappear from the system. In the reaction system literature this is typically referred to as the *non-permanency principle*.

From the point of view of formal language theory, these characteristics of reaction systems situate them in the realm of set rewriting models of computing—the state of a reaction system is defined as the set of species which are currently present, and transitions happen by replacing the current state with the new one, given by the right-hand sides of the reactions. While set rewriting is a natural concept, it has attracted less attention, and formal language theory has historically been more focused on rewriting strings [e.g. Rozenberg and Salomaa (1997)], especially because set rewriting can be seen as a form of finite-state transition system.

The starting motivation of the present work is to extend reaction systems by considering ways of introducing strings. On the one hand, this enriches the modelling potential of

✉ Sergiu Ivanov
sergiu.ivanov@universite-paris-saclay.fr

Artiom Alhazov
artiom@math.md

Rudolf Freund
rudi@emcc.at

[1] Vladimir Andrunachievici Institute of Mathematics and Computer Science, State University of Moldova, Academiei 5, 2028 Chişinău, Moldova

[2] Faculty of Informatics, TU Wien, Favoritenstraße 9–11, 1040 Vienna, Austria

[3] IBISC, Univ. Évry, Paris-Saclay University, 23, boulevard de France, 91034 Évry, France

Published online: 11 April 2024

reaction system, since such new variants operating on strings may be used to represent additional details in real-world phenomena. On the other hand, extensions generally modulate the computational power in non-trivial ways, which allows for further theoretical exploration. Finally, characterizing the expressive power of such formalisms also gives important hints with respect to feasibility of model checking, as exemplified in the works (Męski et al. 2015, 2017, 2019; Azimi et al. 2015, 2016, 2017).

Direct connections between reaction systems and string rewriting have already been explored before, e.g. Brijder et al. (2011), Dennunzio et al. (2015b), but mostly for characterizing computational power by comparing the behavior of reaction systems to that of automata operating on strings. Reaction automata—an extension of reaction systems operating on multisets and recognizing strings—have been extensively considered as well (Okubo et al. 2012b, a; Yokomori and Okubo 2021). This model of computing operates on multisets and recognizes strings by sequentially reading symbols from the input, somewhat in the spirit of P automata (Csuhaj-Varjú and Vaszil 2002). However, reaction automata do not respect the original non-permanency principle, and their reactions operate on multisets rather than on strings.

In this paper, we give several ways of introducing strings in reaction systems, and show that they form, in fact, a spectrum in the following sense:

1. all proposed variants are subject to the threshold principle and the non-permanency principle as the original reaction systems, implying in particular the absence of explicit competition for resources,
2. the proposed variants are progressively closer to string rewriting.

We commence here the exploration of this spectrum between reaction systems and string rewriting. We start by formulating intuitive expectations of the formalism in Sect. 3, and we then introduce a formal framework for extending reaction systems with string rewriting mechanisms in Sect. 4, somewhat in the spirit of Freund and Verlan (2007). In Sect. 5, we set up theoretical tools for comparing the behaviors of thus extended reaction systems to the original model. Then we propose three restrictions of this general formalism in Sect. 6, to which we refer by the terms "reaction systems with strings of type 0", "type 1", and "type 2", in reference to the Chomsky hierarchy of formal grammars. We study the computational power of these variants by comparing them to the original reaction systems and to other well-known models of computing—register machines and Turing machines. We conclude by discussing some of the subtleties, and by drawing the general landscape of the questions remaining open (Sect. 7).

2 Preliminaries

In this section we briefly recall the notions of reaction systems and formal language theory necessary for reading this paper. For a more extensive introduction and overview we recommend (Rozenberg and Salomaa 1997; Brijder et al. 2011; Ivanov and Petre 2020).

Given a set A, we denote by $\mathcal{P}(A)$ the power set of A, i.e., the set of all subsets of A. A multiset over A is a function $w : A \to \mathbb{N}$ assigning to every element $x \in A$ its multiplicity $w(x)$. We will often represent a multiset by any string containing the elements of w in the same number of copies. We denote the set of all multisets over A by A°.

2.1 Reaction systems

Given a finite set S—called the background set of species—a reaction in S is a triple $a : (R_a, I_a, P_a)$, with $R_a, I_a, P_a \subseteq S$, and $R_a \cap I_a = \emptyset$. The subsets of species R_a, I_a, and P_a are called the reactants, the inhibitors, and the products of reaction a.

Given a subset of species $W \subseteq S$, reaction a is said to be enabled in W if $R_a \subseteq W$ and $I_a \cap W = \emptyset$. The result of a on W is denoted by $\mathsf{res}_a(W)$ and is equal to the set of products of a: $\mathsf{res}_a(W) = P_a$. Given a set of reactions A, all of which are enabled in W, the result of A is $\mathsf{res}_A(W) = \bigcup_{a\in A} \mathsf{res}_a(W) = \bigcup_{a\in A} P_a$. A reaction system over the background set S is a tuple $\mathcal{A} = (S, A)$, where A is a set of reactions in S. The result of $\mathcal{A}$ on W is defined as the result of all its reactions: $\mathsf{res}_\mathcal{A}(W) = \mathsf{res}_A(W)$. A state of a reaction system $\mathcal{A} = (S, A)$ over the background set of species S is any subset of species $W \subseteq S$. A state W is reachable by $\mathcal{A}$ if there exists another state $V \subseteq S$ such that $\mathsf{res}_\mathcal{A}(V) = W$.

Reaction systems are *open* dynamical systems, a feature formally defined in their dynamics. Consider the following sequence of subsets of species, called the context sequence $\gamma = (C_i)_{0\le i\le n}$, $C_i \subseteq S$, $n \in \mathbb{N}$. An interactive process in $\mathcal{A}$ is defined as the pair $\pi_\gamma(\mathcal{A}) = (\delta_\gamma(\mathcal{A}), \tau_\gamma(\mathcal{A}))$, where

- $\delta_\gamma(\mathcal{A}) = (D_i)_{0\le i\le n}$ is the result sequence: $D_i = \mathsf{res}_\mathcal{A}(D_{i-1} \cup C_{i-1})$ for $0 < i \le n$, and $D_0 = \emptyset$,
- $\tau_\gamma(\mathcal{A}) = (W_i)_{0\le i\le n}$ is the state sequence: $W_i = C_i \cup D_i$.

Each element D_i of $\delta_\gamma(\mathcal{A})$ is computed as the result of $\mathcal{A}$ on the previous state W_{i-1}, and each state is computed as the union between the result of $\mathcal{A}$ and the corresponding element of the context sequence. C_0 is the called the initial context of $\pi_\gamma(\mathcal{A})$, W_0 is called its initial state, and W_n is the called the result of $\mathcal{A}$ along the context sequence γ, and is denoted by $\mathsf{res}^\gamma_\mathcal{A}(W_0) = W_n$. If the context sequence γ only consists of empty contexts, then the corresponding interactive process $\pi_\gamma(\mathcal{A})$ is called context-independent. A state W is called

reachable from state V if there exists a context sequence γ such that $W = \mathsf{res}^{\gamma}_{\mathcal{A}}(V)$.

Example 1 Consider the reaction system $\mathcal{A} = (S, A)$, where $S = \{x, y, z\}$ and $A = \{a_1 : (\{x\}, \{y\}, \{z\}), a_2 : (\{y\}, \{x\}, \{z\})\}$. Reaction a_1 is enabled in $\{x\}$, but not in $\{x, y\}$ or $\emptyset$. The result of a_1 on $\{x\}$ is $\mathsf{res}_{a_1}(\{x\}) = \{z\}$. The result of A on $\{x\}$ is $\mathsf{res}_A(\{x\}) = \{z\}$ again, because a_2 is not enabled in $\{x\}$.

Consider the context sequence $\gamma = (C_0 = \{x\}, C_1 = \{y\}, C_2 = \emptyset)$. The interactive process $\pi_\gamma(\mathcal{A})$ consists of the following sequences:

- result sequence: $\delta_\gamma(\mathcal{A}) = (D_0 = \emptyset, D_1 = \{z\}, D_2 = \{z\})$;
- state sequence: $\tau_\gamma(\mathcal{A}) = (W_0 = \{x\}, W_1 = \{y, z\}, W_2 = \{z\})$.

Therefore, $\{z\}$ is reachable from $\{x\}$ as there exists the context sequence γ driving $\mathcal{A}$ from $\{x\}$ to $\{z\}$.

Remark 1 In Brijder et al. (2011), the reactant and the inhibitor sets of a reaction are required to be nonempty, while the seminal paper (Ehrenfeucht and Rozenberg 2007) does not explicitly state this requirement. Later works have studied the influence of this restriction on different aspects ranging from expressive power to modelling, e.g. Azimi et al. (2014), Salomaa (2014, 2015), Dennunzio et al. (2019, 2015a, b), Azimi et al. (2015, 2016). In this paper, we do not explicitly consider the nonemptiness requirement.

2.2 Strings and grammars

The presentation in this section follows the works (Alhazov et al. 2021; Freund 2019).

Given a finite set of symbols (an alphabet) V, we denote V^* the set of all finite strings over V, sometimes also referred to as the free monoid over V. The set V^* includes the empty string λ—the neutral element of the monoid. The set of all non-empty strings is denoted by $V^+ = V^* \backslash \{\lambda\}$. The length of a string $w \in V^*$ is denoted by $|w|$, and the number of occurrences of a symbol $x \in V$ in w is denoted by $|w|_x$.

Given a finite set of symbols V, a string rewriting grammar is a construct $G = (V, T, S, R)$, where V is a finite alphabet of symbols, $T \subseteq V$ is the set of terminal symbols, $S \in V$ is the axiom—the starting symbol, and R is the set of string rewriting rules of the form $\alpha \to \beta$, $\alpha, \beta \in V^*$.

Classically, the left-hand side α is required to be non-empty, but in this work we do not impose this restriction. Allowing empty left-hand sides gives rise to what is typically called context-free insertion rules of the form $\lambda \to \beta$, which effectively insert the new substring β at a non-deterministically chosen position in the string. Grammars in which rules are only allowed to carry out some types of insertion and deletion, with or without context, are typically referred to as insertion-deletion systems and are known to have hierarchies of computational power which do not overlap with the classical Chomksy hierarchy (Verlan 2010; Ivanov 2015).

Given a rule $r : \alpha \to \beta$, applying it to a string $w_1\alpha w_2 \in V^+$, results in the string $w_1\beta w_2$, denoted by $w_1\alpha w_2 \overset{r}{\Rightarrow} w_1\beta w_2$, where $w_1, w_2 \in V^*$. This operation generates the binary relation $\overset{r}{\Rightarrow} \subseteq V^* \times V^*$, i.e., $(w, v) \in \overset{r}{\Rightarrow}$, also written as $w \overset{r}{\Rightarrow} v$ if v can be obtained from w by applying r. Given a set of rules R, we write $\overset{R}{\Rightarrow} = \bigcup_{r \in R} \overset{r}{\Rightarrow}$. Informally, $w \overset{R}{\Rightarrow} v$ means that there exists a rule $r \in R$ such that $w \overset{r}{\Rightarrow} v$. We use the notations $\overset{r}{\Rightarrow}^*$ and $\overset{R}{\Rightarrow}^*$ to refer to the reflexive and transitive closure of the relations $\overset{r}{\Rightarrow}$ and $\overset{R}{\Rightarrow}$ respectively. In other words, $w \overset{r}{\Rightarrow}^* v$ means that v can be obtained from w in zero or more applications of r, and $w \overset{R}{\Rightarrow}^* v$ means that v can be obtained from w by zero or more applications of rules from R. Given the grammar $G = (V, T, S, R)$, the language generated by G is defined as the set of all terminal strings which can be obtained from S:

$$L(G) = \left\{ w \in T^* \mid S \overset{R}{\Rightarrow}^* w \right\}. \tag{1}$$

Example 2 Consider the alphabet $V = \{S, a, b\}$, the set of terminal symbols $T = \{a, b\}$, and the set of rules $R = \{S \to aSb, S \to \lambda\}$. Then the language generated by the grammar $G = (V, T, S, R)$ is $\{a^n b^n \mid n \in \mathbb{N}\}$. Remark that R contains the erasing rule λ, but it is possible to replace it by the rule $S \to ab$, which allows for generating the very similar language $\{a^m b^m \mid m \in \mathbb{N}, m > 0\}$. Both of these languages belong to the class of context-free languages, since they can be generated by rules with exactly one non-terminal in the left-hand side.

