Technische Universität Dresden
Fakultät Mathematik und Naturwissenschaften
Fachrichtung Mathematik

Fakultät Informatik
Institut für Theoretische Informatik
Lehrstuhl für Automatentheorie

Universality Results for Spiking Neural P Systems with Cooperating Rules

Diplomarbeit
zur Erlangung des ersten akademischen Grades

Diplommathematiker

vorgelegt von

Name:	Marx	Vorname:	Maximilian
geboren am:	1987-05-09	in:	Hamm

Tag der Einreichung: 2016-03-14

Betreuerin:	Dr.-Ing. Monika Sturm
Betreuender Hochschullehrer:	Prof. Dr.-Ing. Franz Baader

For my grandfather Heinrich, who passed away during the final days of the preparation of this thesis.

Contents

1. Introduction

1.1. Overview of the field

Since their introduction in [IPY06], Spiking Neural P systems (SN P systems) have been extensively studied in the field of *Membrane Computing* that is concerned with the study of biologically inspired models of computation. As such, it clearly is a sub-field of *Automata theory*, but is also part of the interdisciplinary field of *Computational biology*, which uses methods from computer science to model and study biological processes. Automata theory, in turn, is deeply intertwined with the study of *Formal languages*, and indeed the families of languages generated by certain types of SN P systems have been one of the focal points of research on SN P systems.

Spiking Neural systems with cooperating rules aim to introduce the concept of cooperating distributed grammar systems (CD grammar systems) to the theory of SN P systems. CD grammar systems themselves arose as a grammar-theoretic formulation of the blackboard model of problem solving (cf. [Nii89]) from *Artificial Intelligence*, where distributed knowledge sources cooperate in a structured fashion to solve a common problem. Since actual neurons also cooperate towards a common goal, this seems to be a fitting extension of the basic model of SN P systems.

1.2. Structure of this thesis

In chapter 2, we introduce the necessary concepts and tools from the theories of formal languages, Turing machines, register machines, and CD grammar systems, along with some mathematical background and a (very brief) overview of P systems. While this may seem a bit excessive, we have written this thesis with a mixed audience of Computer Scientists and Mathematicians in mind and thus could not assume familiarity with those concepts that may appear to be well-known to one or the other.

We begin chapter 3 by defining Spiking Neural P systems and proceed to introduce SN P systems with cooperating rules, which work according to some fixed cooperation protocol. We restate the universality proof for the case of the terminating protocol from [MRK14b], and proceed to prove universality for the other cooperation protocols. These universality proofs are the main results of this thesis and positively answer an open question posed by [MRK14b].

In chapter 4, we design an SN P system with cooperating rules that generates a non-semi-linear set and contrast it with a CD grammar system generating the same set. We show that this system can be assembled in a bottom-up fashion from smaller “modules,” and that the composability of such systems is one of the model’s strengths.

We end with a discussion of possible future research in chapter 5, giving several suggestions for future works, both towards further development of the theory and towards an application in the modeling of biological processes.

1.3. Acknowledgments

I wish to thank my parents, for their continued support throughout all these years, and my supervisor, Dr. Sturm, for all those fruitful discussions, for keeping me on track, and for all the helpful advice.

Furthermore, I extend my gratitude to Daniel, Felix, Juliane, and Tom, for their most helpful feedback on drafts of this thesis, to the C3 Subtitles team, for providing a welcome distraction when I most needed one, and to my friends and family, for everything.

2. Preliminaries

2.1. Basic notation

Definition 2.1
By $\mathbb{N} := \{0, 1, 2, \ldots\}$ we denote the set of *natural numbers*. We refer to the set of *positive natural numbers* by $\mathbb{N}_{>0} := \mathbb{N} \backslash \{0\}$, and denote the *prefix of length* n *of the positive natural numbers* by $\underline{n} := \{1, 2, \ldots, n\} \subsetneq \mathbb{N}_{>0}$ (note that $\underline{0} = \emptyset$). □

Definition 2.2
For a given set S, we write $\mathfrak{P}(S) := \{T \mid T \subseteq S\}$ to denote the *power set* of S. We take $|S|$ to mean the *cardinality* of S. Given two sets S and T, we say that S and T are *isomorphic* and write $S \cong T$ if there is a bijective mapping $f : S \to T$ (note that if $S \cong T$, then also $T \cong S$). We say that S is *finite* if there is an $n \in \mathbb{N}$ such that $|S| = n$, and *infinite* otherwise. By an abuse of notation, we write $|S| < \infty$ to denote that S is finite. We say that S is *countably infinite* if $S \cong \mathbb{N}$. If S is finite or countably infinite, we may say that S is *countable*, and we say that S is *uncountable* or *uncountably infinite* if S is infinite and not countable. □

Definition 2.3
For a map $f : X \to Y$, we take $\operatorname{dom} f := X$ and $\operatorname{cod} f := Y$ to mean the *domain* and *codomain* of f, respectively. Given a set $S \subseteq X$, we write $f[S] := \left\{f(x) \mid x \in S\right\} \subseteq \operatorname{cod} f$ to denote the *image* of S under f (note that, in general, $f[X] \subsetneq Y$), and $f|_S : S \to Y$ for the *restriction* of f to S. For a set $I \subseteq f[X]$, we denote by $f^{-1}[I]$ the *preimage* of I under f. Given a map $f : X \to \mathbb{N}$, we denote by $\operatorname{supp} f := \left\{x \in X \mid f(x) > 0\right\}$ the *support* of f. Given two maps $f : X \to Y$ and $g : Y \to Z$, we denote by $f \circ g := x \mapsto f(g(x))$ the *composition* of f and g. Given a map $h : X \to X$ and $i \in \mathbb{N}_{>1}$, we let $h^1 := h$, and denote by $h^i := h \circ h^{i-1}$ the i-fold *iteration* of h. □

Definition 2.4 (Product, Coproduct)
Given a finite family of sets $\mathcal{S} = (S_i)_{i \in I}$, where $I = \{i_1, i_2, \dots, i_n\}$, we write

$$\prod_{i \in I} S_i := \left\{ (s_{i_1}, s_{i_2}, \dots, s_{i_n}) \,\middle|\, \forall i \in I.\ s_i \in S_i \right\}$$

to denote the (cartesian) *product* of $\mathcal{S}$, and say that $\pi_i : \prod_{i \in I} S_i \to S_i$ are the *projections* associated with $\prod_{i \in I} S_i$. For any set T and maps $p_i : T \to S_i$ (for each $i \in I$), there is a unique map $\langle p_i \rangle_{i \in I} : T \to \prod_{i \in I} S_i$ satisfying $p_i = \pi_i \circ \langle p_i \rangle_{i \in I}$, which we call the *tupling* of the maps p_i.

Similarly, we denote by

$$\coprod_{i \in I} S_i := \bigcup_{i \in I} \{ (s, i) \mid s \in S_i \}$$

the *coproduct* of $\mathcal{S}$, and refer to the maps $\iota_i : S_i \to \coprod_{i \in I} S_i$ as the *injections* associated with $\coprod_{i \in I} S_i$. For any set T and maps $i_i : S_i \to T$ ($i \in I$), we say that the unique map $[i_i]_{i \in I} : \coprod_{i \in I} S_i \to T$ that satisfies $i_i = [i_i]_{i \in I} \circ \iota_i$ is the *cotupling* of the maps i_i.

In particular, for a given set S and a finite index set I, we may also write $S^I := \prod_{i \in I} S$ to refer to the *power* of S. Finally, if $n \in \mathbb{N}_{>0}$, we may even write S^n instead of $S^{\underline{n}}$. Given two sets S, T, we may also write $S \times T := \prod_{X \in \{S,T\}} X$ and $S \coprod T := \coprod_{X \in \{S,T\}} X$. □

Note that products and coproducts are unique only up to isomorphism. Hence, when we refer to *the* product or *the* coproduct, we always mean the (canonical) representation as defined above. For further details, we refer the reader to [Awo06].

The notation S^T is also commonly used to denote the set of all maps $T \to S$. Indeed, for finite T, every such map $f : T \to S$ corresponds to exactly one element of $\prod_{t \in T} S$, namely $f^{-1}[S]$, and for each element $(\tau_t)_{t \in T}$ of $\prod_{t \in T} S$ there is a corresponding map $t \mapsto \tau_t$.

Definition 2.5 (Graph, Tree)
A (directed) *graph* $G = (V, E)$ is a structure consisting of

- V, a finite set of *vertices*, and
- $E \subseteq V^2$ a set of *edges*.

For two vertices $u, v \in V$, we write $u \to v$ if $(u, v) \in E$ and say that there is an edge from u to v. We refer to u as the *parent* of v, and say that v is the *child* of u. Given vertices $v_0, v_1, \ldots, v_k \in V$, we say that there is a *path* $v_0 v_1 \cdots v_k$ from v_0 to v_k if $v_{i-1} \to v_i$ for $i \in \underline{k}$, and denote by $|v_0 v_1 \cdots v_k| := k$ the *length* of $v_0 v_1 \cdots v_k$. We say that two vertices $u, v \in V$ have *distance* k if there is a path from u to v of length k and no such path of length j exists for any $j < k$. We say that u and v are *adjacent* if $u \to v$ or $v \to u$. If E is symmetric, we say that G is *undirected*.

An (ordered, directed, rooted) *tree* $T = (V, E)$ is a graph such that every vertex has at most one parent, there is exactly one vertex r that is not the child of any parent, and for any vertex $v \in V$, there is a path from r to v. Furthermore, we require that the children of each vertex are ordered in some fixed (but arbitrary) way. We call r the *root* of T, and say that a vertex $v \in V$ is a *leaf* if v has no children. If v is not a leaf, we say that v is an *interior* vertex. The *height* of T is the length of of the longest path in T, i.e.,

$$\operatorname{height} T := \max\bigl\{k \in \mathbb{N} \bigm| \exists v_0, v_1, \ldots, v_k \in V.\ v_0 v_1 \cdots v_k \text{ is a path in } T\bigr\}.$$

The ordering on children extends to an ordering on the leaves by observing that for two leaves $u, v \in V$, the paths from r to u and from r to v diverge at a vertex $x \in V$. Then the ordering of the children $y, z \in V$ of x on the paths to u and v, respectively, yields the ordering of u, v (since both u and v are leaves, we have $y = z$ iff $u = v$).

A *labeled tree* is a structure $T = (V, E, \rho)$ such that (V, E) is a tree, and $\rho : V \to S$ for some set of *labels* S. The *yield* of a labeled tree is the sequence $\operatorname{yield} T := \rho(l_1), \rho(l_2), \ldots, \rho(l_k)$, where $l_1, l_2, \ldots, l_k$ are the leaves of T, ordered from least to greatest.

We represent trees by drawing the root at the top and arranging the children of each vertex, in order, underneath their parent. Since the direction of the edges is always from top to bottom, we draw the edges as simple lines. □

Definition 2.6 (Algebra)

A *signature* is a set Ω together with a map $\operatorname{ar} : \Omega \to \mathbb{N}$. We say that ar is the *type* of Ω. For any $\omega \in \Omega$, we say that $\operatorname{ar} \omega$ is the *arity* of ω.

Given a signature Ω and its associated type ar, an Ω-*algebra* is a tuple $\mathcal{A} = \left(A, (f_\omega)_{\omega\in\Omega}\right)$, where

- $A \neq \varnothing$ is a set, called the *carrier set* of $\mathcal{A}$, and
- $(f_\omega)_{\omega\in\Omega}$ is a family of finitary operations on A such that for any $\omega \in \Omega$, we have
$$f_\omega : A^{\mathrm{ar}(\omega)} \to A.$$

When Ω is finite, we may give the operations directly as members of the tuple, i.e., if $\Omega = \{f_1, f_2, \dots, f_n\}$, we may write $\mathcal{A} = (A, f_1, f_2, \dots, f_n)$. If $\mathrm{ar}(\omega) = 1$ for some $\omega \in \Omega$, we say that f_ω is a *constant*.

Let $\mathcal{A} = \left(A, (f_\omega)_{\omega\in\Omega}\right)$ be an Ω-algebra, and let $\varnothing \subsetneq S \subseteq A$. We say that $\mathcal{S} = \left(S, (g_\omega)_{\omega\in\Omega}\right)$ is a *subalgebra* of $\mathcal{A}$ and write $S \leqslant A$ if for any $\omega \in \Omega$, we have

$$\forall s_1, s_2, \dots, s_{\mathrm{ar}(\omega)} \in S.\ f_\omega\left(s_1, s_2, \dots, s_{\mathrm{ar}(\omega)}\right) = g_\omega\left(s_1, s_2, \dots, s_{\mathrm{ar}(\omega)}\right) \in S.$$

For a set $\varnothing \subsetneq B \subseteq A$, we denote by

$$\langle B\rangle_{\mathcal{A}} \coloneqq \Bigg(\bigcap_{\substack{B\subseteq S\subseteq A,\\ (S,(f_\omega)_{\omega\in\Omega})\leqslant\mathcal{A}}} S, (f_\omega)_{\omega\in\Omega}\Bigg)$$

the *subalgebra generated by* B *in* $\mathcal{A}$. We set

$$\langle\varnothing\rangle_{\mathcal{A}} \coloneqq \begin{cases} \left\langle \mathrm{ar}^{-1}\left[\{0\}\right]\right\rangle_{\mathcal{A}}, & \mathrm{ar}^{-1}\left[\{0\}\right] \neq \varnothing, \text{ and} \\ \varnothing, & \text{otherwise} \end{cases}$$ □

Example 2.7 (Power set algebra)
Given any set S, we set $\Omega \coloneqq \left\{\varnothing, S, \cap, \cup, (-)^{-1}\right\}$ and $\mathrm{ar}(\varnothing) \coloneqq 0 \eqqcolon \mathrm{ar}(S)$, $\mathrm{ar}\left((-)^{-1}\right) = 1$, and $\mathrm{ar}(\cup) \coloneqq 2 \eqqcolon \mathrm{ar}(\cap)$. Then

$$\mathcal{P}(S) \coloneqq \left(\mathfrak{P}(S), (f_\omega)_{\omega\in\Omega}\right)$$

is an algebra, where

- $f_\varnothing \coloneqq \varnothing$ is the empty set,
- $f_S \coloneqq S$ is S,

- for every $X \in \mathfrak{P}(S)$, $f_{(-)^{-1}}(X) := S \backslash X$ is the complement of X,
- for every $X, Y \in \mathfrak{P}(S)$, $f_{\cup}(X, Y) := X \cup Y$ is the union of X and Y, and
- for every $X, Y \in \mathfrak{P}(S)$, $f_{\cap}(X, Y) := X \cap Y$ is the intersection of X and Y.

We say that $\mathcal{P}(S)$ is the *power set algebra* of S. □

Definition 2.8 (Homomorphism)
Let Ω be a signature and ar the associated type, and let $\mathcal{A} = (A, (f_\omega)_{\omega\in\Omega})$ and $\mathcal{B} = (B, (g_\omega)_{\omega\in\Omega})$ be Ω-algebras. We say that a map $\phi : A \to B$ is a *homomorphism* if for all $\omega \in \Omega$ and all $a_1, a_2, \ldots, a_{\mathrm{ar}(\omega)} \in A$, we have

$$\phi\Big(f_\omega\left(a_1, a_2, \ldots, a_{\mathrm{ar}(\omega)}\right)\Big) = g_\omega\Big(\phi(a_1), \phi(a_2), \ldots, \phi\left(a_{\mathrm{ar}(\omega)}\right)\Big).$$

If ϕ is injective, we say that ϕ is a *monomorphism*. We say that ϕ is an *epimorphism* if ϕ is surjective, and if ϕ is bijective, we say that ϕ is an *isomorphism*. □

Definition 2.9 (Quotient algebra)
Let $\mathcal{A} = \left(A, (f_\omega)_{\omega\in\Omega}\right)$ be an Ω-algebra with type ar. Then we say that $\sim \subseteq A \times A$ is a *congruence relation* on $\mathcal{A}$ if $\sim$ is an equivalence relation (i.e., a reflexive, symmetric, transitive relation), and for every $\omega \in \Omega$ and all $(a_1, b_1), (a_2, b_2), \ldots, \left(a_{\mathrm{ar}(\omega)}, b_{\mathrm{ar}(\omega)}\right) \in \sim$ we have that

$$f_\omega\left(a_1, a_2, \ldots, a_{\mathrm{ar}(\omega)}\right) \sim f_\omega\left(b_1, b_2, \ldots, b_{\mathrm{ar}(\omega)}\right).$$

For an element $a \in A$, we denote by $[a]_\sim$ the *congruence class* of a, and we say that a is a *representative* of $[a]_\sim$.

Note that if $\sim_1, \sim_2$ are congruence relations on $\mathcal{A}$, then so is $\sim_1 \cap \sim_2$. Hence, for a set $R \subseteq A \times A$, we may define the *congruence relation generated by* R as

$$\langle R \rangle_{\mathcal{A}} := \bigcap \big\{\sim \,\big|\, R \subseteq \sim \subseteq A \times A, \sim \text{ congruence}\big\}.$$

Given an Ω-algebra $\mathcal{A}$ with type ar and a congruence relation $\sim$ on $\mathcal{A}$, the *quotient algebra* of $\mathcal{A}$ with respect to $\sim$ is

$$\begin{aligned} \mathcal{A}/_\sim &:= (\, A/_\sim\,, (f_\omega^\sim)_{\omega\in\Omega}), \text{ where} \\ A/_\sim &:= \big\{[a]_\sim \,\big|\, a \in A\big\}, \text{ and} \\ f_\omega^\sim\left([a_1]_\sim, [a_2]_\sim, \ldots, [a_{\mathrm{ar}(\omega)}]_\sim\right) &:= \big[f_\omega(a_1, a_2, \ldots, a_{\mathrm{ar}(\omega)})\big]_\sim. \end{aligned}$$

Note that this is indeed well-defined as the results of the operations do not depend on the chosen representatives of the equivalence classes. For details, see, e.g., [Grä08].

Finally, we denote by $\phi_\sim : \mathcal{A} \to \mathcal{A}/_\sim : a \mapsto [a]_\sim$ the *canonical homomorphism* associated with $\sim$. $\square$

Definition 2.10 (Term algebra, [BN98])

Let Ω be a signature and ar its type, and let X be a set (we say that the elements of X are *variables*). We define the set of *terms* over Ω with variables X, denoted by $T_\Omega(X)$, as follows:

- $X \subseteq T_\Omega(X)$, i.e., any variable is a term, and
- for any $n \in \mathbb{N}$, any $\omega \in \mathrm{ar}^{-1}[n]$, and any $t_1, t_2, \ldots, t_n \in T_\Omega(X)$, we have $\omega(t_1, t_2, \ldots, t_n) \in T_\Omega(X)$, i.e., we obtain terms by applying operations to terms.

The *term algebra* over Ω with variables X is the algebra

$$\mathcal{T}_\Omega(X) \coloneqq \left(T_\Omega(X), (f_\omega)_{\omega\in\Omega}\right),$$

where

$$f_\omega : T_\Omega(X)^{\mathrm{ar}(\omega)} \to T_\Omega(X) : \left(t_1, t_2, \ldots, t_{\mathrm{ar}(\omega)}\right) \mapsto \omega\left(t_1, t_2, \ldots, t_{\mathrm{ar}(\omega)}\right). \quad \square$$

Definition 2.11 (Monoid)

A monoid on a set M is an algebra $\mathcal{M} = (M, *, \lambda)$ such that

- $* : M \times M \to M$ is an associative operation on M, i.e.,
$$\forall a, b, c \in M.\ (a * b) * c = a * (b * c), \text{ and}$$
- $\lambda \in M$ is an identity element with respect to $*$, i.e.,
$$\forall m \in M.\ \lambda * m = m = m * \lambda. \quad \square$$

Example 2.12 (Monoid homomorphism)

Given two monoids $\mathcal{M} = (M, *, \lambda)$ and $\mathcal{N} = (N, +, \epsilon)$, a map $\phi : M \to N$ is a monoid *homomorphism* if for all $m, n \in M$ we have

$$\begin{aligned} \phi(\lambda) &= \epsilon, \text{and} \\ \phi(m * n) &= \phi(m) + \phi(n). \end{aligned} \quad \square$$

Definition 2.13 (Free monoid)
For a set S, we denote by

$$S^* := \left(\bigcup_{i \geqslant 0} S^i, *, \lambda \right)$$

the *free monoid* generated by S, where $*$ is concatenation, i.e., for

$$s := (s_i)_{i \in \underline{m}}, t := (t_i)_{i \in \underline{n}} \in S^*,$$

we have

$$\begin{aligned}(s_1, \dots, s_n) * (t_1, \dots, t_m) &:= (s_1, \dots, s_n, t_1, \dots, t_m), \\ \lambda * t &:= t, \text{and} \\ s * \lambda &:= s.\end{aligned}$$

We interpret the elements of S^* as *words* over the alphabet S and write $s_1 \cdots s_m$ to denote $(s_1, \dots, s_m) \in S^m$, and denote by λ the *empty word*. Slightly abusing the notation, we may also write S^* when referring to its carrier set, i.e., the set of words over S. Used in that way, we refer to the operation $(\cdot)^*$ as the *Kleene star*. □

Indeed, the free monoid is an instance of the concept of a *free object* (see [Awo06] for a definition). As such, it is unique only up to isomorphisms. Hence, when we say that some structure *is* the free object (of a certain kind), we actually mean that it is *a* free object, and we may even say that two different (but isomorphic) structures *are* the free object.

Example 2.14
The monoid $(\mathbb{N}, +, 0)$ of natural numbers is a free monoid generated by $\{1\}$: We interpret the strings in $\{1\}^*$ as unary numerals. Clearly,

$$\{1\}^* \ni \underbrace{11 \cdots 1}_{n \text{ times}} \mapsto n \in \mathbb{N}$$

is a monoid isomorphism. □

Example 2.15
Let S be a finite, nonempty set and S^* be the free monoid over S. We define $\Omega \coloneqq \{*, \lambda\}$, and set $\mathrm{ar}(*) = 2$ and $\mathrm{ar}(\lambda) = 0$. Finally, we let $\sim$ be the congruence generated as follows:

$$\begin{aligned}\sim &\coloneqq \langle I \cup A \rangle_{\mathcal{T}_\Omega(S)}, \text{ where}\\ I &\coloneqq \Big\{ \big(*(s,\lambda), s\big), \big(*(\lambda, s), s\big) \;\Big|\; s \in S \Big\}, \text{ and}\\ A &\coloneqq \Big\{ \Big(*\big(s, *(t,u)\big), *\big(*(s,t), u\big) \Big) \;\Big|\; s, t, u \in S \Big\}\end{aligned}$$

Then the free monoid is isomorphic to a quotient of the term algebra:

$$S^* \cong \mathcal{T}_\Omega(S)/_\sim . \qquad \square$$

Definition 2.16 (Free commutative monoid)
Let S be a set. Then by

$$S^\odot \coloneqq S^*/_\sim$$

we denote the *free commutative monoid*, where $\sim$ is the congruence relation generated by

$$\big\{ (ab, ba) \mid a, b \in S \big\}. \qquad \square$$

2.2. Formal languages

We provide a short introduction to formal languages and an overview of the classification of formal languages known as the Chomsky hierarchy as presented in [MS97b].

Definition 2.17 (Language, [MS97b])
An *alphabet* is a finite, nonempty set V. We refer to the elements of V as *symbols*. The elements of V^* are called *words*, λ in particular is called the *empty word*. Given two words $x, y \in V^*$, we denote by $xy \coloneqq x * y$ the *concatenation* of x and y with respect to the free monoid over V. We denote by $V^+ \coloneqq V^* \backslash \{\lambda\}$ the set of all *non-empty words* over V, and for $n \in \mathbb{N}_{>0}$, we write x^n as a shorthand for $x * x^{n-1}$, and set $x^0 \coloneqq \lambda$. For a word $w \coloneqq w_1 w_2 \cdots w_n$, we let $|w| \coloneqq n$ be the *length* of w, and we denote by

$$|w|_a \coloneqq \big|\{ i \in \underline{n} \mid w_i = a \}\big|$$

the *number of occurrences* of a in w.

A *language* over V is a subset $L \subseteq V^*$. If $L \subseteq V^+$, we say that L is λ-*free*. Given two languages $L \subseteq V, M \subseteq V'$, we denote by $LM := \{lm \mid l \in L, m \in M\}$ the concatenation of L and M (note that LM is a language over $V \cup V'$). For a given language L, the *length set* of L is

$$\operatorname{length} L := \{|w| \mid w \in L\}. \qquad \square$$

Definition 2.18 (Grammar, [MS97b])

A *phrase-structure grammar* is a tuple $G = (N, T, S, P)$, where

- N, T are disjoint alphabets (we call their elements *non-terminal* and *terminal* symbols, respectively),
- $S \in N$ is the *start symbol*, and
- $P \subseteq V^*NV^* \times V^*$ is the set of *production rules*, where $V := N \cup T$. For an element $(u, v) \in P$, we write $u \to v$.

For $x, y \in V^*$, we write $x \Longrightarrow y$ iff $x = aub$ and $y = avb$ for some $a, b \in V^*$ and a production rule $u \to v \in P$. By $\Longrightarrow^*$ we denote the reflexive and transitive closure of $\Longrightarrow$. The *language generated* by G is

$$\mathcal{L}(G) := \{x \in T^* \mid S \Longrightarrow^* x\}.$$

We classify grammars by the types of production rules, and call a grammar

context-sensitive if every rule $u \to v \in P$ is of the form $u = aAb$, $v = axb$ for some $a, b \in V^*, A \in N, x \in V^+$ (if S does not appear in the right-hand side of any rule, we also allow a rule $S \to \lambda$ to be present),

context-free if for every rule $u \to v \in P$, we have $u \in N$,

linear if each rule $u \to v \in P$ is such that $u \in N$ and $v \in T^* \cup T^*NT^*$, and

regular if for every $u \to v \in P$ we have $u \in N$ and $v \in T \cup TN \cup \{\lambda\}$.

Note that, while, e.g., [HU79] allow production rules in regular grammars to contain more than one terminal symbol, any such rule can be transformed into the form above by introducing additional non-terminal symbols. As the definition above is also found in [Pău02; PRS10], this seems to be the more appropriate approach.