In this work, we use the conventional notations for classes of languages generated by string rewriting grammars with rules of different shapes:

1. REG, *regular languages:* rules of the form $A \to cB$, $A \to c$, and $A \to \lambda$, for $A, B \in V \backslash T$, and $c \in T$;
2. CF, *context-free languages:* rules of the form $A \to w$, with $A \in V \backslash T$ and $w \in V^*$;
3. CS, *context-sensitive languages:* rules of the form $\alpha A \beta \to \alpha w \beta$, with $\alpha, \beta \in V^*$, $A \in V \backslash T$, and $w \in V^+$ a non-empty string over V;
4. RE, *recursively enumerable languages:* arbitrary rewriting rules.

The expressive power of string rewriting grammars can be modulated in powerful ways by introducing various con-

trol mechanisms which modify when and how the rules are applied. We refer to Dassow and Păun (1989) for an in-depth overview. In this paper, we focus on random-context and semi-contextual grammars, which extend the applicability conditions of rules as described in the following paragraphs [see also Ivanov and Verlan (2015, 2021)].

Given a finite alphabet V, a semi-contextual rule is the triple $r : (\alpha \to \beta, P, Q)$, where $\alpha \to \beta$ is a string rewriting rule with $\alpha, \beta \in V^*$ and $P, Q \subseteq V^+$ are sets of strings called the permitting and the forbidden context respectively. The rule r is applicable to a string w if the following 3 conditions are satisfied:

1. α is a substring of w,
2. all strings in P are substrings of w,
3. all strings in Q are *not* substrings of w.

A semi-contextual grammar is a grammar whose rules are semi-contextual, and the language of such a grammar is defined as in equation (1) above.

If all elements in the permitting context P and the forbidden context Q of a semi-contextual rule r are individual symbols, i.e., $P, Q \subseteq V$, then r is called a random-context rule. A random-context grammar is a grammar containing only random-context rules.

Example 3 Consider the semi-conditional grammar $G = (V, T, S, R)$ with the alphabet $V = \{S, A, \bar{A}, B, \bar{B}, a, b\}$, the set of terminal symbols $T = \{a, b\}$, and the following set of rules:

$$r_0 : (S \to AB, \emptyset, \emptyset)$$
$$r_1 : (A \to a\bar{A}, \{B\}, \emptyset)$$
$$r_2 : (B \to b\bar{B}, \{\bar{A}\}, \emptyset)$$
$$r_3 : (\bar{A} \to A, \{\bar{B}\}, \emptyset) \quad r_5 : (\bar{A} \to \lambda, \{\bar{B}\}, \emptyset)$$
$$r_4 : (\bar{B} \to B, \{A\}, \emptyset) \quad r_6 : (\bar{B} \to \lambda, \{A, \bar{A}\}, \emptyset)$$

Derivations in this grammar start by applying r_0 which yields the string AB. The only rule applicable at this point is r_1, since all other rules require symbols absent from the string. Applying r_1 yields the string $a\bar{A}B$. The only rule applicable now is r_2, yielding $a\bar{A}b\bar{B}$. A choice between r_3 and r_5 is possible at this step. If r_3 is applied, the string is transformed into $aAb\bar{B}$, and the only rule applicable at this point will be r_4, which will lead to the string $aAbB$. These are therefore the first 5 possible steps of a derivation in G:

$$S \overset{r_0}{\Rightarrow} AB \overset{r_1}{\Rightarrow} a\bar{A}B \overset{r_2}{\Rightarrow} a\bar{A}b\bar{B} \overset{r_3}{\Rightarrow} aAb\bar{B} \overset{r_4}{\Rightarrow} aAbB.$$

From this point on, rules r_1 through r_4 may applied in a 4-step loop, yielding strings of the shape $a^k Ab^k B$. If at some point instead of applying r_3, r_5 is applied, then the string $a^k \bar{A} b^k \bar{B}$ will be rewritten into $a^k b^k \bar{B}$, which will render r_6 applicable, yielding the terminal string $a^k b^k$. Therefore, the language of G is

$$L(G) = \{a^n b^n \mid n \in \mathbb{N}, n > 0\}.$$

Note that the rewriting rules in R are essentially regular rules,[1] except the rule $r_0 : S \to AB$ which is only applied once, while the language $L(G)$ is non-regular. Therefore, adding semi-contextual conditions to the rules strictly increases their computational power. Finally, note that the elements of permitting and forbidden contexts in G are individual symbols, meaning that G is also a random-context grammar.

3 Reaction systems meet strings

In this section, we informally discuss three ways of modifying reaction systems to handle strings, which we then formalize and study in Sect. 6. Clearly, the first step in considering reaction systems with strings is imagining that states are strings rather than sets over a finite alphabet. With this starting point fixed, the possibilities of further modifications and extensions are quite rich, as we briefly show in the following paragraphs.

Reactions as insertions The immediate first challenge is translating reactions to operate on strings. Consider the background set of species S and a reaction $a : (R_a, I_a, P_a)$. Due to the threshold principle postulating that if a resource is present it is always present in sufficient quantities to satisfy any needs, the reactants of a behave rather like a permitting context, instead of a resource that a should consume. Similarly, the inhibitors can be seen as a forbidden context for a. Informally, the reaction a can be seen as an insertion rule with permitting and forbidden contexts of the form $(\lambda \to P_a, R_a, I_a)$.

Direct parallelism Reactions in reaction systems are always applied all at once, in parallel. Since there is effectively no competition for resources and no multiplicities, this implies that every reaction is applied once at every step. If the state and the products P_a are considered to be strings, then reactions extended to strings should be applied in parallel, but not in a maximally parallel mode, because if a is applicable once, it can be applied any arbitrary number of times at the same step. In P systems, this mode is called the set-maximal mode: apply non-extendable *sets* of rules, instead

[1] Classically, regular rules have exactly one non-terminal in the left-hand side, with the right-hand side of the form λ, a, or aB, $a \in T$ and $B \in V \backslash T$. This example includes the renaming rules $\bar{A} \to A$ and $\bar{B} \to B$. These would correspond to ϵ-transitions in the finite automaton, which can be avoided by a simple transformation on the non-terminals of the grammar.

of non-extendable *multisets* of rules, see e.g. Alhazov et al. (2016b).

Employing the set-maximal mode is a very reasonable first approach, but a direct consequence of it is that the properties of the state as a string will not be used at all, as nothing in how the reactions are applied will depend on the number of symbols in the state or on their positions. As a consequence, this way of introducing strings will produce a model of computing of power essentially identical to the original reaction systems (Sect. 6.1).

Preserve multiplicities Consider the state string w over the alphabet of species S and a reaction $a : (R_a, I_a, P_a)$. To make the derivation step depend on the multiplicities of the symbols in w, it is possible to modulate the number of times a is applied by these multiplicities. More concretely, the number of applications of a to w may be defined as a function mul of $|w|_x$, for all $x \in R_a \cup I_a$. The function mul may take into account these multiplicities in different ways, e.g. take the minimum, the maximum, or the sum of the multiplicities of symbols in R_a, I_a, or both. The function mul can therefore be seen as a way to introduce the notion of resources in string-based reaction systems, without necessarily violating the threshold principle.

From the standpoint of computational power, introducing a dependency between the multiplicities of the symbols in the string and the number of times a reaction is applied allows for reliable encoding of natural numbers in the number of copies of a symbol a_r. Indeed, all copies of a_r can be reliably maintained by picking a "good" function mul, and constructing the corresponding reactions compensating for the degradation of symbols, i.e., the non-permanency principle. In fact, we show in Sect. 6.2 that this idea allows for simulating register register machines, given an appropriate family of context sequences.

Preserve positions To go beyond preserving the multiplicities, it is possible to also attach the positions at which the insertions induced by a reaction $a : (R_a, I_a, P_a)$ occur to the locations of the elements of R_a and I_a in the string. Similarly to preserving the multiplicities, this can be achieved by defining a function pos taking the positions of the symbols in R_a, I_a, or both, and producing the set of positions at which the insertions of P_a should happen.

This particular extension brings the model of computing even closer to string rewriting, while still explicitly maintaining compliance with the threshold principle and the non-permanency principle. In Sect. 6.3, we show how reaction systems extended in this way can simulate Turing machines, given an appropriate family of context sequences.

4 Reaction systems with strings

As with conventional reaction systems in Sect. 2.1, we start with the background set of species V—a finite alphabet of symbols. A state w is any finite string $w \in V^*$. To streamline the definitions in this section, we also introduce a slight generalization of the notion of string rewriting grammars as defined in Sect. 2.2.

Definition 1 A *string transformer* is a construct $G = (V, R)$, where V is a finite alphabet of symbols and R is a set of string rewriting rules.

With respect to the definition of a string rewriting grammar, that of a string transformer omits the starting symbol S as well as the terminal alphabet T. Semantics-wise, *applying* a string transformer G to a string $w \in V^*$ consists in applying the applicable rules of G to w, according to the mode—sequential, maximally parallel, etc.—as in the case of string rewriting grammars and their different flavors.

Definition 2 A *reaction system with strings* (RSS) is a construct $rss = (G, \mathsf{mctx}, \mathsf{mode}, \mathsf{apply}, \mathsf{post})$, where:

- $G = (V, R)$ is a string transformer;
- $\mathsf{mctx} : V^* \times V^* \to \mathcal{P}(V^*)$ is the context merging function, defining how to merge a context into a state[2];
- $\mathsf{mode} : V^* \to \mathcal{P}(R^\circ)$ is the derivation mode, defining which multisets of rules R are applicable to a state in V^*;
- $\mathsf{apply} : R^\circ \times V^* \to \mathcal{P}(V^*)$ gives all possible results of applying a multiset of rules to a string;
- $\mathsf{post} : V^* \to V^*$ is the post-processing function, applied to the string after the rules have been applied.

The functions mctx, mode, apply, and post must be computable.

In what follows, to make notation more compact, we will also use the symbol post to refer to the natural extension of post as defined above to sets of strings, i.e., the function with the type $\mathcal{P}(V^*) \to \mathcal{P}(V^*)$ assigning to a set $B \subseteq V^*$ the set $\{\mathsf{post}(w) \mid w \in B\}$.

As in the original reaction systems, the dynamics of RSS are defined by interactive processes. A context is any string $\gamma_i \in V^*$, and a context sequence of length $n+1$ is a sequence of such strings: $\gamma = (\gamma_i)_{0 \le i \le n}$. The following definition lifts the original notion of an interactive process to reaction systems with strings (cf. Sect. 2.1). We keep the notations largely similar.