By RE, CS, CF, LIN, and REG, we denote the families of languages generated by arbitrary, context-sensitive, context-free, linear, and regular grammars, respectively, i.e.,

$$\begin{aligned} \mathrm{RE} &:= \{\mathcal{L}(G) \mid G \text{ is an arbitrary grammar}\}, \\ \mathrm{CS} &:= \{\mathcal{L}(G) \mid G \text{ is a context-sensitive grammar}\}, \\ &\vdots \\ \mathrm{REG} &:= \{\mathcal{L}(G) \mid G \text{ is a regular grammar}\}. \end{aligned}$$

We may also refer to the members of RE, CS, CF, LIN, and REG as recursively enumerable, context-sensitive, context-free, linear, and regular languages, respectively. For such a family

$$\mathcal{F} \in \{\mathrm{RE}, \mathrm{CS}, \mathrm{CF}, \mathrm{LIN}, \mathrm{REG}\},$$

we denote by $\mathcal{N}\mathcal{F}$ the subset of languages $L \in \mathcal{F}$ over $\mathbb{N}$, and by $\Psi\mathcal{F}$ the subset of languages $L \in \mathcal{F}$ over alphabets $\bigcup_{i \in \mathbb{N}_{>0}} \mathbb{N}^i$, i.e., writing $\mathcal{N}\mathrm{RE}$ we refer to the sets of natural numbers generated by arbitrary grammars, and we refer to the sets of tuples of natural numbers generated by context-free grammars as $\Psi\mathrm{CF}$. □

Remark 2.19 ([Pău02])

For a family $\mathcal{F} \in \{\mathrm{RE}, \mathrm{CS}, \mathrm{CF}, \mathrm{LIN}, \mathrm{REG}\}$ of languages, the family $\mathcal{N}\mathcal{F}$ consists exactly of the length sets of languages in $\mathcal{F}$, i.e.,

$$\mathcal{N}\mathcal{F} = \{\operatorname{length} F \mid F \in \mathcal{F}\}.$$

Furthermore, $\Psi\mathcal{F}$ consists exactly of the images of languages under the Parikh mapping (see Definition 2.30), i.e.,

$$\Psi\mathcal{F} = \big\{\Psi[F] \bigm| F \in \mathcal{F}\big\}.$$

The words of such a language $F \in \Psi\mathcal{F}$ are commonly referred to as “Parikh vectors.” However, while $\mathbb{N}^n$ embeds into the vector space $\mathbb{Q}^n$, it does not form a vector space itself, but rather a module over the semiring $(\mathbb{N}, 0, +, 1, \cdot)$. Hence, we prefer to speak of *Parikh tuples* or *Parikh images* instead. □

Remark 2.20 (The λ-convention, [KRS97])
We follow an established convention in the study of formal languages and treat languages as equal if they differ by λ, i.e., for any two languages L, L' we say that

$$L \text{ equals } L' \iff L \cup \{\lambda\} = L' \cup \{\lambda\}.$$

We shall see that the equality of certain families of languages generated by Spiking Neural P systems is only due to this convention. □

2.2.1. The Chomsky hierarchy

From the definition above, we immediately obtain

$$\begin{aligned} \mathrm{REG} \subseteq \mathrm{LIN} \subseteq \mathrm{CF}, &\text{ and} \\ \mathrm{CS} \subseteq \mathrm{RE}. \end{aligned}$$

In fact, even the following can be shown:

Proposition 2.21 (Chomsky hierarchy, [MS97b])
The following well-known strict inclusions hold:

$$\mathrm{REG} \subsetneq \mathrm{LIN} \subsetneq \mathrm{CF} \subsetneq \mathrm{CS} \subsetneq \mathrm{RE}.$$ □

Proposition 2.22 ([Pău02])
Considering only families of languages of natural numbers, parts of the hierarchy collapse:

$$\mathcal{N}\mathrm{REG} = \mathcal{N}\mathrm{LIN} = \mathcal{N}\mathrm{CF} \subsetneq \mathcal{N}\mathrm{CS} \subsetneq \mathcal{N}\mathrm{RE}.$$ □

2.2.2. Regular languages

Regular languages play an important role in the definition of Spiking neural P systems. Hence, we need a concise way of describing regular languages and regular expressions are that. The following definition formalizes regular expressions as defined by [Sip97] and used by [IPY06].

Definition 2.23 (Regular expression)
Let V be an alphabet, $\Omega \coloneqq \{\varnothing, \lambda, \circ, \cup, (-)^*, (-)^+\}$ and set $\mathrm{ar}(\lambda) \coloneqq 0 \eqqcolon \mathrm{ar}(\varnothing)$, $\mathrm{ar}\big((-)^*\big) \coloneqq 1 \eqqcolon \mathrm{ar}\,\big((-)^+\big)$, and $\mathrm{ar}(\circ) \coloneqq 2 \eqqcolon \mathrm{ar}(\cup)$. Furthermore, let $\sim$ be the congruence generated as follows:

$$\begin{aligned}
\sim &\coloneqq \langle I \cup S \cup A \rangle_{\mathcal{T}_\Omega(V)} \\
I &\coloneqq \Big\{ \big(*(v, \lambda), v\big), \big(*(\lambda, v), v\big) \,\Big|\, v \in V \Big\}, \\
S &\coloneqq \Big\{ \big(\cup(v, w), \cup(w, v)\big) \,\Big|\, v, w \in V \Big\}, \text{ and} \\
A &\coloneqq \Big\{ \big(\omega\big(v, \omega(w, x)\big), \omega\big(\omega(v, w), x\big)\big) \,\Big|\, v, w, x \in V, \omega \in \{\circ, \cup\} \Big\}.
\end{aligned}$$

For clarity, we may write the operations $\circ$ and $\cup$ as infix operators, or even omit $\circ$ entirely.

We define the algebra of *regular expressions* over V by setting

$$\mathcal{R}\mathrm{egEx}(V) \coloneqq \mathcal{T}_\Omega\,(V)/_\sim$$

and denote by $\mathrm{RegEx}(V)$ its carrier set.

With each regular expression we associate a language over S by defining the following mapping which extends to a homomorphism into the power set algebra over V^*:

$$\begin{aligned}
\mathcal{L} : \mathrm{RegEx}(V) &\to V^* \text{ such that for all } v, w \in V : \\
\mathcal{L}(\varnothing) &\coloneqq \varnothing, \\
\mathcal{L}(\lambda) &\coloneqq \{\lambda\}, \\
\mathcal{L}(v) &\coloneqq \{v\}, \\
\mathcal{L}(v^*) &\coloneqq \mathcal{L}(v)^*, \\
\mathcal{L}(v^+) &\coloneqq \big\{w \,\big|\, w \in \mathcal{L}(v^*)\backslash\{\lambda\}\big\}, \\
\mathcal{L}(v \cup w) &\coloneqq \mathcal{L}(v) \cup \mathcal{L}(w), \text{and} \\
\mathcal{L}(v \circ w) &\coloneqq \big\{xy \,\big|\, x \in \mathcal{L}(v), y \in \mathcal{L}(w)\big\}.
\end{aligned}$$

For a regular expression $r \in \mathrm{RegEx}(V)$, we say that $\mathcal{L}(r)$ is the *language matched* by r. □

Proposition 2.24 ([MS97b, Section 3.2])
The languages matched by regular expressions are exactly the regular languages:

$$\mathrm{REG} = \big\{\mathcal{L}(r) \,\big|\, \exists V.\ r \in \mathrm{RegEx}(V)\big\}. \qquad \square$$

2.2.3. Context-free languages

We introduce some concepts pertaining to context-free languages. In particular, we are interested in proving Parikh's theorem, a useful tool in showing that a given language is not context-free.

To aid in the proof, we introduce the Chomsky normal form and derivation trees. Whereas [HU79] restrict the Chomsky normal form to λ-free languages, the definition (and the proof of Proposition 2.26) are easily adapted to languages containing λ.

Definition 2.25 (Chomsky normal form, [HU79])
Let $G = (N, T, S, P)$ be a context-free grammar. We say that G is in *Chomsky normal form* if every production rule is of one of the following forms:

- $A \to BC$ for some $A, B, C \in N$,
- $A \to a$ for some $A \in N$, $a \in T$, or
- $S \to \lambda$ if $\lambda \in \mathcal{L}(G)$. □

Proposition 2.26 ([HU79, Theorem 4.5])
Let G be a context-free grammar. Then there exists an equivalent context-free grammar G' in Chomsky normal form, i.e.,

$$\mathcal{L}(G) = \mathcal{L}(G'). \qquad \square$$

Definition 2.27 (Derivation tree, [HU79])
Let $G = (N, T, S, P)$ be a context-free grammar. A labeled tree $D = (V, E, \rho)$ with root r is a *derivation tree* (or *parse tree*) for G if

1. $\rho : V \to N \cup T \cup \{\lambda\}$,
2. $\rho(r) = S$,
3. for any interior vertex v (i.e., a vertex that is not a leaf), $\rho(v) \in N$,
4. for a vertex n with children $n_1, n_2, \ldots, n_k$ with labels $\rho(n) = A$, $\rho(n_i) = X_i$ (for $i \in \underline{k}$), there is a production $A \to X_1 X_2 \cdots X_k \in P$, and

5. if $\rho(v) = \lambda$ for a vertex v, then v is a leaf and the only child of its parent.

A *subtree* of D is a tree $D' = (V', E', \rho|_{V'})$ such that

- $V' \subseteq V$,
- $E' := E \cap (V')^2$, and
- if $u \in V'$ and $u \to v$ in D, then $v \in V'$ (i.e., for any vertex u in D', the children of u in D are also in D'). □

Derivation trees correspond to the repeated application of production rules in the derivation of a word. For the proof of the following proposition, see [HU79].

Proposition 2.28 ([HU79, Theorem 4.1])
Let $G = (N, T, S, P)$ be a context-free grammar. Then $S \Longrightarrow^* w$ iff there is a derivation tree for G with yield w. □

Remark 2.29
If G is a grammar in Chomsky normal form, any derivation tree for G is such that any interior vertex has either exactly two children, none of which are leaves, or exactly one child that is a leaf, and any leaf is labeled with exactly one terminal symbol. Consider a derivation tree D for G of height h. Then any maximal path in this tree consists of exactly h interior vertices and one leaf, and $|\operatorname{yield} D| = 2^{h-1}$. □

We now introduce the definitions we need to state Parikh's theorem.

Definition 2.30 (Parikh mapping)
Given an alphabet $V = \{v_1, \ldots, v_n\}$, let $\phi : V^* \to V^{\odot}$ be the canonical homomorphism. Then

$$\psi : V^{\odot} \to (\mathbb{N}, +, 1)^n : w \mapsto \big(|w|_{v_1}, \ldots, |w|_{v_n}\big)$$

is a monoid isomorphism. We define the *Parikh mapping* associated with V as follows:

$$\Psi := \psi \circ \phi.$$ □

Strictly speaking, ψ depends on the ordering of $v_1, \ldots, v_n$ of elements of V. However, any permutation on $\underline{n}$ (and hence any permutation of $v_1, \ldots, v_n$) lifts to an automorphism on $(\mathbb{N}, +, 1)^n$, so we can assume a consistent ordering throughout this thesis.

Definition 2.31 (Semi-linear set, [Par66])
Let $S \subseteq \mathbb{N}^n$ for some $n \in \mathbb{N}_{>0}$. We say that S is *linear* if there are $a_0, a_1, \ldots, a_m \in \mathbb{N}^n$ such that

$$S = \left\{ a_0 + \sum_{i \in \underline{n}} n_i a_i \;\middle|\; n_1, \ldots, n_m \in \mathbb{N} \right\},$$

and *semi-linear* if S is a finite union of linear sets. □

Theorem 2.32 (Parikh, [Par66])
Let $L \in \mathrm{CF}$. *Then* $\Psi[L]$ *is semi-linear.* □

The original proof in [Par66] is quite technical, and the proof in [ABB97] makes use of the theory of equation systems over commutative semigroups that we do not wish to introduce here. Instead, we reproduce the proof in [Gol77], which makes use only of the basic theory of formal languages. We note that [Kui97] proves a generalized version of Theorem 2.32 for arbitrary semirings.

For the proof, we need a (slightly strengthened) version of the Pumping lemma for context-free languages.

Lemma 2.33 (Pumping lemma, [Gol77])
Let $G = (N, T, S, P)$ *be a context-free grammar. Then there is an integer* p *such that, for any* $k \geqslant 1$, *if* $w \in \mathcal{L}(G)$ *and* $|w| \geqslant p^k$, *any derivation* $S \Longrightarrow^* w$ *is equivalent to*

$$\begin{aligned} S &\Longrightarrow^* uAv \\ &\Longrightarrow^* ux_1Ay_1v \\ &\Longrightarrow^* ux_1x_2Ay_2y_1v \\ &\Longrightarrow^* \cdots \\ &\Longrightarrow^* ux_1x_2\cdots x_kAy_k\cdots y_2y_1v \\ &\Longrightarrow^* ux_1x_2\cdots x_kzy_k\cdots y_2y_1v = w, \end{aligned}$$

where $A \in N$, $\lambda \notin \{x_iy_i \mid i \in \underline{k}\}$, *and* $|x_1x_2\cdots x_kzy_k\cdots y_2y_1| \leqslant p^k$. □

Note For $k = 1$, we obtain the pumping lemma as stated in [BPS61; HU79]. □

Proof (Lemma 2.33) We adapt the proof from [HU79] to the strengthened statement of Lemma 2.33. Without loss of generality, we assume that G is in Chomsky normal form, and that $\mathcal{L}(G)$ is λ-free (since we are concerned only with words of a certain minimum length, shorter words are irrelevant).

First, observe that if $w \in \mathcal{L}(G)$ has a derivation tree of height at most i, then $|w| \leqslant 2^{i-1}$. For $i = 1$, the derivation tree must consist of exactly two vertices, and we obtain $w \in D$. Thus, we have $|w| = 1 = 2^0$. Consider now a derivation tree D of height $i > 1$. Then D is as described in Remark 2.29, and the children of the root vertex are themselves roots of subtrees D_1, D_2 of height (at most) $i - 1$. By the induction hypothesis, we obtain $|\text{yield}\, D_j| \leqslant 2^{i-2}$ for $j = 1, 2$. Hence, $|\text{yield}\, D| = |\text{yield}\, D_1\, \text{yield}\, D_2| \leqslant 2^{i-1}$.