[2] The first argument of mctx is the context and the second is the state. This function need not be commutative, nor deterministic.

Definition 3 Given a reaction system with strings $rss = (G, \mathsf{mctx}, \mathsf{mode}, \mathsf{apply}, \mathsf{post})$ and a context sequence $\gamma = (\gamma_i)_{0 \le i \le n}$, *an interactive process generated by* γ is the pair $\pi_\gamma = (\delta_\gamma, \tau_\gamma)$, where:

- $\delta_\gamma = (\delta_i)_{0 \le i \le n}$ is the result sequence, with $\delta_0 = \lambda$, $\delta_i \in \mathsf{post}(\mathsf{apply}(\rho, w_{i-1}))$, and $\rho \in \mathsf{mode}(w_{i-1})$;
- $\tau_\gamma = (w_i)_{0 \le i \le n}$, is the state sequence: $w_i \in \mathsf{mctx}(\gamma_i, \delta_i)$.

This definition incorporates three levels of non-determinism. First of all, the state w_i is non-deterministically picked among all possible ways to merge the context γ_i into δ_i given by $\mathsf{mctx}(\gamma_i, \delta_i)$. Secondly, ρ is a multiset of rules non-deterministically picked among the applicable multisets of rules given by $\mathsf{mode}(w_i)$. Finally, δ_i is non-deterministically picked among the possible results of applying ρ to w_{i-1}, post-processed by post. The immediate consequence of these levels of non-determinism is that a single context sequence may yield multiple different interactive processes.

Remark 2 Definition 3 is mutually recursive: δ_i relies on w_{i-1}, while w_{i-1} relies on δ_{i-1}. This is similar to the original definition in Sect. 2.1, in which D_i is essentially defined to be $\mathsf{res}_{\mathcal{A}}(W_{i-1})$. Classic definitions of interactive processes in reaction systems do not feature explicit mutual recursion because the procedure of constructing the state W_i is a simple union $D_i \cup C_i$. On the other hand, in reaction systems with strings $\mathsf{mctx}(\gamma_i, \delta_i)$ may yield multiple different states, which means that the definition of an interactive process must feature explicit mutual recursion.

The following example shows that Definition 3 is a strict generalization of reaction systems as defined in Sect. 2.1.

Example 4 Consider a finite alphabet V with an arbitrary total order on the symbols, e.g., $V = \{x_1, \ldots, x_m\}$, and the embedding $e : \mathcal{P}(V) \to V^*$ associating to every subset $V' \subseteq V$ the string containing exactly once the symbols in V', in order: $e(\{x_{i_1}, \ldots, x_{i_k}\}) = x_{i_1} \ldots x_{i_k}$. Also consider the dual projection $\mathsf{supp} : V^* \to \mathcal{P}(V)$ associating to every string the set of symbols it contains: $\mathsf{supp}(w) = \{x \mid x \text{ appears in } w\}$. Finally, define the function $\mathsf{flat} : V^* \to V^*$, $\mathsf{flat}(w) = e(\mathsf{supp}(w))$, which essentially removes the duplicates from w and reorders the symbols according to the total order on V.

Take a reaction system $\mathcal{A} = (V, R)$, and construct the following set of random-context string rewriting rules:

$$R' = \{\, r_a : (\lambda \to e(P_a), R_a, I_a) \mid a : (R_a, I_a, P_a) \in R \,\}.$$

Furthermore, define the following functions:

$$\begin{aligned}
\mathsf{mctx}(\gamma', w) &= \{\gamma' \cdot w\},\\
\mathsf{mode}(w) &= \{\rho_w\},\\
\rho_w(r_a) &= \begin{cases} 1, & \text{if } r_a \text{ is applicable to } w,\\ 0, & \text{otherwise,}\end{cases}\\
\mathsf{apply}(\rho, w) &= \left\{\textstyle\prod_{r_a \in \rho} e(P_a)\right\},\\
\mathsf{post}(w) &= \mathsf{flat}(w).
\end{aligned}$$

The function mctx simply concatenates the context γ' with the current state w, $\mathsf{mode}(w)$ constructs the multiset ρ_w in which all rules applicable to w appear with multiplicity 1, apply concatenates the product strings $e(P_a)$ of all reactions in ρ in the order given by an arbitrary total order on the reactions in R, and finally post removes all duplicates from the string. Remark that all these function are deterministic.

Construct the reaction system with strings $rss = (G, \mathsf{mctx}, \mathsf{mode}, \mathsf{apply}, \mathsf{post})$ with $G = (V, R')$, consider a context sequence $\gamma = (\gamma_i)_{0 \le i \le n}$ and the corresponding interactive process $\pi_\gamma = \big((\delta_i)_{0 \le i \le n}, (w_i)_{0 \le i \le n}\big)$. Then the pair $\big((\mathsf{supp}(\delta_i))_{0 \le i \le n}, (\mathsf{supp}(w_i))_{0 \le i \le n}\big)$ is an interactive process of the original reaction system $\mathcal{A}$, induced by the context sequence $(\mathsf{supp}(\gamma_i))_{0 \le i \le n}$. Indeed, the functions above essentially implement in strings the original semantics of interactive processes as defined in Sect. 2.1; see Theorem 2 for a formal argument.

The following example gives a concrete instantiation of the construction above.

Example 5 Consider the alphabet $V = \{x, y\}$ and the reactions $R = \{a : (\{x\}, \{y\}, \{y\}), b : (\{x\}, \emptyset, \{x, y\})\}$. Construct the following set of random-context string rewriting rules:

$$R' = \{\, r_a : (\lambda \to y, \{x\}, \{y\}),\ r_b : (\lambda \to xy, \{x\}, \emptyset) \,\},$$

as well as the reaction system with strings $rss = (G, \mathsf{mctx}, \mathsf{mode}, \mathsf{apply}, \mathsf{post})$ with $G = (V, R')$, and the functions mctx, mode, apply, and post defined as in Example 4. Consider the context sequence $\gamma = (x, x, \lambda)$. The following is the corresponding interactive process $\pi_\gamma = \big((\delta_i)_{0 \le i \le 2}, (w_i)_{0 \le i \le 2}\big)$:

i	0	1	2
γ_i	x	x	λ
δ_i	λ	xy	xy
w_i	x	xxy	xy

Both rules r_a and r_b can be applied at the first step, since w_0 contains x and no y, and even if w_0 contains only one copy of x, because r_a and r_b do not compete for this copy.

Applying both rules results in the string $\mathsf{apply}(r_a r_b, x) = yxy$, yielding $\delta_1 = xy = \mathsf{flat}(yxy)$, from which mctx is used to obtain $w_1 = xxy$ by prepending the context $\gamma_1 = x$. Only rule r_b can be applied now, yielding $\delta_2 = xy = w_2$.

Reaction systems with strings are also a direct member of the family of string rewriting models of computing. The following example stresses this strong relationship.

Example 6 Consider the string transformer $G = (V, R)$ (with semi-conditional rules) and define the following functions:

$$\begin{aligned}\mathsf{mctx}(\gamma', w) &= \{\gamma' \cdot w\},\\ \mathsf{mode}(w) &= \{r \mid r \in R,\ r \text{ is applicable to } w\},\\ \mathsf{apply}(r, w) &= \{v \mid w \overset{r}{\Rightarrow} v\},\\ \mathsf{post}(w) &= w.\end{aligned}$$

Thus, mctx appends the context to the current string as in Example 4, $\mathsf{mode}(w)$ returns the set of all singleton multisets each containing one rule applicable to w, $\mathsf{apply}(w)$ constructs the set of strings derivable from w by the rule r, and post does no post-processing of the string.

Construct now the reaction system with strings $rss = (G, \mathsf{mctx}, \mathsf{mode}, \mathsf{apply}, \mathsf{post})$. Supposing that $S \in V$, each interactive process yielded by the context sequence $\gamma = (\gamma_i)_{0 \le i \le n}$ with $\gamma_0 = S$ and $\gamma_i = \lambda$ for $1 \le i \le n$ will correspond to a derivation of length n of the string rewriting grammar (V, V, S, R).

As before, the following example gives a concrete instantiation of this construction.

Example 7 Take the alphabet $V = \{S, a, b\}$ and the set of semi-conditional rules $R = \{r_1 : (S \to aSb, \emptyset, \emptyset),\ r_2 : (S \to \lambda, \emptyset, \emptyset)\}$ with empty permitting and forbidden contexts. Consider the string transformer $G = (V, R)$ and the reaction system with strings $rss = (G, \mathsf{mctx}, \mathsf{mode}, \mathsf{apply}, \mathsf{post})$, with the functions mctx, mode, apply, and post defined as in Example 6. The following is one of the interactive processes in rss yielded by the context sequence $\gamma = (S, \lambda, \lambda, \lambda)$:

i	0	1	2	3
γ_i	S	λ	λ	λ
δ_i	λ	aSb	$aaSbb$	$aabb$
w_i	S	aSb	$aaSbb$	$aabb$

In the first step both rules $r_1$and r_2 are applicable, meaning that $\mathsf{mode}(S) = \{r_1, r_2\}$, $\mathsf{apply}(r_1, S) = \{aSb\}$, and $\mathsf{apply}(r_2, S) = \{\lambda\}$. In this interactive process, r_1 is applied, yielding $\delta_1 = aSb = w_1$. The same rule is applied in the second step, leading to $\delta_2 = aaSbb = w_2$. Both rules are again applicable in the third step, but this time r_2 is applied, yielding $\delta_3 = aabb = w_3$. We insist on the fact that this is only one of the interactive processes yielded by γ. The following table presents the result sequences of all such interactive processes, together with the applied rules:

i	0	1	2	3	Rules applied
γ_i	S	λ	λ	λ	
$\delta_i^{(1)}$	λ	λ	λ	λ	r_2
$\delta_i^{(2)}$	λ	aSb	ab	ab	$r_1 r_2$
$\delta_i^{(3)}$	λ	aSb	$aaSbb$	$aabb$	$r_1 r_1 r_2$
$\delta_i^{(4)}$	λ	aSb	$aaSbb$	$aaaSbbb$	$r_1 r_1 r_1$

Remark 3 The definition of apply in Example 6 is incomplete: indeed, the function is only defined for singleton multisets, containing exactly one copy of a rule. To formally comply with Definition 2, we would need to instead provide a description similar to the following one:

$$\mathsf{apply}(\rho, w) = \begin{cases}\{v \mid w \overset{r}{\Rightarrow} v\}, & \rho = r \in R,\\ \emptyset, & \text{otherwise.}\end{cases} \tag{2}$$

Note however that according to the dynamics of reaction systems with strings (Definition 3), apply is only used with multisets of rules produced by mode, and in the example above mode only produces singleton multisets. Therefore, the value apply is assigned in the second line of equation (2) is of no importance.

Given this observation, in what follows we will systematically define apply only for the multisets of rules produced by mode, and will implicitly suppose that apply is also defined to produce e.g. $\emptyset$ for all other multisets of rules.

5 Relationship between reaction systems and RSS

In order to compare the behavior of different variants of reaction systems with strings to that of conventional reaction systems, we introduce in this section two compatibility conditions.