Set $p := 2^{|N|}$ and let $k \in \mathbb{N}_{>0}$. Consider $w \in \mathcal{L}(G)$ with $|w| \geqslant p^k$. Then we have

$$|w| \geqslant \left(2^{|N|}\right)^k = 2^{k|N|} > 2^{k|N|-1},$$

and thus any derivation tree for w must have height at least $k|N| + 1$. Hence, a maximal path in a derivation tree for w must have length at least $k|N| + 1$ (for simplicity, we assume without loss of generality that it has length exactly $k|N| + 1$), and therefore consists of $k|N| + 2$ vertices, only one of which is a leaf. Since the remaining $k|N|+1$ vertices are labeled with non-terminal symbols, of which there are exactly $|N|$, by the pigeonhole principle, there must be a symbol $A \in N$ such that at least k vertices are labeled with A. Consider such a maximal path, and let $v_1, v_2, \ldots, v_k$ be those vertices, ordered by decreasing distance to the leaf. Note that the distance of v_1 to the leaf is at most $k|N| + 1$. Consider the subtrees $D_1, D_2, \ldots, D_k$ with roots $v_1, v_2, \ldots, v_k$, respectively, and denote by $w_i :=$ yield D_i their yields. Since D_1 has height at most $k|N| + 1$ (because the path is maximal), we have $|w_1| \leqslant 2^{k|N|} = p^k$. But w_1 must be of the form $x_1 w_2 y_1$, since v_2 is closer to the leaf than v_1, and D_2 must be completely contained in one of the two subtrees starting at children of v_1 (because both D_1 and D_2 are of the form as described in Remark 2.29). Hence, $x_1 y_1 \neq \lambda$.

Analogously, we obtain $w_2 = x_2 D_3 y_2$ up to $w_{k-1} = x_{k-1} D_k y_{k-1}$, and finally $w_k = x_k z y_k$. Now, we have

$$|x_1 x_2 \cdots x_k z y_k \cdots y_2 y_1| = |w_1| \leqslant p^k,$$

and clearly we have

$$w = u w_1 v = u x_1 x_2 \cdots x_k z y_k \cdots y_2 y_1$$

for some $u, v \in (N \cup T)^*$. ■

Proof (Theorem 2.32, [Gol77]) Let $G = (N, T, S, P)$ be a grammar satisfying $\mathcal{L}(G) = L$. Let p be the constant obtained from Lemma 2.33. For any set $U \subseteq N$ with $S \in U$, set

$$L_U \coloneqq \{w \in L \mid \exists D = (V, E, \rho) \text{ derivation tree for } w.\ \rho[V] \cap N = U\}.$$

Since N is finite, there are only finitely many L_U, and clearly

$$\bigcup_{\{S\} \subseteq U \subseteq N} L_U = L.$$

We show that each $\Psi[L_U]$ is semi-linear, which proves the claim.

Let $U \subseteq N$ be such that $S \in U$. From now on, we only consider derivations using productions $A \to v$ in P such that $A \in U$ and $v \in (U \cup T)^*$. Let $k \coloneqq |U|$, and set

$$\begin{aligned} F &\coloneqq \{w \in L_U \mid |w| < p^k\}, \text{ and} \\ G &\coloneqq \{xy \mid 1 \leqslant |xy| \leqslant p^k \text{ and } A \Longrightarrow^* xAy \text{ for some } A \in U\}. \end{aligned}$$

We claim that $\Psi[L_U] = \Psi[FG^*]$. Consider $w \in L_U$. If $|w| < p^k$, then $w \in F \subseteq FG^*$. Otherwise, we have $|w| \geqslant p^k$. Since $w \in L_U$, there is a derivation $S \Longrightarrow^* w$ using exactly the non-terminal symbols in U. By

Lemma 2.33, this derivation is equivalent to a derivation

$$\begin{aligned} S \underbrace{\Longrightarrow^*}_{d_0} & uAv \\ \underbrace{\Longrightarrow^*}_{d_1} & ux_1Ay_1v \\ \underbrace{\Longrightarrow^*}_{d_2} & \cdots \\ \underbrace{\Longrightarrow^*}_{d_k} & ux_1x_2\cdots x_kAy_k\cdots y_2y_1v \\ \underbrace{\Longrightarrow^*}_{d_{k+1}} & ux_1x_2\cdots x_kzy_k\cdots y_2y_1v = w, \end{aligned}$$

where $A \in U$, $x_iy_i \in G$ for any $i \in \underline{k}$, and $d_0, d_1, \ldots, d_{k+1}$ are certain distinguished sub-derivations. Let $f : U\backslash\{A\} \to \{d_i \mid i \in \underline{k}\}$ be injective. Then, since $\left|U\backslash\{A\}\right| = k - 1$, there is a $j \in \underline{k}$ such that $d_j \notin f\left[U\backslash\{A\}\right]$. Thus,

$$S \Longrightarrow^* ux_1x_2\cdots x_{j-1}x_{j+1}\cdots x_kzy_k\cdots y_{j+1}y_{j-1}\cdots y_2y_1v =: w',$$

and $w' \in L_U$. Since $|w'| < |w|$, we can assume $\Psi(w') \in \Psi[FG^*]$ by induction. We obtain $\Psi(w) = \Psi(w'x_jy_j) \in \Psi[FG^*]$, since $x_jy_j \in G$, and hence, we have $\Psi[L_U] \subseteq \Psi[FG^*]$.

Conversely, let $w \in FG^*$. If $w \in F$, then $w \in L_U$. Otherwise, $w = w_0s$ for some $w_0 \in FG^*$ and $s \in G$. Then $s = xy$, where $A \Longrightarrow^* xAy$ for some $A \in U$. Since $|w_0| < |w|$, we obtain $\Psi(w_0) = \Psi(w')$ for some $w' \in L_U$ by induction. Hence, $S \Longrightarrow^* w'$, and every non-terminal symbol in U (including A) occurs in this derivation. Thus, we have

$$\begin{aligned} S &\Longrightarrow^* uAv \Longrightarrow^* uzv = w', \text{ and} \\ S &\Longrightarrow^* uAv \Longrightarrow^* uxAyv \Longrightarrow^* uxzyv = w'', \text{ where} \\ \Psi(w'') &= \Psi(w'xy) = \Psi(w_0s) = \Psi(w). \end{aligned}$$

Since $w'' \in L_U$, $\Psi(w) = \Psi(w'') \in \Psi[L_U]$, and we have $\Psi[FG^*] \subseteq \Psi[L_U]$.

Now we have $\Psi[L_U] = \Psi[FG^*]$. Let $G = \{s_1, s_2, \ldots, s_m\}$. Then

$$\Psi[FG^*] = \Psi[Fs_1^*s_2^*\cdots s_m^*] = \left\{ v + \sum_{i\in\underline{m}} w_i \;\middle|\; v \in \Psi[F], w_i \in \Psi[s_i^*] \right\}.$$

Since F is finite, $\Psi[F]$ is semi-linear, and clearly, $\Psi[s_i^*]$ is linear for each $i \in \underline{m}$. Hence, $\Psi[L_U] = \Psi[FG^*]$ is semi-linear. ■

2.2.4. L systems

L systems were introduced in [Lin68a; Lin68b] by Aristid Lindenmayer to model the growth of filamentous organisms. Certain classes of L systems, however, form a hierarchy like the Chomsky hierarchy, the L hierarchy. It is primarily this hierarchy, not the L systems themselves, that we are interested in. Hence, we give a brief overview of the relevant classes of L systems and the families of languages generated by them based on [KRS97]. L systems also serve to introduce the parallel mode of rule application (in contrast to grammars, where a single rule is applied at a time), which, as we shall see later, is a defining aspect of P systems.

Remark 2.34 ([KRS97, p. 254])
While “L system” (or, similarly, “0L language”) may not be typographically correct, omitting the hyphen is an established practice in the field of L systems. We choose to follow this practice, favoring notational consistency over typographic correctness, and we shall later see that this practice has also been adopted in other fields, such as CD grammar systems, or even P systems. □

Definition 2.35 (Finite substitution, [KRS97])
Let V, W be alphabets, and $\sigma : V^* \to \{\varnothing \subsetneq L \subseteq W^* \mid L \text{ finite}\}$ be a mapping. We say that σ is a *finite substitution* if

- $\sigma(\lambda) = \lambda$, and
- for any $u, v \in V^*$, $\sigma(uv) = \sigma(u)\sigma(v)$.

If $\lambda \notin \sigma(v)$ for all $v \in V$, we say that σ is *λ-free* (or *non-erasing*). Furthermore, if for any $v \in V$, we have that $|\sigma(v)| = 1$, we say that σ is a *morphism*.

We extend σ onto a language $L \subseteq V^*$ by defining

$$\sigma(L) := \bigcup_{w \in L} \sigma(w).$$ □

Note that a finite substitution $\sigma : V^* \to \{\varnothing \subsetneq L \subseteq W^* \mid L \text{ finite}\}$ is a monoid homomorphism from V^* to $\{\varnothing \subsetneq L \subseteq W^* \mid L \text{ finite}\}^*$. As such, σ is uniquely determined by the image of $V \cup \{\lambda\}$, and given that we require $\sigma(\lambda) = \lambda$, it suffices to specify the images of the letters of V. Hence, we may also define a finite substitution in the form of production rules $V \to W^*$, where we have a rule $v \to w$ for each $w \in \sigma(v)$.

Definition 2.36 (0L system, [KRS97])
A *zero-interaction L system* (0L system) is a tuple $S = (V, \sigma, a)$, where

- V is an alphabet,
- $\sigma : V^* \to \{\varnothing \subsetneq L \subseteq V^* \mid L \text{ finite}\}$ is a finite substitution on V, and
- $a \in V^*$ is the *axiom*.

The *language generated* by S is

$$\mathcal{L}(S) := \bigcup_{i \geqslant 0} \sigma^i(a).$$

An *extended L system* (E0L system) is a tuple $E = (V, \sigma, a, T)$, where

- (V, σ, a) is a 0L system, and
- $T \subseteq V$ is the alphabet of *terminal* symbols (we refer to the elements of $V \backslash T$ as *non-terminal* symbols).

The *language generated* by E is

$$\mathcal{L}(E) := \Big(\mathcal{L}\big((V, \sigma, a)\big)\Big) \cap T.$$

A *tabled L system* (T0L system) is a tuple $T = (V, S, a)$, where

- $S := \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ is a finite set of finite substitutions over V, and
- for any $i \in \underline{n}$, (V, σ_i, a) is a 0L system.

We refer to the elements of S as *tables*. The *language generated* by T is

$$\mathcal{L}(T) := \{a\} \cup \bigcup_{k \geqslant 1} \big\{ (\sigma_{i_1} \circ \sigma_{i_2} \circ \cdots \circ \sigma_{i_k})(a) \;\big|\; i_1, i_2, \ldots, i_k \in \underline{n} \big\}.$$

We say that T is extended if every table of T is extended. We denote by $0L, E0L$, and $ET0L$, respectively, the families of languages generated by 0L, E0L, and ET0L systems, respectively. □

Proposition 2.37 ([KRS97; Pău02])
The families of languages generated by certain classes 0L systems relate to each other and to the Chomsky hierarchy in the following way:

$$\mathrm{CF} \subsetneq \mathrm{E0L} \subsetneq \mathrm{ET0L} \subsetneq \mathrm{CS} \text{ and}$$
$$\mathcal{N}\mathrm{CF} \subsetneq \mathcal{N}\mathrm{E0L} \subsetneq \mathcal{N}\mathrm{ET0L} \subsetneq \mathcal{N}\mathrm{CS}\,. \qquad \square$$

2.3. Turing machines

Intuitively, a Turing machine is a computational device consisting of a finite set of states, an infinite tape (divided into cells each storing a symbol over a (finite) alphabet) and a movable head. In each step, the symbol on the tape at the current position of the head can be read and written, the head can move one position to the left or to the right, and then a new state is chosen.

The following definition adapts the definition in [HU79] to the treatment of non-determinism in [Sip97], while defining configurations in a manner that is compatible with [Min67].

Definition 2.38 (Turing machine)
A (non-deterministic) *Turing machine* is a tuple

$$\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, B, F),$$

where

- Q is a finite set of *states*,
- Σ is an alphabet, called the *input alphabet*, such that $B \notin \Sigma$,
- Γ is an alphabet, referred to as the *tape alphabet*, such that $B \in \Gamma$ and $\Sigma \subseteq \Gamma$,
- $\delta : Q \times \Gamma \to \mathfrak{P}\left(Q \times \Gamma \times \{L, R\}\right)$ is the *transition function*,
- $q_0 \in Q$ is the *initial state*,
- B is the *blank symbol*, and

- $F \subseteq Q$ is the set of *final states*.

A *configuration* of $\mathcal{M}$ is a tuple $C_{\mathcal{M}} = (q, s, l, r)$, where

- $q \in Q$ is the *current state* of $\mathcal{M}$,
- $s \in \Gamma$ is the *current symbol*, i.e., the symbol on the tape at the position of the head,
- $l : \mathbb{N} \to \Gamma$ is a map associating with each cell to the left of the head the contents of the tape, and
- $r : \mathbb{N} \to \Gamma$ is a map giving the contents of the tape to the right of the head.

Given a word $w_0 w_1 \cdots w_{|w|} =: w \in \Sigma^*$, we denote by

$$C^0_{\mathcal{M}}(w) := (q_0, w_0, x \mapsto B, \omega)$$

the *starting configuration* of $\mathcal{M}$ on input w, where $\omega : \mathbb{N} \to \Gamma$ is such that

$$\omega(i) := \begin{cases} w_{i+1}, & i+1 \in \underline{|w|} \\ B, & \text{otherwise.} \end{cases}$$

We say that a configuration $C_{\mathcal{M}} = (q, s, l, r)$ is an *accepting configuration* if $q \in F$.