Definition 4 A reaction system with strings rss is *set-compliant* if there exists a reaction system $\mathcal{A}_{rss}$ over the same alphabet such that for any context sequence $\gamma = (\gamma_i)_{1 \le i \le n}$ for rss and a corresponding interactive process $\pi_\gamma = \big((\delta_i)_{0 \le i \le n}, (w_i)_{0 \le i \le n}\big)$ generated by γ, the pair $\big((\mathsf{supp}(\delta_i))_{0 \le i \le n}, (\mathsf{supp}(w_i))_{0 \le i \le n}\big)$ is the interactive process of the reaction system $\mathcal{A}_{rss}$ generated by the context sequence $(\mathsf{supp}(\gamma_i))_{0 \le i \le n}$.

Informally, all interactive processes of a set-compliant RSS are interactive processes of a conventional reaction system, modulo the projection supp. Note that this definition does not require set-compliant reaction systems with strings

to be deterministic, but it does restrict all interactive processes induced by the same context sequence to have the same projection by supp. Furthermore, this definition imposes the same equality restriction on *all* interactive processes induced by *all* context sequences with the same projection by supp.

Definition 5 A reaction system with strings rss is *set-compatible* if there exists a reaction system $\mathcal{A}_{rss}$ over the same alphabet V such that for any context sequence $\gamma = (C_i)_{0\leq i\leq n}$ of $\mathcal{A}_{rss}$ there exists a context sequence $\gamma' = (\gamma'_i)_{0\leq i\leq n}$ such that $\mathsf{supp}(\gamma'_i) = C_i$, and which generates an interactive process $\big((\delta_i)_{0\leq i\leq n}, (w_i)_{0\leq i\leq n}\big)$ of rss with the property that $\big((\mathsf{supp}(\delta_i))_{0\leq i\leq n}, (\mathsf{supp}(w_i))_{0\leq i\leq n}\big)$ is the interactive process of $\mathcal{A}_{rss}$ generated by γ.

Informally, a reaction system with strings rss is set-compatible if all interactive processes of a conventional reaction system $\mathcal{A}_{rss}$ can be mapped onto a subset of the interactive processes of rss.

Example 8 Consider the reaction system with strings $rss = (G, \mathsf{mctx}, \mathsf{mode}, \mathsf{apply}, \mathsf{post})$ with $G = (V, R)$, $V = \{a, b, c\}$, $R = \{r_b : a \to b, r_c : aa \to c\}$, the functions mode and apply defined as follows:

$$\mathsf{mode}(w) = \{\rho_w\},$$
$$\rho_w = \{r_c^k r_b^t \mid k \text{ is the biggest natural s.t. } |w|_a = 2k + t\}\,[3],$$
$$\mathsf{apply}(r_c^k r_b^t, w) = \{c^k b^t\},$$

and the other functions defined as in Example 4: $\mathsf{mctx}(\gamma', w) = \{\gamma' \cdot w\}$ and $\mathsf{post}(w) = \mathsf{flat}(w)$. Intuitively, the function $\mathsf{mode}(w)$ picks the rule r_c as many times as there are pairs of a in w, and if an odd a is left, it adds r_b to ρ_w. apply in its turn simply discards the structure of w and concatenates the right-hand sides of r_c and r_b in the corresponding number of copies.

Consider now two context sequences γ_1 and γ_2 of rss, as well as the state sequences τ_1 and τ_2 of the corresponding interactive processes, together with the projections by supp:

$$\gamma_1 = (a, \lambda), \qquad \gamma_2 = (aa, \lambda),$$
$$\tau_1 = (a, b), \qquad \tau_2 = (aa, c),$$
$$\mathsf{supp}(\gamma_1) = (a, \lambda) = \mathsf{supp}(\gamma_2) = (a, \lambda),$$
$$\mathsf{supp}(\tau_1) = (a, b) \neq \mathsf{supp}(\tau_2) = (a, c).$$

The projections $\mathsf{supp}(\gamma_1)$ and $\mathsf{supp}(\gamma_2)$ are the same, but the projections of the state sequences $\mathsf{supp}(\tau_1)$ and $\mathsf{supp}(\tau_2)$ are different. Since conventional reaction systems are deterministic, there cannot exist a reaction system featuring both state sequences for the same context sequence $\mathsf{supp}(\tau_1) = \mathsf{supp}(\tau_2)$. The reaction system with strings rss is therefore *not set-compliant*.

On the other hand, take the embedding $e : \mathcal{P}(V) \to V^*$ defined in Example 4, consider the reaction system $\mathcal{A}_{rss}$ over V with the only reaction $(\{a\}, \emptyset, \{b\})$, and take an arbitrary context sequence $\gamma = (C_i)_{1\leq i\leq n}$. It follows from the definition of e that all strings $e(C_i)$ will contain at most one copy of a, meaning that in any interactive process of rss generated by the sequence $(e(C_i))_{1\leq i\leq n}$ only the rule $a \to b$ will ever be applied. It follows from the definitions of the functions mctx, mode, apply, and post that embedding a result sequence $(D_i)_{1\leq i\leq n}$ of $\mathcal{A}_{rss}$ into V^* will yield the corresponding result sequence $(e(D_i))_{i\leq i\leq n}$ of rss, meaning that the reaction system with strings rss *is set-compatible*.

The previous example shows that set-compliance is a stronger requirement, since it requires essentially that *all* interactive processes of a reaction system with strings should project by supp onto interactive processes of a single conventional reaction system. On the other hand, set-compatibility intuitively requires that only a part of all possible interactive processes should be representable by the interactive processes of a conventional reaction system. This intuitive observation directly leads to the following statement.

Proposition 1 *A reaction system with strings rss which is set-compliant is set-compatible.*

Proof According to Definition 4, that rss is set-compliant means that there exists a conventional reaction system $\mathcal{A}_{rss}$ such that all interactive processes of rss map onto interactive processes of $\mathcal{A}_{rss}$ by supp. By directly checking the conditions of Definition 5, one can verify that the same reaction system $\mathcal{A}_{rss}$ satisfies them, which proves the statement of the proposition. □

Example 8 shows that the converse statement is not true—not every set-compatible reaction system is set-compliant.

6 The spectrum between reaction systems and string rewriting

Examples 4 and 6 show the two ends of the spectrum between reaction systems and string rewriting. In this section we will provide further constructs situated along this spectrum and formalizing the ideas from Sect. 3. For brevity, we will refer to these formalizations using the terms "type k", in which k can be seen as an informal measure of the strength of the restrictions added to the general model of reaction systems with strings—similarly to the Chomsky hierarchy.

[3] In other words, k is the quotient and t the remainder of dividing $|w|_a$ by 2.

6.1 RSS of Type 0

The following definition further formalizes the construction from Example 4, which also corresponds to the informal discussion in paragraphs "Reactions as insertions" and "Direct parallelism" from Sect. 3.

Definition 6 A *reaction system with strings of type 0* is a reaction system with strings as in Definition 2 $rss = (G, \mathsf{mctx}, \mathsf{mode}, \mathsf{apply}, \mathsf{post})$, which additionally satisfies the following conditions:

1. the rules in the string transformer $G = (V, R)$ are random-context rules of the form $r : (\lambda \to \alpha_r, P_r, Q_r)$, with $\alpha_r \in V^*$, $P, Q \subseteq V$;
2. $\mathsf{mctx}(\gamma', w) = \{\gamma' \cdot w\}$ [3]
3. $\mathsf{mode}(w) = \{\rho_w\}$, where $\rho_w(r) = \begin{cases} 1, & \text{if } r \text{ is applicable to } w, \\ 0, & \text{otherwise;} \end{cases}$
4. $\mathsf{apply}(\rho, w) = \left\{\prod_{r \in \rho} \alpha_r\right\}$, where the concatenation happens according to an arbitrary but fixed total order on R,
5. $\mathsf{post}(w) = \mathsf{flat}(w)$.

The following statements formalize the remark at the end of Example 4 about the relation between the interactive processes of reaction systems with strings of type 0 and conventional reaction systems.

Theorem 2 *All reaction systems with strings of type 0 are set-compliant.*

Proof Consider a reaction system with strings of type 0 $rss = (G, \mathsf{mctx}, \mathsf{mode}, \mathsf{apply}, \mathsf{post})$ with $G = (V, R)$, and pick any context sequence $\gamma = (\gamma_i)_{1 \le i \le n}$. Since rss is of type 0, all 4 functions mctx, mode, apply, and post are deterministic, meaning that γ generates exactly one interactive process $((\delta_i)_{1 \le i \le n}, (w_i)_{1 \le i \le n})$.

Define the function $\mathsf{pj}\big(r : (\lambda \to \alpha, P_r, I_r)\big) = (\mathsf{supp}(\alpha_r), I_r, P_r)$ converting a random-context rule from R into a reaction, and construct the reaction system $\mathcal{A} = (V, R')$ with $R' = \{\mathsf{pj}(r) \mid r \in R\}$. Pick a state $w_i \in V^*$ and remark that, if $\mathsf{mode}(w_i) = \{\rho_{w_i}\}$, then $\{\mathsf{pj}(r) \mid r \in \rho_{w_i}\}$ is exactly the set of reactions enabled in $\mathcal{A}$ in the set $\mathsf{supp}(w_i)$. Therefore, the only string $w_{i+1} \in \mathsf{post}(\mathsf{apply}(\rho_{w_i}, w_i))$ will have the property $\mathsf{supp}(w_{i+1}) = \mathsf{res}_{\mathcal{A}}(\mathsf{supp}(w_i))$. This, together with the commutativity $\mathsf{supp}(\mathsf{mctx}(\gamma_i, w_i)) = \mathsf{supp}(\gamma_i) \cup \mathsf{supp}(w_i)$ implies the set-compliance of rss. □

By Proposition 1, we also directly obtain the following result.

Corollary 3 *All reaction systems with strings of type 0 are set-compatible.*

Remark 4 Defining post to be flat in Definition 6 does not have a major impact on the expressiveness of the behavior of RSS of type 0: the applicability conditions of the rules do not depend on the number of copies of symbols, and mode will anyway only include one copy of every applicable rule into the multiset ρ_w. In particular, if Definition 6 defined $\mathsf{post}(w) = w$, Theorem 2 would also hold. To formally distinguish between these two slight variations, we use the term *reaction systems with strings of type* $0'$ (read as "type zero-prime") to refer to the variant in which $\mathsf{post}(w) = w$.

6.2 RSS of Type 1

The previous results show that reaction systems of type 0 have essentially the same behavior as conventional reaction systems. We now formalize the discussion from paragraph "Preserve multiplicities" in Sect. 3. We start by defining the multiset support function $\mathsf{supp}^* : V^* \to V^\circ$ over an alphabet V in the following way: $\mathsf{supp}^*(w)(a) = |w|_a$, i.e., supp^* assigns to a string w the multiset containing the same symbols in exactly the same multplicities.

Definition 7 A *reaction system with strings of type 1* or *with multiplicity preservation* is a reaction system with strings as in Definition 2, $rss = (G, \mathsf{mctx}, \mathsf{mode}, \mathsf{apply}, \mathsf{post}, \mathsf{mul})$, with $G = (V, R)$, the additional computable function $\mathsf{mul} : V^\circ \times R \to \mathbb{N}$, and satisfying the following conditions:

1. the rules in the string transformer $G = (V, R)$ are random-context rules of the form $r : (\lambda \to \alpha_r, P_r, Q_r)$, with $\alpha_r \in V^*$, $P, Q \subseteq V$;
2. $\mathsf{mctx}(\gamma', w) = \{\gamma' \cdot w\}$;
3. $\mathsf{mode}(w) = \{\rho_w\}$, where $\rho_w(r) = \begin{cases} \mathsf{mul}(\mathsf{supp}^*(w), r), & \text{if } r \text{ is applicable to } w, \\ 0, & \text{otherwise;} \end{cases}$
4. $\mathsf{apply}(\rho, w) = \left\{\prod_{r \in \rho} (\alpha_r)^{\rho(r)}\right\}$, where the concatenation happens according to an arbitrary but fixed total order on R, and a right hand side of a rule r occurs as many times as r appears in ρ;
5. $\mathsf{post}(w) = w$.