Given two configurations $C_{\mathcal{M}} := (q, s, l, r)$ and $C'_{\mathcal{M}} := (q', s', l', r')$, we say that $\mathcal{M}$ makes a transition from $C_{\mathcal{M}}$ to $C'_{\mathcal{M}}$ and write $C_{\mathcal{M}} \implies C'_{\mathcal{M}}$ if there is $(q', \hat{s}, d) \in \delta(q, s)$ and

- if $d = L$, we have $s' = l(0)$, $l'(i) := l(i+1)$ for any $i \in \mathbb{N}$, $r'(0) := \hat{s}$, and $r'(i) := r(i-1)$ for any $i \in \mathbb{N}_{>0}$, and
- otherwise, we have $d = r$, $s' = r(0)$, $r'(i) := r(i+1)$ for any $i \in \mathbb{N}$, $l'(0) := \hat{s}$, and $l'(i) := l(i-1)$ for any $i \in \mathbb{N}_{>0}$.

We write $C_{\mathcal{M}} \implies^* C'_{\mathcal{M}}$ if there are configurations $C^1_{\mathcal{M}}, C^2_{\mathcal{M}}, \ldots, C^n_{\mathcal{M}}$ such that

$$C_{\mathcal{M}} \implies C^1_{\mathcal{M}} \implies C^2_{\mathcal{M}} \implies \cdots \implies C^n_{\mathcal{M}} \implies C'_{\mathcal{M}},$$

and we may omit the indices on configurations if $\mathcal{M}$ is understood from the context.

The *language accepted* by $\mathcal{M}$ is

$$\mathcal{L}(\mathcal{M}) := \left\{ w \in \Sigma^* \;\middle|\; C^0_{\mathcal{M}}(w) \Longrightarrow^* C_{\mathcal{M}} \text{ accepting} \right\}.$$

Without loss of generality, we may assume that $\mathcal{M}$ *halts* (i.e., there are no further transitions that $\mathcal{M}$ can make from the current configuration) whenever $\mathcal{M}$ reaches an accepting configuration. By $\mathcal{L}_{\mathrm{TM}}$ we denote the class of languages accepted by Turing machines.

Finally, we say that $\mathcal{M}$ is *deterministic* if

$$\max_{(q,s)\in Q\times\Gamma} |\delta(q,s)| \leqslant 1. \qquad \square$$

Proposition 2.39 ([Sip97, Theorem 3.10])
For every non-deterministic Turing machine $\mathcal{M}$, there is an equivalent deterministic Turing machine $\mathcal{D}$, i.e.,

$$\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{D}). \qquad \square$$

Remark 2.40 (The Universal Turing machine)
There exists a Turing machine $\mathcal{U}$ that is *universal* in the sense that for any description of a Turing machine $\mathcal{T}$ and a starting configuration $C^0_{\mathcal{T}}(w)$ of $\mathcal{T}$, $\mathcal{U}$ computes the output of $\mathcal{T}$ on $C^0_{\mathcal{T}}(w)$. For a construction of a Universal Turing machine, see [Min67, sec. 7.2], or even Turing's original paper [Tur36, sec. 6]. $\square$

Theorem 2.41 (Universality of arbitrary grammars)
The languages generated by arbitrary grammars are exactly the languages accepted by Turing machines, i.e.,

$$\mathrm{RE} = \mathcal{L}_{\mathrm{TM}}. \qquad \square$$

Proof (Theorem 2.41, [MS97a]) Let $L \in \mathrm{RE}$ and $G = (N, T, S, P)$ be a grammar such that $\mathcal{L}(G) = L$. We construct a non-deterministic Turing machine $\mathcal{M}$ such that $\mathcal{L}(M) = L$ as follows: For any input $w \in T^*$, M non-deterministically chooses a position i in w and a production rule $u \to v \in P$. If v occurs in w at position i, replace v by u, adjusting the position of β

on the tape if $|u| \neq |v|$. $\mathcal{M}$ accepts w iff after a finite number of steps, the tape contains S. Clearly, $\mathcal{L}(\mathcal{M}) = L$.

Conversely, let $L \in \mathcal{L}_{TM}$ and $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be a Turing machine such that $\mathcal{L}(\mathcal{M}) = L$. We construct a grammar $G = (N, \Sigma, S_0, P)$ such that $\mathcal{L}(G) = L$, where

$$N := \big((\Sigma \cup \{\lambda\}) \times \Gamma\big) \cup Q \cup \{S_0, S_1, S_2\}, \text{ and}$$

P contains all rules of the following forms:

1. $S_0 \to q_0 S_1$,
2. $S_1 \to (a, a) S_1$ (for some $a \in \Sigma$),
3. $S_1 \to S_2$,
4. $S_2 \to (\lambda, B) S_2$,
5. $S_2 \to \lambda$,
6. $q(a, X) \to (a, Y)p$ iff $(p, Y, R) \in \delta(q, X)$ (where $a \in \Sigma \cup \{\lambda\}$, $p, q \in Q$, and $X, Y \in \Gamma$),
7. $(b, Z)q(a, X) \to p(b, Z)(a, Y)$ iff $(p, Y, L) \in \delta(q, X)$ (where $a, b \in \Sigma \cup \{\lambda\}$, $p, q \in Q$, and $X, Y, Z \in \Gamma$), and
8. $(a, X)q \to qaq$, $q(a, X) \to qaq$, $q \to \lambda$ (for some $q \in F$ and some $a \in \Sigma \cup \{\lambda\}$).

Using rules of the first five forms, we obtain a derivation

$$S_0 \Longrightarrow^* q_0(a_1, a_1) \cdots (a_n, a_n)(\lambda, B)^m,$$

where $a_i \in \Sigma$ ($i \in \underline{n}$) and $m \geqslant 0$. We then use rules of forms 6 and 7 to simulate the transitions of $\mathcal{M}$. When $\mathcal{M}$ reaches a final state, rules of form 8 become applicable, and the resulting word is $w = a_1 a_2 \cdots a_n$. Clearly, $\mathcal{M}$ accepts w iff w is derivable in G, and hence we have

$$\mathcal{L}(M) = \mathcal{L}(G). \qquad \blacksquare$$

Remark 2.42 (Church's Thesis, [MS97b])
Introduced by Alonzo Church in [Chu36], Church's Thesis equates Turing machines with intuitively effective procedures, i.e., it states that for any intuitively effective procedure, there exists an equivalent Turing machine. Since the notion of an intuitively effective procedure cannot be formalized, Church's Thesis cannot be proven. However, to date no counterexample has been found, whereas lots of formal models of computability have been shown to be equivalent to Turing machines. □

2.4. Register machines

Register machines are particularly well-suited to prove the computational completeness of classes of P systems, as simulating them is possible by implementing only three instructions.

As we are interested in simulating a given register machine by a P system, we strive for a definition of register machines with as few instructions as possible. Hence, the approach taken by [Weg05] (where a register machine is equipped with instructions for e.g., multiplication and division) is unsuitable for our purposes. [EP02] give, among others, a definition for a register machine executing "GOTO programs," which are closely related to the ones we describe here.

The deterministic variant of the register machine we describe is as is used in [MRK14b], and resembles the definition of a "program machine" given by [Min67].

Definition 2.43 (Register machine)
A *(deterministic) register machine* is a tuple $\mathcal{M} = (m, H, l_0, l_h, I)$ such that

- $m \in \mathbb{N}_{>0}$ is the *number of registers*,
- H is a set of *instruction labels*,
- $l_0 \in H$ is the *starting label*,
- $l_h \in H$ is the *halting label*, and
- $I \cong H$ is the *instruction set*, and each *instruction* $i \in I$ is of one of the following forms:

- $\big(\mathrm{ADD}(r), l_j\big)$ (increment register r and jump to instruction l_j)
- $\big(\mathrm{SUB}(r), l_j, l_k\big)$ (if register r is 0, jump to l_k, otherwise decrement register r and jump to l_j)
- HALT (halt the computation).

We denote by $\phi_{\mathcal{M}} : H \to I$ the isomorphism mapping instruction labels to the corresponding instructions, and we require that $\phi_{\mathcal{M}}(l_h) = \mathrm{HALT}$. If the register machine can be inferred from the context, we may omit the index.

A *configuration* of a register machine $\mathcal{M} = (m, H, l_0, l_h, I)$ is a pair $C_{\mathcal{M}} = (i, \rho)$, where

- $i \in I$ is the current instruction, and
- $\rho : R \subseteq \underline{m} \to \mathbb{N}$ is a map assigning values to registers. We say that a register $r \in \underline{m}$ is *non-empty* iff $r \in \operatorname{supp} \rho$, and *empty* otherwise.

We may also write $C_{\mathcal{M}} = (i; r_1, r_2, \ldots, r_m)$ if m is small, and omit the index if $\mathcal{M}$ is obvious. By $C^0_{\mathcal{M}}(n) := (\phi(l_0), 1 \mapsto n)$ we denote the *starting configuration* of $\mathcal{M}$ for input $n \in \mathbb{N}$, and by $C^h_{\mathcal{M}} := (\phi(l_h), 1 \mapsto 1)$ the *halting configuration* of $\mathcal{M}$. We denote by $\mathcal{C}(\mathcal{M})$ the set of configurations of $\mathcal{M}$.

Given two configurations $C_{\mathcal{M}} = (i, \rho)$ and $C'_{\mathcal{M}} = (i', \rho')$, we say that $\mathcal{M}$ makes a transition from $C_{\mathcal{M}}$ to $C'_{\mathcal{M}}$ and write $C_{\mathcal{M}} \Longrightarrow_{\mathcal{M}} C'_{\mathcal{M}}$ iff either

- $i = \big(\mathrm{ADD}(r), l\big)$, $i' = \phi(l)$, $\rho'(r) = \rho(r) + 1$, and $\rho|_{\underline{m}\setminus\{r\}} = \rho'|_{\underline{m}\setminus\{r\}}$, i.e., C' arises from C by incrementing the register r and jumping to the instruction i',
- $i = \big(\mathrm{SUB}(r), l_j, l_k\big)$, $i' = \phi(l_j)$, $\rho(r) > 0$, $\rho'(r) = \rho(r) - 1$, and $\rho|_{\underline{m}\setminus\{r\}} = \rho'|_{\underline{m}\setminus\{r\}}$, i.e., C' arises from C by decrementing the non-empty register r and jumping to i', or
- $i = \big(\mathrm{SUB}(r), l_j, l_k\big)$, $i' = \phi(l_k)$, $\rho(r) = 0$, and $\rho' = \rho$, i.e., C' arises from C by jumping to i'.

We write $C \Longrightarrow^k C'$ if there are configurations $C = C_0, C_1, \ldots, C_k = C' \in \mathcal{C}(\mathcal{M})$ such that $C_{i-1} \Longrightarrow C_i$ for $i \in \underline{k}$ and some $k \in \mathbb{N}_{>0}$, and $C \Longrightarrow^* C'$ if there exits some $k \in \mathbb{N}_{>0}$ such that $C \Longrightarrow^k C'$.

We say that $\mathcal{M}$ *accepts* input n iff

$$C^0_{\mathcal{M}}(n) \Longrightarrow^* C^h_{\mathcal{M}},$$

and denote by

$$\mathcal{L}(\mathcal{M}) := \{n \in \mathbb{N} \mid \mathcal{M} \text{ accepts } n\}$$

the set of numbers accepted by $\mathcal{M}$. □

Remark 2.44 (Composition of Register machines)
Given two register machines, we may construct a register machine that executes both in sequence by taking the coproducts of the corresponding label and instruction sets, and replacing the halting label of the first machine by the starting label of the second machine. This justifies composing register machines from smaller components, akin to "subroutines" as commonly used in imperative programming languages. □

2.4.1. Universality

The following theorem is shown in [Min67]. It needs to be adapted because the definitions of both Turing and register machines are different from our definitions. As this result is vital to the universality results for Spiking Neural P systems with cooperating rules, we choose to reproduce it in full. [Kor96, (a2)] gives an example of a Universal register machine with 22 instructions, but depends on the theory of recursive functions that we do not wish to elaborate on here.

Theorem 2.45 (Universality of register machines, [Min67])
The family of languages accepted by deterministic register machines is exactly the family $\mathcal{N}\mathrm{RE}$ *of recursively enumerable sets of natural numbers.* □

Lemma 2.46
Let $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ *be a Turing machine, and let* $C = (q, s, l, r)$ *be a configuration of* $\mathcal{M}$ *such that* $C^0(w) \Longrightarrow^* C$ *for some input* $w \in \Sigma^*$. *Let* $f \in \{l, r\}$. *Then there are only finitely many indices* $n \in \mathbb{N}$ *such that* $f(n) \neq B$. □

Proof (Lemma 2.46) Consider $C^0(w) = (q^0, s^0, l^0, r^0)$. Then $l^0(n) = B = r^0(n)$ for $n \geqslant |w|$. Each transition increases the number of indices by at most one. Since $C^0(w) \Longrightarrow^* C$ in finitely many steps, the claim follows. ■

PROOF (THEOREM 2.45) We adapt the proof in [Min67] to our definition of register machines. Let $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be a Turing machine. We construct a register machine simulating $\mathcal{M}$.