The central difference between RSS of type 0 in Definition 6 and type 1 in Definition 7 is in function mode: it is still deterministic in RSS of type 1, but this time the number of times a rule appears in the single multiset of rules is determined by the function mul, which in its turn derives this number from the multiplicities of symbols in w, and may therefore do arbitrary computable connections with the components of the rule r. Accordingly, apply is slightly modified

[3] More generally, mctx; may be the shuffle of γ' and w, $\mathsf{mctx}(\gamma', w) = \gamma' \sqcup\!\sqcup w$.

to account for multiplicities of rules in ρ. Finally, post just keeps the only string resulting from apply as it is.

We will now show a connection between RSS of type 1 with a subclass of register machines, which are a model of computing equipped with a finite number of numerical registers, each of which stores a natural number, and with a program consisting of increment and decrement instructions. Each instruction is labelled with a state, and indicates which instruction or instructions are to be executed next. The program also has a special halting instruction which indicates that the execution is finished and the result can be retrieved. We refer to Korec (1996) for a detailed technical presentation of register machines and their computing power, and to Păun et al. (2010) for a rich overview of the use of register machines in evaluating the computing power of other models.

Example 9 Consider register machines only equipped with increment instructions of the form $(p, \mathrm{ADD}(r), q)$, meaning that in state p the machine increments register r and goes to state q. Such register machines are deterministic, and each one produces exactly one output: the vector of numbers stored in its registers when the halting configuration is reached.

Consider such a deterministic register machine M and construct the RSS of type 1 rss_M with the alphabet $V = Q \cup A$, where Q is the set of states of M and $A = \{a_1, \ldots, a_m\}$ has a symbol for every register of M. The rules of rss_M are the following:

$$\begin{aligned} &r_p : (\lambda \to q a_r, \{p\}, \emptyset), && \text{for every instruction } (p, \mathrm{ADD}(r), q),\\ &r_r : (\lambda \to a_r, \{a_r\}, \emptyset), && \text{for every } a_r \in A. \end{aligned}$$

Define the function mul in the following way:

$$\begin{aligned} &\mathsf{mul}(\bar{w}, r_p) = \bar{w}(p), && p \in Q,\\ &\mathsf{mul}(\bar{w}, r_r) = \bar{w}(a_r), && a_r \in A. \end{aligned}$$

Intuitively, the rules r_p will produce the new state symbol corresponding to instruction p in as many copies as p is present in the current string, and the rules r_r will simply maintain the register symbols a_r in the same number of copies.

Take now a context sequence $\gamma = (q_0, \lambda, \ldots, \lambda)$ of length $n+1$, where q_0 is the starting state of M. Then the interactive process generated by γ will essentially simulate the computation of M of length n. Remark that if M reaches the halting instruction p_h in less than n steps, then rss_M will simply lose the state symbol p_h and maintain the multiplicities of a_r, because the halting instruction will have no rule associated to it in R. Finally, note that the simulation will only be valid if the context sequence has the form specified above. Introducing extra symbols from Q or A via the contexts may obviously break the simulation relationship to M.

The previous example shows that RSS of type 1 can count, and the size of their configuration is unbounded, even as they respect the threshold principle and the non-permanency principle. Nevertheless, this result is somewhat underwhelming, since we picked a very restrained class of register machines. The following example shows how non-determinism can be injected via the context sequence, considerably enriching the possible behaviors of an RSS of type 1.

Example 10 Consider register machines equipped with non-deterministic increment instructions of the form $(p, \mathrm{ADD}(r), q, s)$, meaning that in state p the machine increments register r and non-deterministically chooses between going to state q or to state s. For such instructions we will use the notation $\mathsf{next}(p) = \{q, s\}$. Register machines of this form generate all linear vector languages [see Păun et al. (2010)].

Consider such a register machine M and construct the RSS of type 1 rss_M with the alphabet $V = Q \cup \bar{Q} \cup A$, where Q is the set of states of M, $\bar{Q} = \{\bar{q} \mid q \in Q\}$, and $A = \{a_1, \ldots, a_m\}$ has a symbol for every register of M. The rules of rss_M are the following:

$$\begin{aligned} &r_p : (\lambda \to q a_r, \{p, \bar{q}\}, \emptyset), && p \in Q, q \in \mathsf{next}(p),\\ &\bar{r}_p : (\lambda \to p, \{p\}, \{\bar{q}\}), && p \in Q, q \in \mathsf{next}(p),\\ &r_r : (\lambda \to a_r, \{a_r\}, \emptyset), && \text{for every } a_r \in A. \end{aligned}$$

We define the function mul as follows:

$$\begin{aligned} &\mathsf{mul}(\bar{w}, r_p) = \bar{w}(p), && p \in Q,\\ &\mathsf{mul}(\bar{w}, \bar{r}_p) = \bar{w}(p), && p \in Q,\\ &\mathsf{mul}(\bar{w}, r_r) = \bar{w}(a_r), && a_r \in A. \end{aligned}$$

Intuitively, if the current configuration contains both p and $\bar{q}$ such that $q \in \mathsf{next}(p)$, then the corresponding rule r_p will simulate the transition to q, adding a copy of a_r. Otherwise, if the configuration does not contain such a $\bar{q}$, then r_p will not be applicable, and $\bar{r}_p$ will instead maintain the current state symbol p. Note that if both p and $\bar{q}$ are present in the current configuration, only the multiplicity of p affects the multiplicity assigned to the rules r_p and $\bar{r}_p$.

Consider now the following family of context sequences

$$\Gamma_n = \{(q_0\gamma_0, \gamma_1, \ldots, \gamma_n) \mid \gamma_i \in \bar{Q}, 0 \le i \le n\}.$$

In other words, Γ_n contains all context sequences in which each context is exactly one symbol from $\bar{Q}$, except for the very first one which also contains the initial state q_0 of M. When run with such a context sequence $\gamma \in \Gamma_n$, rss_M will apply a rule r_p simulating an increment whenever the context sequence injects the correct $\bar{q}$—i.e., such a symbol $\bar{q}$ that $q \in \mathsf{next}(p)$—or a rule $\bar{r}_p$ whenever the wrong $\bar{q}$ is injected. Therefore, the set of interactive processes generated by the context sequences in Γ_{n+1} will simulate all computations of M of length less or equal to n. This means that the contexts from the set $\Gamma_* = \bigcup_{n\in\mathbb{N}} \Gamma_n$ will generate interactive processes corresponding to all computations of M.

Observe that rss_M contains no rules for the halting instruction h of M, because $\mathsf{next}(h) = \emptyset$. This means that if a simulation reaches h at step $i < n$, i.e., before the context sequence is exhausted, then all the subsequent results $(\delta_j)_{i<j\leq n}$ will be equal to the same string over A in which the multplicities of the symbols a_r will correspond to the values of the corresponding registers r when M reached the halting instruction.

The two examples above show that RSS of type 1 can explore unbounded state spaces by counting register symbols a_r. This is ensured by the function mul which allows for carrying over symbol multiplicities from the current string-state to the numbers of times rules are applied. In general, this comes with losing the strong connection to conventional reaction systems featured by RSS of type 0.

Example 11 Consider the singleton alphabet $V = \{a\}$ and the RSS of type 1 rss over it with the only rule $r : (\lambda \to a, \{a\}, \emptyset)$, with the function mul defined as follows:

$$\mathsf{mul}(\bar{w}, r) = \begin{cases} 1, & \bar{w}(a) = 1, \\ 0, & \bar{w}(a) > 1. \end{cases}$$

Consider now two context sequences $\gamma_1 = (a, \lambda)$ and $\gamma_2 = (aa, \lambda)$. The state sequences of the interactive processes of rss generated by γ_1 and γ_2 will be respectively $\tau_1 = (a, a)$ and $\tau_2 = (aa, \lambda)$, because in the case of γ_2 mul will deny the application of r, even though it is applicable to the string aa. Note now that the projections by supp of γ_1 and γ_2 are the same: $(\{a\}, \emptyset)$, but the projections of the state sequences τ_1 and τ_2 are different, meaning that rss is *not* set-compliant.

This reaction system with strings rss is on the other hand *still* set-compatible. Indeed, consider the reaction system with no reactions at all $\mathcal{A} = (V, \emptyset)$. Then the result sequence of any interactive process of $\mathcal{A}$ entirely consists of empty sets, and the state sequence is equal to the context sequence. Furthermore, the elements of any context sequence are either $\emptyset$ or $\{a\}$. Take any such context sequence of $\mathcal{A}$ and construct a context sequence for rss by replacing all $\emptyset$ by λ and all $\{a\}$ by aa. According to the definition of mul, always injecting two copies of a will essentially disable the only rule r of rss, meaning that the corresponding result sequence will only contain empty strings, and that the state sequence will be equal to the context sequence. Therefore, $\mathcal{A}$ satisfies the conditions of Definition 5, implying that rss is set-compatible.

The salient feature of function mul used in the example above is that it can deny the application of an applicable rule. It is possible to restrict mul to comply with the mode and not disable rules in this way.

Definition 8 The multiplicity function $\mathsf{mul} : V^\circ \times R \to \mathbb{N}$ of an RSS of type 1 is called *non-denying* if, for any $w \in V^*$ and for any $r \in R$, $\mathsf{mul}(\mathsf{supp}^*(w), r) > 0$ if r is applicable to w.

Thus, a non-denying multiplicity function can modulate the number of times a rule is applied, but not prevent it from being applied if it is applicable. This drastically changes the effect of the multiplicity function, as the following theorem shows.

Theorem 4 *All reaction systems with strings of type 1 with a non-denying multiplicity function* mul *are set-compliant.*

Proof It suffices to remark that a non-denying mul only affects the number of times an applicable rule is applied, i.e., the number of copies in which its right-hand side will appear in the result produced by apply. Modifying this number does not change the projection on sets by supp in any way. Similarly, post being the identity function in the definition of RSS of type 1 (Definition 7) does not change the projection of the result sequence by supp with respect to RSS of type 0, in which post is defined to be the function flat (Definition 6).[4] These observations allows us to directly apply the argument of Theorem 2 to prove the above statement. □

On the other hand, denying multiplicity functions allow for directly simulating computations of register machines with both increment and decrement instructions up to a certain length.

Example 12 Consider a deterministic register machine M with two kinds of instructions:

- $(p, ADD(r), q)$: in state p, increment register r and go to state q:
- $(p, SUB(r), q, z)$: in state p, if register r is non-zero, decrement it and go to state q, otherwise go to state z.

Such register machines are known to be computationally complete (Korec 1996; Cocke and Minsky 1964; Păun et al. 2010), meaning that for any computable function f of vectors of numbers there exists a register machine which computes f.

Construct now the RSS of type 1 rss_M with the alphabet $V = Q \cup A$, where, as in Example 9, Q is the set of states of M and $A = \{a_1, \ldots, a_n\}$ has a symbol for every register of M. The following list gives the rules of rss_M, as well as the multiplicities they are assigned in order to simulate the program of M:

- for every instruction $(p, \mathrm{ADD}(r), q)$:

$$r_p : (\lambda \to q a_r, \{p\}, \emptyset) \quad \text{and} \quad \mathsf{mul}(\bar{w}, r_p) = \bar{w}(p);$$

[4] See also Remark 4 for a discussion of post taken to be flat in Definition 6.

- for every instruction $(p, \mathrm{SUB}(r), q, z)$:

  $r_p^q : (\lambda \to q, \{p\}, \emptyset)$ and $r_p^z : (\lambda \to z, \{p\}, \emptyset)$,

  together with the multiplicities

  $$\mathsf{mul}(\bar{w}, r_p^q) = \begin{cases} 1, & \bar{w}(a_r) > 0, \\ 0, & \text{otherwise}, \end{cases} \quad \text{and} \quad \mathsf{mul}(\bar{w}, r_p^z) = \begin{cases} 1, & \bar{w}(a_r) = 0, \\ 0, & \text{otherwise}; \end{cases}$$

- for every symbol $a_r \in A$:

  $r_r : (\lambda \to a_r, \{a_r\}, \emptyset)$,

  and the following multiplicity assignment:

  $$\mathsf{mul}(\bar{w}, r_r) = \begin{cases} \bar{w}(a_r) - 1, & \exists p \in \mathsf{dec}(r) : \bar{w}(p) = 1, \text{ and } \bar{w}(a_r) > 0, \\ \bar{w}(a_r), & \text{otherwise}. \end{cases}$$

  where $\mathsf{dec}(r)$ is the set of states in which M attempts a decrement on register r.

The behaviour of rule r_p for increment instructions $(p, \mathrm{ADD}(r), q)$ is exactly the same as in Example 9: a copy of a_r is added once if p appears in the string.

In the case of a decrement instruction $(p, \mathrm{SUB}(r), q, z)$, the two rules r_p^q and r_p^z add the new state symbol, while the function mul checks the corresponding conditions on the multiplicity of a_r and decides which of the two rules to apply. More concretely, if the string still contains copies of a_r, rule r_p^q is applied; otherwise, rule r_p^z is applied. Note that mul will deny applications of applicable functions, mul is therefore a denying multiplicity function.

Finally, rule r_r, and particularly the multiplicity mul assigns to it, has the role of maintaining the current number of copies of a_r in the string, unless the current instruction symbol corresponds to a decrement instruction on register r, and there are still copies of a_r in the string.

Consider now the context sequence $\gamma = (\gamma_0, \lambda, \ldots, \lambda)$ of length $n + 1$ and $\gamma_0 = q_0 a_{r_1}^{k_1} \ldots a_{r_m}^{k_m}$, where q_0 is the initial state of M, and k_i is the initial value of register r_i. The interactive process generated by γ will exactly simulate the computation of M of length n on the corresponding input vector $(k_1, \ldots, k_m)$.

Remark 5 In Examples 9, 10, and 12, the function mul assigns to the rules r_p simulating ADD instructions the same multiplicity as that of the corresponding symbol p in the current state of the reaction system with strings, i.e., $\mathsf{mul}(\bar{w}, r_p) = \bar{w}(p)$. In Example 10, mul assigns the same multiplicity to rule $\bar{r}_p$, which is also part of simulating an ADD instruction: $\mathsf{mul}(\bar{w}, \bar{r}_p) = \bar{w}(p)$. The important part about these definitions of mul is that they *enable the correct rules* r_p (and $\bar{r}_p$) corresponding to the symbol p appearing in the current state, *not* the actual multiplicities assigned to these rules. In other words, in these examples mul could also be defined in the following way:

$$\mathsf{mul}(\bar{w}, r_p) = \begin{cases} 1, & \bar{w}(p) > 0, \\ 0, & \text{otherwise}, \end{cases}$$

similarly to how $\mathsf{mul}(\bar{w}, r_p^q)$ is defined in Example 12.

6.3 RSS of Type 2

As discussed in paragraph "Preserve positions" of Sect. 3, we can even go beyond preserving symbol multiplicities—which yields quite powerful devices as the previous section shows—and seek to also maintain a connection between the positions at which insertions occur, and the positions at which some of the reactants or inhibitors appear in the string.

In what follows, we will rely on the position function $\mathsf{pos} : V^* \times R \to \mathcal{P}(\mathbb{N})$ which, given a word w and a rule r, produces a set of "positions" at which the right-hand side of r should appear in the result. The exact meaning of the term "position" will be clarified in Definition 9, but we first define two utility functions to manipulate subsets of $\mathbb{N} \times V^*$, which will intuitively correspond to pairs of the form (position, word) in our constructions.

First, we introduce the function sort_1 which sorts a subset $B \subseteq \mathbb{N} \times V^*$ by the numerical value of its first element. We do not require the values of the first elements in B to be distinct, so sort_1 produces a set of ordered sequences of pairs (position, word)[5]. Thus, a sequence $(i_1, \alpha_1) \ldots (i_m, \alpha_m) \in \mathsf{sort}_1(B)$ has the property that $i_k \leq i_{k+1}$, for all $1 \leq k < m$. If, as an example, $B = \{(1, \alpha_1), (1, \alpha_2), (2, \alpha_3)\}$ for some strings $\alpha_1, \alpha_2, \alpha_3$, then

$$\mathsf{sort}_1(B) = \{\, (1, \alpha_1)(1, \alpha_2)(2, \alpha_3), \\ (1, \alpha_2)(1, \alpha_1)(2, \alpha_3) \,\}.$$

Second, we introduce the function cat_2 which concatenates the second elements of a sequence of pairs (position, word)[6]:

$$\mathsf{cat}_2\big((i_1, \alpha_1) \ldots (i_m, \alpha_m)\big) = \alpha_1 \ldots \alpha_m.$$

[5] Formally, the type of this function is $\mathsf{sort}_1 : \mathcal{P}(\mathbb{N} \times V^*) \to \mathcal{P}\big((\mathbb{N} \times V^*)^*\big)$.

[6] Formally, the type of this function is $\mathsf{cat}_2 : (\mathbb{N} \times V^*)^* \to V^*$. Note that cat_2 discards the first elements of the pairs in the argument sequence, and just concatenates the words in the order in which the appear.

We will also use the symbol cat_2 to refer to a natural extension of this function to *sets of sequences* of pairs from $\mathbb{N} \times V^*$.

We now define RSS of type 2 using these utility functions.

Definition 9 A *reaction system with strings of type 2* or *with position preservation* is a reaction system with strings as in Definition 2, $rss = (G, \mathsf{mctx}, \mathsf{mode}, \mathsf{apply}, \mathsf{post}, \mathsf{pos})$, with $G = (V, R)$, the additional computable function $\mathsf{pos} : V^* \times R \to \mathcal{P}(\mathbb{N})$, and satisfying the following conditions:

1. the rules in the string transformer $G = (V, R)$ are random-context rules of the form $r : (\lambda \to \alpha_r, P_r, Q_r)$, with $\alpha_r \in V^*$, $P, Q \subseteq V$;
2. $\mathsf{mctx}(\gamma', w) = \{\gamma' \cdot w\}$;
3. $\mathsf{mode}(w) = \{\rho_w\}$, where $\rho_w = \begin{cases} 1, & \text{if } r \text{ is applicable to } w, \\ 0, & \text{otherwise;} \end{cases}$
4. The function apply is defined in the following way:

$$\mathsf{apply}(\rho, w) = \mathsf{cat}_2\left(\mathsf{sort}_1\left(\bigcup_{r \in \rho}(\mathsf{pos}(w, r) \times \{\alpha_r\})\right)\right).$$

5. $\mathsf{post}(w) = w$.

In RSS of type 2, the function mode is in fact the same as in RSS of type 0: it produces one single multiset of rules, in which a rule may have multiplicity 0 or 1, meaning that the only element of $\mathsf{mode}(w)$ is essentially just a set. The function apply on the other hand employs the function pos in the most intimate way. The operation of apply can alternatively be described by the following algorithm:

1. For every rule r in multiset ρ (which is essentially a set), use pos to construct the set of positions at which its right-hand side α_r should appear in the result.
2. Sort the resulting set of pairs (position, word) by position, potentially producing multiple sorted sequences.
3. For each obtained sequence, concatenate the corresponding subwords.

Thus, the exact meaning of the term "position" in the above is "relative position with respect to the other subwords in the resulting string".

We will now show that having a fine control over the order in which the right-hand sides of the rules are concatenated allows for directly simulating Turing machines—the golden standard of computationally complete models of computing. Multiple equivalent definitions of the model exist—e.g. Freund and Staiger (2019). Here we quickly recall the essential parts. A Turing machine is an automaton operating on a finite unbounded tape, whose cells contain elements of the tape alphabet Σ, and which is bounded by the left end marker $\triangleright$ and the right end marker $\triangleleft$. The Turing machine has a head with a state coming from the finite state alphabet Q. The head points on one of the tape cells. Depending on the state, it may rewrite the symbol to another one, and move one cell left or right. Without losing generality, we may consider that the machine always moves its head, i.e., that its program contains no instructions that rewrite the tape symbol, but keep the head in place. In order to obtain a uniform string representation of a configuration of the machine, we will write the current state symbol *to the left* of the tape symbol to which the head is currently pointing. For example, here is how we write a configuration in which the machine is in state p and its head points to the i-th tape cell:

$$\triangleright a_1 \dots a_{i-1} p a_i \dots a_m \triangleleft .$$

This notation allows us to write every instruction in the program of a (deterministic) Turing machine in one of the following three forms:

- $pa \to bq$, $p, q \in Q$, $a, b \in \Sigma$: in state p and reading symbol a on the tape, replace it by b (b may be the same as a), switch to state q, and move one cell right;
- $cpa \to qcb$, $p, q \in Q$, $a, b, c \in \Sigma$: in state p and reading symbol a on the tape, replace it by b (b may be the same as a), switch to state q, and move one cell left;
- $p\triangleleft \to aq\triangleleft$, $p, q \in Q$, $a \in \Sigma$: in state p and with the head pointing to the right end of the tape, add a new cell to the tape, write a onto it[7], switch to state q, and move one cell right.

Before showing how RSS of type 2 can simulate the computations of a Turing machine, we introduce the helper function $\mathsf{find} : V \times V^* \to \mathcal{P}(\mathbb{N})$: given a symbol $a \in V$ and a string $w \in V^*$, $\mathsf{find}(a, w)$ returns the set of all positions at which a appears in w. For example, $\mathsf{find}(l, hello) = \{3, 4\}$, and $\mathsf{find}(a, hello) = \emptyset$.

Example 13 Consider a Turing machine M defined as above, with the tape alphabet Σ and the state alphabet Q, and construct an RSS of type 2 rss_M with the alphabet $V = \Sigma \cup Q \cup \{\triangleright, \triangleleft\}$. The set of rules of rss_M will only contain rules of one form:

$$r_x : (\lambda \to x, \{x\}, \emptyset),$$

where $x \in V$ can be any symbol. We will use the notation r_a for rules r_x in which $x \in \Sigma$, and r_p in the cases in which

[7] Often this new cell contains a special blank symbol $\flat \in \Sigma$, designating the fact the cell does not yet contain "useful" information. In this paper, we do not rely on the distinction between $\flat$ and $\Sigma \setminus \{\flat\}$, so we do not emphasize it.

$x \in Q$, and we will write $r_{\triangleright}$ and $r_{\triangleleft}$ for the rules sustaining the left and right end markers respectively.