Without loss of generality, we assume that $\mathcal{M}$ is deterministic, and we further assume that $\mathcal{M}$ halts whenever it reaches an accepting configuration. Consider $(q, s, l, r) =: C_\mathcal{M} \implies C'_\mathcal{M} := (q', s', l', r')$. Since the registers of register machines store natural numbers, we need to encode these configurations as tuples of numbers. Since Q is finite, we fix an (arbitrary) enumeration $\chi : Q \to \underline{|Q|}$. Γ is also finite, but we choose $\gamma : \Gamma \to \underline{|\Gamma|}$ such that γ is an isomorphism yielding

$$(\Gamma, \oplus, B) \cong \left(\underline{|\Gamma|}, +, 0\right) =: \mathbb{Z}/_{|\Gamma|}$$

for a suitably defined $\oplus$, where $\mathbb{Z}/_{|\Gamma|}$ is the finite cyclic group of order $|\Gamma|$. Let $i_f := \max\{i \in \mathbb{N} \mid f(i) \neq B\}$. Using this, we obtain the following representation of l and r:

$$\begin{aligned} \eta : \Gamma^\mathbb{N} &\to \mathbb{N} \\ f &\mapsto \sum_{i=0}^{i_f} \left(\gamma\big(f(i)\big) \cdot |\Gamma|^i\right). \end{aligned}$$

Note that due to Lemma 2.46, the maximum i_f exists. Thus, we have

$$(q, s, l, r) \mapsto \big(\chi(q), \gamma(s), \eta(l), \eta(r)\big) \in \mathbb{N}^4,$$

a representation of C as a tuple of natural numbers. Set

$$\begin{aligned} H(n) &:= \max\big\{i \in \mathbb{N} \,\big|\, i \cdot |\Gamma| \leqslant n\big\} \text{ and} \\ P(n) &:= n - H(n). \end{aligned}$$

Let $(q', \hat{s}, d)$ be the unique element in $\delta(q, s)$. Observe that we can express C' in terms of C in the following way:

$$\begin{aligned} s' &= P\big(\eta(l)\big) \cdot \delta_{dL} + P\big(\eta(r)\big) \cdot \delta_{dR}, \\ l' &= \big(|\Gamma| \cdot \eta(l) + \hat{s}\big) \cdot \delta_{dR} + H\big(\eta(l)\big) \cdot \delta_{dL}, \text{ and} \\ r' &= \big(|\Gamma| \cdot \eta(r) + \hat{s}\big) \cdot \delta_{dL} + H\big(\eta(r)\big) \cdot \delta_{dR}, \end{aligned}$$

where δ_{xy} is the *Kronecker delta*, i.e.,

$$\delta_{xy} = \begin{cases} 1, & x = y, \text{ and} \\ 0, & \text{otherwise.} \end{cases}$$

Thus, if we can compute these values on a register machine, we can simulate $\mathcal{M}$. Computing both addition and multiplication is straightforward, and [Min67, p. 203] gives an example of a register machine computing $H(n)$ and $P(n)$ for the case $|\Gamma| = 2$ that easily generalizes. By setting aside a register that always contains 0, we can jump to arbitrary instructions using the SUB instruction (we reserve register 5 for that purpose). Similarly, we may set a register to 0 by repeatedly decrementing it, and assign arbitrary numbers by then incrementing it as necessary. From these components, we assemble register machines corresponding to each $(q, s) \in Q \times \Gamma$, and for each $q \in Q$, a register machine dispatching to the components for $\big(q, \rho(3)\big)$ according to the register encoding the current tape symbol s. Consider the input word $w = w_0 w_1 \cdots w_{|w|} \in \Sigma^*$. Then, we start the compound register machine $\mathcal{R}$ in the starting configuration

$$C^0_{\mathcal{R}} = \left(\sum_{i+0}^{|w|} \gamma(w_i) \cdot |\Gamma|^i \right).$$

$\mathcal{R}$ then proceeds to assign $P\big(\rho(1)\big)$ to register 3 and $H\big(\rho(1)\big)$ to register 1, and jumps to the dispatching instruction corresponding to the state q_0.

For a state q and current symbol s, let $(q', \hat{s}, d)$ be the unique element of $\delta(q, s)$. Then $\mathcal{R}$ first assigns $|\Gamma| \cdot \rho\big(1 + \delta_{dR}\big) + \hat{s}$ to register 4, then assigns contents of register 4 to register $1 + \delta_{dR}$. Next, $\mathcal{R}$ assigns $H\big(\rho(1 + \delta_{dL})\big)$ to register 4, $P\big(\rho(1 + \delta_{dL})\big)$ to register 3, and $\rho(4)$ to register 1. If $q \in F$, $\mathcal{R}$ halts, and otherwise it jumps to the dispatching instruction for state q'.

Clearly $\mathcal{R}$ accepts $w \in \Sigma^*$ iff $\mathcal{M}$ accepts w, i.e., $\mathcal{R}$ simulates $\mathcal{M}$. By Theorem 2.41 we know that for any language $L \in \mathcal{N}\mathrm{RE}$, there is a Turing machine $\mathcal{M}$ such that $\mathcal{L}(\mathcal{M}) = L$. Using the above construction, we obtain a register machine $\mathcal{R}$ with $\mathcal{L}(\mathcal{R}) = L$. ■

[Min67] shows that it is possible to reduce the number of registers to two and still retain universality. The key observation is to make use of the isomorphism $(\mathbb{N}, +, 0)^5 \cong (\mathbb{N}, \cdot, 1)$ given by

$$(m, n, a, z, w) \mapsto 2^m 3^n 5^a 7^z 11^w,$$

which arises from the uniqueness of prime factorization. The increment and decrement operations now need to be implemented by multiplying and dividing by the respective prime, and the second register is needed to implement these operations.

2.4.2. Non-deterministic register machines

A straightforward extension used by [MRK14b] is the introduction of non-determinism into the model of register machines. To that end, we modify the ADD instruction to introduce a second instruction label.

Definition 2.47 (Non-deterministic register machine)
A *non-deterministic register machine* $\mathcal{M} = (m, H, l_0, l_h, I)$ is defined in the same way as a deterministic register machine, except that instead of $(\mathrm{ADD}(r), l_j)$, we have instructions of the form $(\mathrm{ADD}(r), l_j, l_k)$ (i.e., we add a second instruction label), and we allow transitions from $C_{\mathcal{M}} = (i, \rho)$ to $C'_{\mathcal{M}} = (i', \rho')$ if $i = \mathrm{ADD}(r)$, $\rho'(r) = \rho(r) + 1$, $\rho|_{\underline{m}\setminus\{r\}} = \rho'|_{\underline{m}\setminus\{r\}}$, and $i' \in \phi\big[\{l_j, l_k\}\big]$, i.e., if C' arises from C by incrementing the register r and jumping to the instruction labeled by either l_j or l_k. □

Lemma 2.48
For any deterministic register machine $\mathcal{M}$, there is an equivalent non-deterministic register machine $\mathcal{M}'$ accepting the same language, i.e.,

$$\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{M}'). \qquad \square$$

Proof (Lemma 2.48) Let $\mathcal{M} = (m, H, l_0, l_h, I)$ be a deterministic register machine. We Construct $\mathcal{M}' := (m, H, l_0, l_h, I')$ by replacing each instruction $(\mathrm{ADD}(r), l)$ in I with $(\mathrm{ADD}(r), l, l)$ in I', and copying the remaining instructions, i.e.,

$$\begin{aligned} I' := I \cap \Big(&\{\mathrm{HALT}\} \cup \big\{(\mathrm{SUB}(r), l_j, l_k) \mid (\mathrm{SUB}(r), l_j, l_k) \in I\big\}\Big) \\ &\cup \big\{(\mathrm{ADD}(r), l, l) \mid (\mathrm{ADD}(r), l) \in I\big\}. \end{aligned}$$

Clearly, $\mathcal{M}'$ accepts $n \in \mathbb{N}$ iff $\mathcal{M}$ accepts n. ■

2.4.3. Generating register machines

Instead of considering register machines accepting sets of natural numbers, we can also look at register machines working in the generating mode, as is done by [MRK14b].

Definition 2.49 (Generating register machine)
A *generating register machine* is a (possibly non-deterministic) register machine $\mathcal{M} = (m, H, l_0, l_h, I)$ with a *starting configuration* $C^0_{\mathcal{M}} := (l_0, \varnothing)$ and a *halting configuration* $C^h_{\mathcal{M}}(n) := (l_h, 1 \mapsto n)$.

We say that $\mathcal{M}$ *generates* $n \in \mathbb{N}$ iff

$$C^0_{\mathcal{M}} \Longrightarrow^* C^h_{\mathcal{M}}(n),$$

and denote by

$$\mathcal{N}(\mathcal{M}) := \{n \in \mathbb{N} \mid \mathcal{M} \text{ generates } n\}$$

the *set of numbers generated by* $\mathcal{M}$. □

Unfortunately, deterministic generating register machines are strictly less powerful than their accepting counterparts: Clearly, due to branching only on decrementing an empty register, they can only ever generate sets of size at most 1. Luckily, introducing non-determinism is sufficient to remediate that.

Theorem 2.50 (Universality of generating register machines)
The sets of numbers generated by non-deterministic register machines are exactly the sets of numbers accepted by (deterministic) register machines. □

Proof (Theorem 2.50) Let $S \in \mathcal{N}RE$. By Theorem 2.45 we know that there exists a deterministic register machine $\mathcal{M}$ satisfying $\mathcal{L}(\mathcal{M}) = S$, and using Lemma 2.48 we know that it has a non-deterministic equivalent $\mathcal{M}' = (m, H, l_0, l_h, I)$. We construct a non-deterministic register machine $\mathcal{G}$ generating S in the following way: We start by repeatedly incrementing register 1 (thus creating the starting configuration $C^0_{\mathcal{M}'}(n)$ for some $n \in \mathbb{N}$), non-deterministically jumping to l_0, running $\mathcal{M}'$, and, upon reaching l_h, checking whether $\mathcal{M}'$ accepts n. If it does, we create $C^h_{\mathcal{G}}(n)$ and halt, otherwise we jump into an infinite loop.

Let $\Phi : H \to I$ be such that $\Phi[H] = I$, and set $\mathcal{G} = (m+2, H', l_{0'}, l_{h'}, I')$, where

- $H' \coloneqq H \coprod \{l_{0'}, l_a, l_b, l_{h'}, l_\alpha, l_\beta, l_\gamma, l_\omega\}$,
- $I' = \Phi'[H']$, where

$$\Phi'(h) \coloneqq \begin{cases} \big(\mathrm{ADD}(m+2), l_0, l_a\big), & h = \iota_2(l_{0'}) \\ \big(\mathrm{ADD}(1), l_b, l_b\big), & h = \iota_2(l_a) \\ \big(\mathrm{ADD}(m+1), l_{0'}, l_0\big), & h = \iota_2(l_b) \\ \big(\mathrm{SUB}(1), l_\alpha, l_\omega\big), & h = \iota_1(l_h) \\ \big(\mathrm{SUB}(1), l_\omega, l_\beta\big), & h = \iota_2(l_\alpha) \\ \big(\mathrm{SUB}(m+1), l_\gamma, l_{h'}\big), & h = \iota_2(l_\beta) \\ \big(\mathrm{ADD}(1), l_\beta, l_\beta\big), & h = \iota_2(l_\gamma) \\ \mathrm{HALT}, & h = \iota_2(l_{h'}) \\ \big(\mathrm{ADD}(1), l_\omega, l_\omega\big), & h = \iota_2(l_\omega) \text{ and} \\ \Phi(h), & \text{otherwise.} \end{cases}$$

Clearly, for every $n \in S$ we have that $\mathcal{M}'$ accepts n, hence there exist configurations $C^0_{\mathcal{M}'}(n) = (l_0, 1 \mapsto n), C^h_{\mathcal{M}'} = (l_h, 1 \mapsto 1)$ such that $C^0_{\mathcal{M}'}(n) \Longrightarrow^* C^h_{\mathcal{M}'}$.

By construction, we obtain configurations

$$\begin{aligned} C^0_{\mathcal{M}'}(n)' &\coloneqq \big(l_0, \{1 \mapsto n, m+1 \mapsto n, m+2 \mapsto \xi\}\big), \\ {C^h_{\mathcal{M}'}}' &\coloneqq \big(l_h, \{1 \mapsto 1, m+1 \mapsto n, m+2 \mapsto \xi\}\big) \end{aligned}$$

of $\mathcal{G}$ (for some $\xi \in \{0,1\}$) that differ only in the contents of registers $m+1$ and $m+2$. Clearly, we now have

$$C^0_{\mathcal{G}} \Longrightarrow^* C^0_{\mathcal{M}'}(n)' \Longrightarrow^* {C^h_{\mathcal{M}'}}' \Longrightarrow^* C^h_{\mathcal{G}}, \text{ i.e.,}$$

$\mathcal{G}$ generates n. Thus we have $S \subseteq \mathcal{N}(\mathcal{G})$. Conversely, consider $n \notin S$. Then, starting from $C^0_{\mathcal{M}'}(n)$, we never reach $C^h_{\mathcal{M}'}$, so if $\mathcal{M}'$ halts, register 1 is either empty or contains a value different from 1 (otherwise we would have a halting configuration). In either case, $\mathcal{G}$ never halts, so we obtain

$$S = \mathcal{N}(\mathcal{G}). \qquad \blacksquare$$

2.4.4. Register machines with multiple output registers

Another extension described in [MRK14b] is to use the first $n \in \mathbb{N}$ registers as output registers. This allows the register machine to generate sets of tuples $(o_1, \dots, o_n) \in \mathbb{N}^n$. Clearly, the set of all languages over $\mathbb{N}^n$ generated by non-deterministic register machines in this way is ΨRE.

2.5. CD grammar systems

Introduced in [CD90], CD grammar are a realization of the *blackboard model* of problem solving, in which multiple agents, in turn, contribute to the solution of a common problem on the blackboard, according to some sort of cooperation protocol. [MRK14b] introduces the concepts of cooperation and distribution to the theory of SN P systems. We base the following short overview of the definition and known results on the expressive power of CD grammar systems on [DPR97].

Definition 2.51 (CD grammar system, [DPR97])
A *cooperating distributed* (CD) grammar system of degree $n \geqslant 1$ is a tuple

$$\Gamma = (N, T, S, P_1, P_2, \dots, P_n),$$

where N, T are disjoint alphabets (of *non-terminal* and *terminal* symbols, respectively), $S \in N$, and for any $i \in \underline{n}$, (N, T, S, P_i) is a phrase-structure grammar. We refer to $P_1, P_2, \dots, P_n$ as the *components* of Γ. □

Here, the components represent the agents cooperating on the derivation of a word, where the current sub-derivation is written on the blackboard. We can interpret the non-terminal symbols as questions being asked (introduced) and answered (eliminated). In this fashion, the components can communicate in a limited manner.