We will now describe how the values of the function pos are computed depending on the shape of the fixed current configuration w, the state symbol p appearing in w, and the rule r_a for which the set of positions needs to be computed. A valid configuration w contains only one state symbol p. In what follows, we denote by i_p the position at which p appears in the string w, i.e., $\{i_p\} = \mathsf{find}(p, w)$.

- when p corresponds to an instruction $pa \to bq$ and w contains the substring pa:
  - for all $x \in \{\triangleright, \triangleleft\} \cup \Sigma \backslash \{a, b\}$: $\mathsf{pos}(w, r_x) = \mathsf{find}(x, w)$;
  - $\mathsf{pos}(w, r_a) = \mathsf{find}(a, w) \backslash \{i_p + 1\}$;
  - $\mathsf{pos}(w, r_b) = \mathsf{find}(b, w) \cup \{i_p\}$;
  - in case $a = b$, $\mathsf{pos}(w, r_a) = \mathsf{pos}(w, r_b) = \mathsf{find}(a, w) \backslash \{i_p + 1\} \cup \{i_p\}$;
  - $\mathsf{pos}(w, r_q) = \{i_p + 1\}$, and $\mathsf{pos}(w, r_y) = \emptyset$ for all $y \in Q \backslash \{q\}$;
- when p corresponds to an instruction $cpa \to qcb$ and w contains the substring cpa:
  - for all $x \in \{\triangleright, \triangleleft\} \cup \Sigma \backslash \{a, b, c\}$: $\mathsf{pos}(w, r_x) = \mathsf{find}(x, w)$;
  - $\mathsf{pos}(w, r_a) = \mathsf{find}(a, w) \setminus \{i_p + 1\}$;
  - $\mathsf{pos}(w, r_b) = \mathsf{find}(b, w) \cup \{i_p + 1\}$;
  - in case $a = b$, $\mathsf{pos}(w, r_a) = \mathsf{pos}(w, r_b) = \mathsf{find}(a, w) = \mathsf{find}(b, w)$;
  - $\mathsf{pos}(w, r_c) = \mathsf{find}(c, w) \backslash \{i_p - 1\} \cup \{i_p\}$;
  - $\mathsf{pos}(w, r_q) = \{i_p - 1\}$, and $\mathsf{pos}(w, r_y) = \emptyset$ for all $y \in Q \backslash \{q\}$;
- when p corresponds to an instruction $p\triangleleft \to aq\triangleleft$ and w contains the substring $p\triangleleft$:
  - for all $x \in \{\triangleright\} \cup \Sigma \backslash \{a\}$: $\mathsf{pos}(w, r_x) = \mathsf{find}(x, w)$;
  - $\mathsf{pos}(w, r_a) = \mathsf{find}(a, w) \cup \{i_p\}$;
  - $\mathsf{pos}(w, r_{\triangleleft}) = \{i_p + 2\}$;
  - $\mathsf{pos}(w, r_q) = \{i_p + 1\}$, and $\mathsf{pos}(w, r_y) = \emptyset$ for all $y \in Q \backslash \{q\}$.
- when none of the above conditions are true: $\mathsf{pos}(w, r_x) = \mathsf{find}(x, w)$, for all $x \in V$.

At every computation step, function pos scans the current string w and calculates the new positions of rule right-hand sides based on the current state symbol p and the symbols appearing around it. Recall that all right-hand sides are single symbols, so in this case pos computes actual symbol positions in the resulting string. Essentially, pos arranges the reproduction of all symbols which are not concerned by the current instruction at the same positions as the ones at which the appear in w. On the other hand, for the symbols which are affected by the current instruction, pos arranges their insertion or disappearance at appropriate positions.

We remark that all positions pos produces for all rules are always distinct, i.e., if $x \neq y$, $\mathsf{pos}(w, r_x) \cap \mathsf{pos}(w, r_y) = \emptyset$, for any fixed w correctly representing a configuration of the Turing machine M. This means that the set produced by apply will always be a singleton set.

Consider now a context sequence of length $n + 1$ of the form $\gamma = (\gamma_0, \lambda, \ldots, \lambda)$, where γ_0 is a string correctly encoding an initial configuration of M. Then it follows from the construction of rss_M above—and particularly from that of pos—that the only interactive process generated by γ will correspond to an n-step computation of the Turing machine M. Note that if at some point M blocks, i.e., encounters the situation in which the current instruction is not applicable to the string, rss_M will simply maintain the tape in the same configuration for the rest of the computation induced by γ.

The example above shows the extent of the power of the (computable) function pos: it causes the reconstruction of the correct new configuration at every step, and renders the function mode essentially useless. Unsurprisingly, pos in RSS of type 2 can also directly simulate any multiplicity function mul in RSS of type 1.

Proposition 5 *For every RSS of type 1 rss_1 there exists an RSS of type 2 rss_2 over the same alphabet V and with the same set of rules R, such that the interactive processes generated by any context sequence γ are the same in rss_1 and rss_2.*

Proof The functions mctx and post are defined in exactly the same way in RSS of types 1 and 2, so we only need to show how pos in conjunction with apply in rss_2 can simulate the effect of the multiplicity function in rss_1.

Fix an arbitrary total order on the set of rules $R = \{r_1, \ldots, r_m\}$ and recursively define pos in the following way:

$$\begin{aligned}
&\mathsf{pos}(w, r_1) = \{1, \ldots, \mathsf{mul}(\mathsf{supp}^*(w), r_1)\},\\
&max_i = \max \mathsf{pos}(w, r_i),\\
&\mathsf{pos}(w, r_{i+1}) = \{max_i + 1, \ldots, max_i + 1\\
&\quad + \mathsf{mul}(\mathsf{supp}^*(w), r_{i+1})\},\ 1 \leq i < m.
\end{aligned}$$

In other words, pos induces the replication of the right-hand side of the rule r_i the number of times prescribed by $\mathsf{mul}(\mathsf{supp}^*(w), r_i)$, and then ensures that the indices associated to these right-hand sides conform to the total order on the set of rules R. This is precisely the behavior of mode, mul, and apply in RSS of type 1, which proves the statement of the proposition. □

Similarly to the discussion around Definition 8 in Sect. 6.2, we remark that in Example 13 a considerable amount of

power comes from the capability of pos to deny the application of an otherwise applicable rule. The following definition captures this property of pos.

Definition 10 The position function $\mathsf{pos} : V^* \times R \to \mathcal{P}(\mathbb{N})$ of an RSS of type 2 is called *non-denying* if, for any $w \in V^*$ and for any $r \in R$, $|\mathsf{pos}(w, r)| \geq 1$ if r is applicable to w.

Similarly to Theorem 4, RSS type of type 2 restricted to only non-denying position functions are set-compatible, for similar reasons.

Theorem 6 *All reaction systems with strings of type 2 with a non-denying position function* pos *are set-compliant.*

Proof Similarly to the proof of Theorem 4, we remark that a non-denying position function pos only affects the positions and the number of copies in which right-hand sides appear in the resulting string, and both these features have no effect on the projection supp of the current configuration on sets. Therefore, RSS of type 2 with non-denying functions are set-compliant, for the same reasons as RSS of type 1. □

7 Discussion

7.1 Summary of the results

In this work, we set out with the idea of outlining different possibilities for extending reaction systems to operate on strings, and established the first landmarks of the spectrum ranging from reaction systems as a set rewriting formalism all the way to string rewriting grammars. Figure 1 gives a graphical summary of the results. The vertical arrow represents proximity to string rewriting, set rewriting being the furthest and Turing machines the closest. The horizontal axis corresponds to the types of reaction systems with strings introduced in this paper. The dashed line illustrates the major separation between set-compliant and non-set-compliant variants (Definition 4), the possible behaviors of the set-compliant variants being quite limited. The arrow from Type 0 to Type 1 with denying mul indicates the fact that RSS of type 0 (or rather of type 0′, cf. Remark 4) can be seen as a particular case of RSS of type 1. Similarly, the arrow from Type 1 with denying mul to Type 2 with denying pos indicates that RSS of type 1 can be seen as a particular case of RSS of type 2, as formalized by Proposition 5.

The strong effect of allowing denying multiplicity functions mul and position functions pos boosting the power of the corresponding variants of reaction systems with strings is in fact not surprising, as these functions implement what is essentially appearance checking—some operations are allowed or not depending on the form of the string (Rozenberg and Salomaa 1997). Non-denying mul and pos do not have this power any more, hence the set compliance of the corresponding types of reaction systems with strings.

Most proposed extensions to strings respect the original mindset, expressed as the threshold principle and the non-permanency principle: all three types of reaction systems with strings discard symbols not explicitly reproduced, and do not impose competition for resources—the presence of a symbol is enough to render applicable any rule needing it.

Furthermore, reaction systems with strings of types 0 and 1 are deterministic, i.e., these RSS will always produce exactly one string, whatever the current configuration. As Sects. 6.1 and 6.2 show, this restricts direct simulation constructions to deterministic models of computing. Nevertheless, Example 10 shows that enough non-determinism can be injected via the context sequence, allowing to simulate non-deterministic register machines. In this sense, RSS of types 0 and 1 further comply with the original mindset in the strong connection between their behavior and the shape of the context sequence.

7.2 Open questions

We leave multiple open questions in this thoroughly exploratory work. This section lists the ones which appear to us the most interesting or promising.

1. *Limits of set-compatibility.* It appears from our results that set-compliance (Definition 4) is a strong property, requiring a direct mapping of the entire behavior space of an RSS onto the interactive processes of a conventional reaction system. We show multiple examples of RSS which are set-compliant, and others which are not. The extent of set-compatibility (Definition 5) on the other hand appears harder to characterize. We conjecture that a large class of RSS are set-compatible, covering in particular all three types of RSS presented here, but we also conjecture that there exist RSS which are not set-compatible.
2. *Beyond simple random-context rules.* In all three types of reaction systems with strings shown in this paper, the rules themselves have a very particular shape: $(\lambda \to \alpha_r, P_r, Q_r)$, where P_r and Q_r only contain individual symbols. The effect of allowing strings in P_r and Q_r remains to be explored. A particular variation might be allowing strings in P_r and Q_r, but requiring these two to be singleton sets. One important degree of freedom in such extensions would be the definitions of mul and pos: how to account for multiplicities and positions of *strings* in the permitting and forbidden contexts? Even further, what effect having non-empty rule *left*-hand sides may bring about?
3. *Derivation modes.* In this work, we mainly consider flavors of the parallel mode, employing all applicable rules at every step of the computation. Modes on the other hand

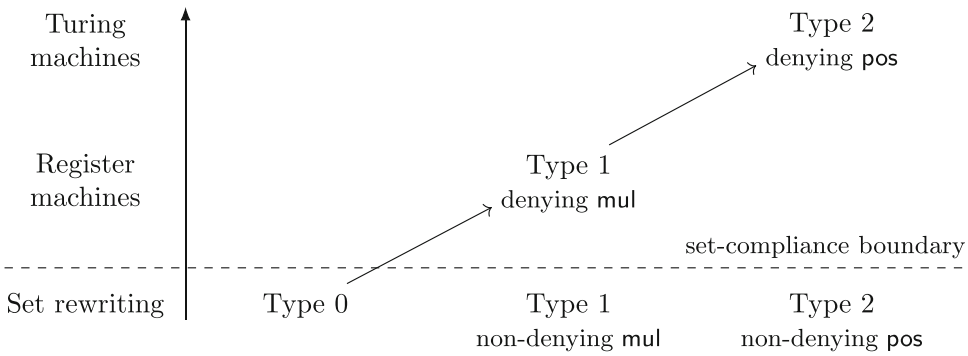

Fig. 1 A visual summary of the spectrum between reaction systems and set rewriting

are known to have a considerable impact on the computational power of a formalism [e.g., Alhazov et al. (2022)]. Exploring different derivation modes in reaction systems with strings should yield further non-trivial results.