In the following, unless explicitly stated otherwise, we will always assume the components of CD grammar systems to be context-free: [CD90] also considers CD grammar systems with E0L components. We, however, follow [Pău02] and [DPR97] in restricting ourselves to context-free components. Since the SN P systems we are interested in only deal with a singleton alphabet, this is not an actual restriction. [Csu+94, Lemma 5.1] notes that CD grammar systems with context-sensitive or recursively enumerable components themselves generate the families CS or RE, respectively.

For passing control between different components, we have the choice of one of several cooperation protocols.

Definition 2.52 (Derivation, [DPR97])
Let $\Gamma = (N, T, S, P_1, P_2, \dots, P_n)$ be a CD grammar system, let $i \in \underline{n}$, and let $V := N \cup T$. We define the

- *terminating derivation* by the i-th component, which we denote by $\Longrightarrow^{t}_{P_i}$, where $x \Longrightarrow^{t}_{P_i} y$ iff $x \Longrightarrow^{*}_{P_i} y$ and there is no $z \in V^*$ with $y \Longrightarrow_{P_i} z$,
- the k-*steps derivation* by the i-th component, written $\Longrightarrow^{=k}_{P_i}$, where $x \Longrightarrow^{=k}_{P_i} y$ iff there are $v_1, v_2, \dots, v_k \in V^*$ such that
$$x = v_1 \Longrightarrow_{P_i} v_2 \Longrightarrow_{P_i} \cdots \Longrightarrow_{P_i} v_k = y,$$
- the *at most* k-*steps derivation* by the i-th component, denoted $\Longrightarrow^{\leqslant k}_{P_i}$, where $x \Longrightarrow^{\leqslant k}_{P_i}$ iff $x \Longrightarrow^{=k'}_{P_i} y$ for some $k' \in \underline{k}$, and
- the *at least* k-*steps derivation* by the i-th component, written $\Longrightarrow^{\geqslant k}_{P_i}$, where $x \Longrightarrow^{\geqslant k}_{P_i} y$ iff $x \Longrightarrow^{=k'}_{P_i} y$ for some $k' \geqslant k$.

Furthermore, we have $\Longrightarrow^{*}_{P_i}$, the *arbitrary derivation* by the i-th component. □

Definition 2.53 (Cooperation protocol, [DPR97])
Let $\Gamma = (N, T, S, P_1, P_2, \dots, P_n)$ be a CD grammar system, and let $\mathcal{D} := \{t, *\} \cup \bigcup_{k \geqslant 1} \{= k, \geqslant k, \leqslant k\}$. We refer to the elements of $\mathcal{D}$ as *cooperation protocols*. The *language generated* by Γ working according to the protocol $f \in \mathcal{D}$ (working in the f mode) is

$$\mathcal{L}_f(\Gamma) := \bigcup_{m \geqslant 1} \{w \in T^* \mid \exists i_1, i_2, \dots, i_m \in \underline{n}.\ S \Longrightarrow^{f}_{P_{i_1}} v_1 \Longrightarrow^{f}_{P_{i_2}} \cdots \Longrightarrow^{f}_{P_{i_m}} w\}$$

Given $n \in \mathbb{N}_{>0}$, we denote by $CD_n(f)$ the family of languages generated by (context-free) λ-free (i.e., systems where each component is λ-free) CD grammar systems of degree at most n working in the derivation mode $f \in \mathcal{D}$, and by $CD(f)$ the family of languages generated by

λ-free CD grammar systems with an unrestricted number of components. If λ-rules are allowed, we denote the corresponding family by $\mathrm{CD}_n^\lambda(f)$ or $\mathrm{CD}^\lambda(f)$, respectively. Finally, we define $\mathrm{CD}_\infty(=) \coloneqq \bigcup_{k\in\mathbb{N}} \mathrm{CD}_\infty(=k)$ and $\mathrm{CD}_\infty(\geqslant) \coloneqq \bigcup_{k\in\mathbb{N}} \mathrm{CD}_\infty(\geqslant k)$. □

Proposition 2.54 ([DPR97, Theorem 3.1])
The following inclusions of families of languages hold:

1. For $f \in \{=1, \geqslant 1, *\} \cup \{\leqslant k \mid k \in \mathbb{N}_{>0}\}$, we have $\mathrm{CD}_\infty(f) = \mathrm{CF}$,
2. $\mathrm{CF} = \mathrm{CD}_1(f) \subsetneq \mathrm{CD}_2(f) \subseteq \mathrm{CD}_r(f) \subseteq \mathrm{CD}_\infty(f) \subsetneq \mathrm{CS}$ for $f \in \{=k, \geqslant k \mid k \geqslant 2\}$ and $r \geqslant 3$,
3. $\mathrm{CD}_r(=k) \subseteq \mathrm{CD}_r(=sk)$ for $k, r, s \in \mathbb{N}_{>0}$,
4. $\mathrm{CD}_r(\geqslant k) \subseteq \mathrm{CD}_r(\geqslant k+1)$ for $r, k \geqslant 1$,
5. $\mathrm{CD}_\infty(\geqslant) \subseteq \mathrm{CD}_\infty(=)$,
6. $\mathrm{CF} = \mathrm{CD}_1(t) = \mathrm{CD}_2(t) \subsetneq \mathrm{CD}_3(t) = \mathrm{CD}_\infty(t) = \mathrm{ET0L}$, and
7. except for $\mathrm{CD}_\infty(f) \subsetneq \mathrm{CS}$, all these inclusions hold for CD grammar systems with λ-rules. Instead, we have $\mathrm{CD}_\infty^\lambda(f) \subsetneq \mathrm{RE}$. □

2.6. P systems

Introduced by Păun in [Pău00], P systems (or membrane systems) are a computational model inspired by the structure and features of biological membranes. As noted in [PR02], the original purpose of P systems was *not* to model the functioning of biological membranes, but to “explore the computational nature of various features of membranes.” Similarly, [Pău02] declares that “we have no intention here of making any claim of direct interest to the (traditional) biologist,” but also states “returning meaningful information to biology” as a possible important target for future research. Indeed, attempts to model real-world biological processes have been made. As a particular example, [RP08] models the quorum sensing behavior of a certain type of bacteria using (modified) P systems.

We purposefully limit ourselves to a very brief overview on P systems here, as they differ significantly from the Spiking Neural P systems introduced in chapter 3, which we describe in more detail. For a detailed description, we refer to [Pău02] and [PR02].

Definition 2.55 (P system, [Pău02])
A (symbol-object) *P system* of degree $m \in \mathbb{N}_{>0}$ is a tuple

$$\Pi = (O, \mu, w_1, w_2, \dots, w_m, R_1, R_2, \dots, R_m, i_0), \text{ where}$$

- O is an alphabet (we refer to the elements of O as *objects*),
- $\mu = (\underline{m}, E)$ is a tree with root 0, called the *membrane structure*,
- for every $i \in \underline{m}$, $w_i : O \to \mathbb{N}$ is a map associating objects with their multiplicity in the membrane i,
- for every $i \in \underline{m}$, also associated with the membrane i, R_i is a finite set of *evolution rules* over O. An evolution rule is of the form $u \to v$, where $u \in O^*$, $v \in (O \times \mathrm{TAR})^*$, and $\mathrm{TAR} := \{\mathrm{here}, \mathrm{out}\} \cup \{\mathrm{in}_j \mid j \in \underline{m}\}$, and
- $i_0 \in \underline{m}$ is a leaf in μ, called the *output membrane.*

The application of a rule consumes an object in a membrane i and adds objects to i and its neighboring membranes (i.e., the membranes that are adjacent in μ). In each step, the rules in every membrane i and the objects in c_i they consume are chosen non-deterministically, and in a maximally parallel manner, i.e., such that no further rule in R_i is applicable. Furthermore, all membranes evolve simultaneously, thus leading to a double parallelism, both in the application of rules in each membrane, and in the evolution of membranes itself. As result of the computation we take the symbols in the output membrane when the system has reached a state where no rule is applicable, i.e., when it has *halted.* $\square$

3. Spiking Neural P systems

3.1. The basic model

Spiking Neural P systems are inspired by the workings of a special kind of cell, the neuron that makes up the nervous system. We base our overview of the particular structure of these cells on [Fri09] and [IPY06]. The task of neurons is twofold: To respond to stimuli with electrical discharges, the *nerve spikes*, and to conduct these spikes over long distances. A neuron consists of the cell itself (called the *soma*), the *axon*, which ends in several *nerve terminals* connecting to other cells, and the *dendrites*, a filamentous structure the nerve terminals of other neurons connect to. The points of these connections are called *synapses*, and it is through these synapses that neurons exchange information.

The axon is the part of the neuron responsible for conducting the nerve spikes to other cells. These spikes are short electrical pulses, typically between one or two milliseconds in length, and with an amplitude of about 100 mV. These pulses, while indistinguishable in form, are well-separated, and it is impossible for a neuron to spike again during (or even shortly after) a spike. Hence, information is not encoded in the form of the spike itself, but in the number of spikes emitted and the interval between these spikes (called the *spike train*).

The most common type of synapse is the chemical synapse, where the receiving of a spike triggers the release of a *neurotransmitter* that passes through the membrane into the target neuron, and is then again converted into electrical energy.

Definition 3.1 (Spiking Neural P system, [IPY06])
A *Spiking Neural P system* (or SN P system) of degree $m \in \mathbb{N}_{>0}$ is a tuple

$$\Pi = (O, \sigma_1, \sigma_2, \dots, \sigma_m, \mathrm{syn}, i_0),$$

where

- $O = \{a\}$ is a singleton alphabet (we say that a is the *spike*),
- for each $i \in \underline{m}$, we have $\sigma_i := (n_i, R_i)$ such that
  - $n_i \in \mathbb{N}$ is the initial amount of spikes,
  - $R_i := S_i \cup F_i$ is a finite set of rules, where
    - $S_i \subseteq (\mathrm{RegEx}(O) \times O^+) \times (O \times \mathbb{N})$ (we will write these rules in the form $(E, a^r) \to (a, d)$),
    - $F_i \subseteq O^+ \times \{\lambda\}$ (we will write those as $a^r \to \lambda$) such that $\pi_1[F_i] \cap (\mathcal{L} \circ \pi_1 \circ \pi_1)[S_i] = \varnothing$, i.e., no a^s that appears in the left-hand side of a rule in F_i is in the language matched by a regular expression from the left-hand side of a rule in S_i,
- $\mathrm{syn} \subseteq \underline{m}^2$ is an irreflexive relation (i.e., $\forall i \in \underline{m}.\ (i, i) \notin \mathrm{syn}$), and
- $i_0 \in \underline{m}$ denotes the *output neuron* of Π.

We say that $\sigma_1, \sigma_2, \dots, \sigma_m$ are the *neurons* of Π (for brevity, we may also refer to a neuron by its index alone), whereas syn models the *synapses* connecting the neurons. We refer to the rules in S_i as *spiking rules* and to the rules in F_i as *forgetting rules*. We call a spiking rule $(E, a^r) \to (a, d)$ a *delayed rule* if $d > 0$.

We may even further abbreviate a spiking rule $(E, a^r) \to (a, d)$, writing $a^r \to (a, d)$ if $\mathcal{L}(E) = \{a^r\}$, $(E, a^r) \to a$ if $d = 0$, and $a^r \to a$ if both apply.

A *configuration* of Π is a tuple

$$C_\Pi := \big((s_1, t_1), (s_2, t_2), \dots, (s_m, t_m)\big),$$

where for any $i \in \underline{m}$,

- $s_i \in \mathbb{N}$ is the number of spikes present in the neuron i, and
- $t_i \in \mathbb{N} \cup \{\infty\}$ is the number of steps that must elapse before the neuron i spikes (we take ∞ to mean that the neuron will not fire unless a firing rule is applied), where we set $n < \infty$ for all $n \in \mathbb{N}$.

In any given neuron i, a spiking rule $(E, a^r) \to (a, d)$ is *applicable* if $s_i \geqslant r$, $a^{s_i} \in \mathcal{L}(E)$, and $t_i \in \{0, \infty\}$. A forgetting rule $a^r \to \lambda$ is *applicable* if $s_i = r$. If $t_i \notin \{0, \infty\}$, we say that the neuron i is *closed*, otherwise we say that it is *open*. We say that the neuron σ_i *spikes* in configuration C_Π if either $t_i = 0$ or $t_i = \infty$ and a rule $(E, a^r) \to (a, 0) \in R_i$ is applied in σ_i.

An *initial configuration* of Π is a configuration

$$C_\Pi^0 := \big((n_1, \infty), (n_2, \infty), \dots, (n_m, \infty)\big).$$

Given two configurations $C_\Pi := \big((s_1, t_1), (s_2, t_2), \dots, (s_m, t_m)\big)$, $C'_\Pi := \big((s'_1, t'_1), (s'_2, t'_2), \dots, (s'_m, t'_m)\big)$, we say that Π makes a transition from C_Π to C'_Π and write $C_\Pi \Longrightarrow C'_\Pi$ if for any $i \in \underline{m}$ we have that either

- $0 < t_i < \infty$, $s'_i = s_i$, $t'_i = t_i - 1$, and no rule in R_i is applicable,
- $t_i \in \{0, \infty\}$, $t'_i = \infty$, $s'_i = s_i + \text{inbound}(i)$, and no rule in R_i is applicable,
- $0 < t_i < \infty$, $t'_i = t_i - 1$, $s'_i = 0$, and $a^{s_i} \to \lambda \in R_i$ is applied,
- $t_i \in \{0, \infty\}$, $t_{i'} = \infty$, $s'_i = \text{inbound}(i)$, and $a^{s_i} \to \lambda \in R_i$ is applied,
- $t_i \in \{0, \infty\}$, $t'_i = d \in \mathbb{N}_{>0}$, $s'_i = s_i - r$, and $(E, a^r) \to (a, d) \in R_i$ is applied, or
- $t_i \in \{0, \infty\}$, $t'_i = \infty$, $s'_i = s_i - r + \text{inbound}(i)$, and $(E, a^r) \to (a, 0) \in R_i$ is applied,

where

$$\text{inbound}(i) := \big|\{j \in \underline{m} \mid (j, i) \in \text{syn} \text{ and } \sigma_j \text{ spikes}\}\big|$$

denotes the number of spikes received by the neuron i.