7.3 Further connections to other models of computing

Besides the spectrum we propose in this work, further connections can be established between reaction systems and other models of computing, string-based or not. Among examples we can cite time-varying distributed Head systems (TVDH systems), which feature multiple components corresponding to test tubes containing different enzymes [e.g. Margenstern and Rogozhin (2001)]. The system transfers a set of strings from a component to another, applies all applicable splicing rules available in that component, and only keeps the results of splicing, discarding the strings that did not interact. In other words, this is an implementation of the non-permanency principle in a splicing-based model of computing. More subtly, any individual string may interact with as many other strings as the splicing rules allow, meaning that TVDH systems also implement the threshold principle.

Another string-based model of computing of the same lineage are Lindenmayer systems or L-systems, in which context-free rewriting rules are applied in parallel to a string (Păun et al. 2010; Lindenmayer 1968; Wikipedia contributors 2023). All symbols must have an associated rule, even if it is of the trivial form $a \to a$. This can be seen again as an implementation of the non-permanency principle: symbols do not explicitly degrade in L-systems, yet they need to be reproduced at every step. The determinism of reaction systems makes them specifically similar to deterministic L-systems (D0L systems), in which exactly one rule is available for every symbol. On the other hand, explicit presence of inhibitors in reaction systems potentially puts them quite apart from L-systems, which do not feature explicit inhibition. Deeper comparisons of reaction systems with strings to L-systems may give further insights into their computational power.

Finally, an exploration similar to the work we present in this paper was done in Alhazov et al. (2016a) with the goal of building a bridge between reaction systems and P systems—a hierarchical multiset rewriting-based model of computing. P systems traditionally operate on multisets of objects and not on strings, and this is the approach taken in Alhazov et al. (2016a). Nevertheless, this work presents similar ideas, in particular concerning the reinterpretation of reactions on a different substrate, relying on sets rather than richer structures, connections to TDVH systems, etc.

Acknowledgements We acknowledge the reviewers for their insightful and helpful suggestions.

Funding Artiom Alhazov acknowledges project 20.80009.5007.22 "Intelligent information systems for solving ill-structured problems, processing knowledge and big data" by the National Agency for Research and Development.

Data availibility No data associated in the manuscript.

Declarations

Conflicts of Interests The authors declare no conflict of interest—financial or non-financial—during the preparation of this work.

References

Alhazov A, Aman B, Freund R, Ivanov S (2016a) Simulating R systems by P systems. In: Leporati A, Rozenberg G, Salomaa A, Zandron C (eds) Membrane computing—17th international conference, CMC 2016, Milan, Italy, July 25–29, 2016, Revised selected papers. Lecture notes in computer science, vol 10105, pp 51–66. Springer, Berlin. https://doi.org/10.1007/978-3-319-54072-6_4

Alhazov A, Freund R, Verlan S (2016b) P systems working in maximal variants of the set derivation mode. In: Leporati A, Rozenberg G, Salomaa A, Zandron C (eds) Membrane computing—17th international conference, CMC 2016, Milan, Italy, July 25-29, 2016, Revised selected papers. Lecture notes in computer science, vol 10105, pp 83–102. Springer, Cham. https://doi.org/10.1007/978-3-319-54072-6_6

Alhazov A, Freund R, Ivanov S, Oswald M (2021) Relations between control mechanisms for sequential grammars. Fundam Informaticae 181(2–3):239–271. https://doi.org/10.3233/FI-2021-2058

Alhazov A, Freund R, Ivanov S, Oswald M (2022) Variants of derivation modes for which purely catalytic P systems are computationally complete. Theor Comput Sci 920:95–112. https://doi.org/10.1016/J.TCS.2022.03.007

Alhazov A, Ferrari-Dominguez V, Freund R, Glade N, Ivanov S (2023) A P systems variant for reasoning about sequential controllability of Boolean networks. Theor Comput Sci 970:114056. https://doi.org/10.1016/j.tcs.2023.114056

Azimi S, Iancu B, Petre I (2014) Reaction system models for the heat shock response. Fundam Informaticae 131(3–4):299–312. https://doi.org/10.3233/FI-2014-1016

Azimi S, Gratie C, Ivanov S, Petre I (2015) Dependency graphs and mass conservation in reaction systems. Theor Comput Sci 598:23–39. https://doi.org/10.1016/j.tcs.2015.02.014

Azimi S, Gratie C, Ivanov S, Manzoni L, Petre I, Porreca AE (2016) Complexity of model checking for reaction systems. Theor Comput Sci 623:103–113. https://doi.org/10.1016/j.tcs.2015.11.040

Azimi S, Panchal C, Mizera A, Petre I (2017) Multi-stability, limit cycles, and period-doubling bifurcation with reaction systems. Int J Found Comput Sci 28(08):1007–1020. https://doi.org/10.1142/S0129054117500368

Brijder R, Ehrenfeucht A, Main MG, Rozenberg G (2011) A tour of reaction systems. Int J Found Comput Sci 22(7):1499–1517. https://doi.org/10.1142/S0129054111008842

Cocke J, Minsky M (1964) Universality of tag systems with P=2. J ACM 11(1):15–20

Csuhaj-Varjú E, Vaszil G (2002) P automata or purely communicating accepting P systems. In: Paun G, Rozenberg G, Salomaa A, Zandron C (eds) Membrane computing, international workshop, WMC-CdeA 2002, Curtea de Arges, Romania, August 19–23, 2002, Revised Papers. Lecture notes in computer science, vol 2597, pp 219–233. Springer, Berlin. https://doi.org/10.1007/3-540-36490-0_14

Dassow J, Păun G (1989) Regulated rewriting in formal language theory. Springer, Heidelberg. https://www.springer.com/de/book/9783642749346

Dennunzio A, Formenti E, Manzoni L (2015a) Reaction systems and extremal combinatorics properties. Theor Comput Sci 598:138–149. https://doi.org/10.1016/J.TCS.2015.06.001

Dennunzio A, Formenti E, Manzoni L, Porreca AE (2015b) Ancestors, descendants, and gardens of Eden in reaction systems. Theor Comput Sci 608:16–26. https://doi.org/10.1016/j.tcs.2015.05.046. From Computer Science to Biology and Back

Dennunzio A, Formenti E, Manzoni L, Porreca AE (2019) Complexity of the dynamics of reaction systems. Inf Comput 267:96–109. https://doi.org/10.1016/j.ic.2019.03.006

Ehrenfeucht A, Rozenberg G (2007) Reaction systems. Fundam Informaticae 75(1–4):263–280

Freund R (2019) A general framework for sequential grammars with control mechanisms. In: Hospodár M, Jirásková G, Konstantinidis S (eds) Descriptional complexity of formal systems—21st IFIP WG 1.02 international conference, DCFS 2019, Košice, Slovakia, July 17–19, 2019, Proceedings. Lecture notes in computer science, vol 11612, pp 1–34. Springer, Cham. https://doi.org/10.1007/978-3-030-23247-4_1

Freund R, Staiger L (2019) Turing machines with activations of transitions. In: Freund R, Holzer M, Sempere JM (eds) Eleventh workshop on non-classical models of automata and applications, NCMA 2019, Valencia, Spain, July 2–3, 2019. Österreichische Computer Gesellschaft, Vienna, pp 79–91

Freund R, Verlan S (2007) A formal framework for static (tissue) P systems. In: Eleftherakis G, Kefalas P, Păun Gh, Rozenberg G, Salomaa A (eds) Membrane computing, 8th international workshop, WMC 2007, Thessaloniki, Greece, June 25–28, 2007 Revised selected and invited papers. Lecture notes in computer science, vol. 4860, pp. 271–284. Springer, Heidelberg. https://doi.org/10.1007/978-3-540-77312-2_17

Ivanov S (2015) On the power and universality of biologically-inspired models of computation. Ph.D. thesis, University of Paris-Est, France. https://tel.archives-ouvertes.fr/tel-01272318

Ivanov S, Petre I (2020) Controllability of reaction systems. J Membr Comput 2(4):290–302. https://doi.org/10.1007/S41965-020-00055-X

Ivanov S, Verlan S (2015) Random context and semi-conditional insertion-deletion systems. Fundam Informaticae 138(1–2):127–144. https://doi.org/10.3233/FI-2015-1203

Ivanov S, Verlan S (2021) Single semi-contextual insertion-deletion systems. Nat Comput 20(4):703–712. https://doi.org/10.1007/S11047-021-09861-3

Korec I (1996) Small universal register machines. Theor Comput Sci 168(2):267–301. https://doi.org/10.1016/S0304-3975(96)00080-1

Lindenmayer A (1968) Mathematical models for cellular interaction in development. J Theor Biol 18:280–315

Margenstern M, Rogozhin Yu (2001) About time-varying distributed H systems. In: Condon A, Rozenberg G (eds) DNA computing. Springer, Berlin, pp 53–62

Męski A, Penczek W, Rozenberg G (2015) Model checking temporal properties of reaction systems. Inf Sci 313:22–42. https://doi.org/10.1016/j.ins.2015.03.048

Męski A, Koutny M, Penczek W (2017) Verification of linear-time temporal properties for reaction systems with discrete concentrations. Fund Inform 154:289–306. https://doi.org/10.3233/FI-2017-1567

Męski A, Koutny M, Penczek W (2019) Model checking for temporal-epistemic properties of distributed reaction systems. Technical report, School of Computing, University of Newcastle upon Tyne

Okubo F, Kobayashi S, Yokomori T (2012a) On the properties of language classes defined by bounded reaction automata. Theor Comput Sci 454:206–221. https://doi.org/10.1016/J.TCS.2012.03.024

Okubo F, Kobayashi S, Yokomori T (2012b) Reaction automata. Theor Comput Sci 429:247–257. https://doi.org/10.1016/j.tcs.2011.12.045. Magic in Science

Păun Gh, Rozenberg G, Salomaa A (eds) (2010) The Oxford handbook of membrane computing. Oxford University Press, Oxford

Rozenberg G, Salomaa A (eds) (1997) Handbook of formal languages, vol 1–3. Springer, Berlin. https://doi.org/10.1007/978-3-642-59136-5

Salomaa A (2014) Minimal reaction systems defining subset functions. In: Calude CS, Freivalds R, Iwama K (eds) Computing with new resources—essays dedicated to Jozef Gruska on the Occasion of His 80th Birthday. Lecture Notes in Computer Science, vol 8808, pp 436–446. Springer, Cham. https://doi.org/10.1007/978-3-319-13350-8_32

Salomaa A (2015) Two-step simulations of reaction systems by minimal ones. Acta Cybern 22(2):247–257. https://doi.org/10.14232/actacyb.22.2.2015.2

Verlan S (2010) Study of language-theoretic computational paradigms inspired by biology, Paris. Habilation thesis

Wikipedia contributors (2023) L-system—Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=L-system&oldid=1182891458. Online; Accessed 19-November-2023

Yokomori T, Okubo F (2021) Theory of reaction automata: a survey. J Membr Comput 3(1):63–85. https://doi.org/10.1007/S41965-021-00070-6