Given a family of configurations $\big(C_\Pi^i\big)_{i \in I}$, where either $I = \{0\} \cup \underline{k}$ for some $k \in \mathbb{N}$, or $I = \mathbb{N}$, C_Π^0 is the initial configuration, and

$$C_\Pi^{i-1} \Longrightarrow C_\Pi^i$$

for any $i \in I \backslash \{0\}$, we say that a strictly increasing sequence $i_1 < i_2 < \cdots \in I$ is a *spike train* of Π if $C_\Pi^{i_1}, C_\Pi^{i_2}, \dots$ are exactly the configurations in $\big(C_\Pi^i\big)_{i \in I}$ in which the output neuron σ_{i_0} spikes.

We denote by

$$\mathcal{N}_2(\Pi) := \left\{ n \in \mathbb{N}_{>0} \,\middle|\, \exists C^0_\Pi \Longrightarrow C^1_\Pi \Longrightarrow \cdots \Longrightarrow C^{k+n}_\Pi \text{ such that } k, k+n \text{ is the associated spike train} \right\}$$

the *language generated* by Π. By $SN_2P_m\left(\mathrm{rule}_k, \mathrm{cons}_p, \mathrm{forg}_q\right)$ we denote the family of languages generated by Spiking Neural P systems in the above fashion, where

- at most $m \geqslant 1$ neurons are used,
- each neuron contains at most $k \geqslant 1$ rules,
- for any rule $(E, a^r) \to (a, d)$, we have $r \leqslant p$, and
- for any rule $a^r \to \lambda$, we have $r \leqslant q$.

For any of these parameters, we write $*$ to denote that it is unbounded. As is common, we may omit the index Π in the configurations. □

The subscript 2 in the context of generated languages refers to taking the number of steps between the two spikes of the output neuron as the generated number. As [PPR06] notes, there are other possible results associated with a given spike train, e.g., we could take the distances between the first and the second spike, and between the second and third spike as the pair of numbers generated by the system. In the notation of [PPR06], this would be denoted by the subscript 3. Another important result mode (especially for asynchronous systems) is to simply take the total number of spikes emitted by a certain neuron.

Definition 3.2 (Total result mode)
Let $\Pi = (O, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, i_0)$ be an SN P system, and let C_Π be a configuration of Π. We say that C_Π is a *halting configuration* if there is no neuron in Π that contains an applicable rule.

We say that the language generated by Π in the total result mode is

$$\mathcal{N}_{\mathrm{tot}}(\Pi) := \left\{ |I| \,\middle|\, C^0_\Pi \Longrightarrow C^1_\Pi \Longrightarrow \cdots \Longrightarrow C^n_\Pi \text{ such that } C^n_\Pi \text{ is a halting configuration with spike train } I \right\}.$$ □

Remark 3.3
While [IPY06] explicitly states that for rules of the form $(E, a^r) \to (a, 0)$ "no restriction is imposed, the neuron can receive spikes in the same step when using the rule" (and this is again corroborated by [PP09]), the formalization in [Fri09, p. 161] does not allow that. However, [Fri09, Example 7.1] features exactly such a situation. In fact, the formalization is contradicting itself even for the situation of only two neurons, where one neuron spikes and the receiving neuron applies a forgetting rule: On one hand, in the resulting configuration, the receiving neuron should contain strictly more spikes than before, while on the other hand, it should contain less spikes. $\square$

Proposition 3.4 ([IPY06, Theorem 7.1])
For any $k \geqslant 2$, $p \geqslant 3$, and $q \geqslant 3$, we have

$$SN_2P_* \left(\text{rule}_k, \text{cons}_p, \text{forg}_q\right) = \mathcal{N}RE\,. \qquad \square$$

Note that this equality holds only if we take the λ-convention (Remark 2.20) into account, as clearly the smallest number generated by any SN P system in this fashion is 1.

A common extension is to allow for spiking rules to produce more than one spike.

Definition 3.5 (SN P system with extended rules, [PP09])
A Spiking Neural P system with extended rules is a Spiking Neural P system where we also allow spiking rules to be of the form $(E, a^r) \to (a^p, d)$ for $r \geqslant p \geqslant 1$. When one of these rules spikes in neuron i, the neurons $\left\{\sigma_j \mid (i, j) \in \text{syn}\right\}$ each receive not one, but p spikes. $\square$

When working with extended rules in the total result mode, we must take care to ensure that if the output neuron emits multiple spikes in a single step, we take all of these into account.

Definition 3.6 (Types of neurons, [Cav+09])
Consider a neuron $\sigma = (n, R)$. We say that a rule $r \in R$ is *bounded* if it is of the form $(a^i, a^j) \to (a^p, d)$ (for some $1 \leqslant i \leqslant j$, $p \in \mathbb{N}_{>0}$, $d \in \mathbb{N}$) or of the form $a^k \to \lambda$ for some $k \geqslant 1$. We say that r is *unbounded* if it is of the form $\left(a^c \left(a^i\right)^*, a^j\right) \to (a^p, d)$ for some $c, d \in \mathbb{N}$ and $i, j, p \in \mathbb{N}_{>0}$.

We say that σ is *unbounded* if all rules in R are unbounded, and *bounded* if all rules in R are bounded. If R contains both bounded and unbounded neurons, we say that σ is a *general* neuron.

Considering an SN P system Π, we refer to Π as *bounded* if every neuron is bounded, while we say that Π is *unbounded* if it contains an unbounded neuron. If Π has a general neuron, we say that Π is *general*. □

3.2. Spiking Neural P systems with cooperating rules

[MRK14b] introduces the concept of cooperating rules in the fashion of CD grammar systems to the model of Spiking Neural P systems.

Definition 3.7 (SN P system with cooperating rules, [MRK14b]) A *Spiking Neural P system with cooperating rules* of degree $m \in \mathbb{N}_{>0}$ and with $p \in \mathbb{N}_{>0}$ components is a tuple

$$\Pi = (O, \sigma_1, \sigma_2, \dots, \sigma_m, syn, i_0),$$

where

- for each $i \in \underline{m}$, the *neuron* σ_i is of the form $(n_i, R_{i,1}, R_{i,2}, \dots, R_{i,p})$, and
- for every $c \in \underline{p}$, we have that

  $$\pi_c(\Pi) := \big(O, (n_1, R_{1,c}), (n_2, R_{2,c}), \dots, (n_m, R_{m,c}), syn, i_0\big)$$

  is a Spiking Neural P system with extended rules. We say that $\pi_c(\Pi)$ is the *projection* onto the c-th component of Π.

A *configuration* of Π is a tuple

$$C_\Pi := \Big(c, \big(s_1, t_1^1, t_1^2, \dots, t_1^p\big), \big(s_2, t_2^1, t_2^2, \dots, t_2^p\big), \dots, \big(s_m, t_m^1, t_m^2, \dots, t_m^p\big)\Big),$$

where

- $c \in \underline{p}$ is the *active component*, and
- for each $c \in \underline{p}$, $\pi_c(C_\Pi) := \big((s_1, t_1^c), (s_2, t_2^c), \dots, (s_m, t_m^c)\big)$ is a configuration of $\pi_c(\Pi)$.

Recall the set

$$\mathcal{D} = \{t, *\} \cup \bigcup_{k\geqslant 1}\{= k, \geqslant k, \leqslant k\}$$

of *cooperation protocols* we have defined for CD grammar systems (Definition 2.53). In a similar fashion, we allow transitions between two configurations C_Π and C'_Π if that transition is a transition in $\pi_c(\Pi)$ for a component $c \in \underline{p}$ (either Π stays in component c or control transfers from component c' to c, and this happens in accordance with the cooperation protocol).

Let $d \in \mathcal{D}$ be a cooperation protocol, and let $\beta \in \{\text{gene}, \text{unb}, \text{boun}\}$. We denote by $C_2\mathcal{N}_d^{\text{maxpar}}(\Pi)$ the *language generated* by Π working according to the d protocol, and by $\mathcal{N}C_pSN_2P_d^{\text{maxpar}}(\beta)$ the *family of languages generated* by a p-component general, unbounded, or bounded, respectively, Spiking Neural P systems with cooperating rules and working according to the d protocol. □

Note that, unlike [MRK14b], we reserve the third subscript for the cooperation protocol (in [MRK14b], it denotes an upper bound on the number of neurons). This is due to the fact that we study various cooperation protocols, while [MRK14b] is concerned only with the terminating protocol, and that we do not impose any bounds on the number of neurons (in fact, neither does [MRK14b]).

Furthermore, the definition in [MRK14b] only allows for the basic rules without any delayed rules, i.e., all spiking rules are of the form $(E, a^r) \rightarrow (a, 0)$. While that is sufficient to prove universality for the terminating protocol, we need the extended rules to retain universality when working in the other protocols.

Definition 3.8 (Multiple output neurons, [MRK14b])
An obvious generalization is to consider systems with more than one output neuron. The resulting systems generate tuples of numbers, and we designate the generated language $C_p\,\Psi\,P_d^{\text{maxpar}}(\Pi)$. Similarly, we denote by $\Psi\,C_pSN_2P_d^{\text{maxpar}}(\beta)$ the family of languages generated by systems of appropriate type. □

Instead of the usual *maximally parallel* mode of operation, [MRK14b] also consider Spiking Neural P systems with cooperating rules working in an asynchronous or a strongly sequential mode.

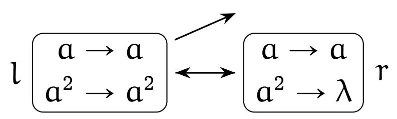

Figure 3.1.: An SN P system exhibiting different behavior in the maximally parallel and strongly sequential modes

Definition 3.9 (Strongly sequential mode, [MRK14b])
In the *strongly sequential mode* of operation, exactly one rule is applied in each transition. This rule is chosen non-deterministically among the applicable rules throughout the system. We denote the strongly sequential mode by sseq. □

A Spiking Neural P system may behave differently in the strongly sequential mode than in the maximally parallel mode. Consider the system in fig. 3.1, where initially both neurons contain a single spike. In the strongly sequential mode, the computation will stop after either one or two spikes have been emitted from the neuron l. In the maximally parallel mode, however, the system never reaches a halting configuration, and a spike is sent to the environment in every step.

Also note that even if a system always were to eventually reach the same configurations in the strongly sequential mode as in the maximally parallel mode, the generated languages might still differ as the spike train might be sensitive to timing.

3.2.1. The terminating protocol

For the terminating protocol, [MRK14b] show universality by simulating register machines. Since we build further results on this construction, we present the complete proof here. There are also examples of small (in the number of neurons) universal systems: [SP14] gives an example of a universal Spiking Neural P system with cooperating rules using 8 neurons, whereas [MRK14a] has a system with 59 neurons that computes numerical functions. Both of these universal systems work in the terminating protocol.

Theorem 3.10 ([MRK14b])

$$\mathcal{N}C_2SN_2P_t^{sseq}(\text{unb}) = \mathcal{N}RE, \textit{ and}$$
$$\Psi\, C_2SN_2P_t^{sseq}(\text{unb}) = \Psi RE\,. \qquad \square$$

Proof (Theorem 3.10, [MRK14b]) Let $L \in \mathcal{N}RE$. Then we know by Theorem 2.50 that there is a (non-deterministic) register machine $\mathcal{M} = (m, H, l_0, l_h, I)$ with m registers generating L, i.e.,

$$\mathcal{N}(\mathcal{M}) = L.$$

Without loss of generality, we assume that the output register 1 is never decremented, but only incremented throughout the computation (we can always achieve this by adding an extra register). We let $\phi : H \to I$ be the isomorphism mapping instruction labels to instructions.

We construct a strongly sequential SN P system Π with cooperating rules and unbounded neurons simulating $\mathcal{M}$, comprising of two components, working in the strongly sequential mode and according to the terminating protocol. For each of the three instruction types ADD, SUB, HALT of register machines, we describe an SN P system with cooperating rules simulating exactly this instruction. We may then compose these modules into the system simulating $\mathcal{M}$ by having one instance of the respective module for each of the instructions of $\mathcal{M}$. Some neurons may appear in more than one module, these are then shared among all modules containing them, thus connecting the modules into a larger system.

Specifically, for each register $r \in \underline{m}$, we have a neuron σ_r that stores the contents of register r throughout the computation: If register r contains the number $n \in \mathbb{N}$, neuron σ_r will contain exactly $2n$ spikes. Furthermore, for each instruction label $i \in H$, we have a neuron σ_{l_i} with a single rule $a \to a$ in the first component. We use these neurons to encode the instruction that is currently being simulated: In each step, at most one of the neurons $\{\sigma_{l_i} \mid i \in H\}$ will contain a spike, indicating that the instruction $\phi(i)$ is to be simulated. When the simulation of this instruction is complete, a spike is sent to the neuron $\sigma_{l_{i'}}$ corresponding to the next instruction label $i' \in H$ (unless, of course, $i = l_h$ is the halting label). Finally, for each instruction, we have some auxiliary neurons $\sigma_{i,j}$ ($j \in \mathbb{N}_{>0}$). Initially, only the neuron $\sigma_{l_{l_0}}$ (associated to the starting instruction l_0 of $\mathcal{M}$) contains a single spike and all other neurons remain empty.

To simulate an instruction $i \in H$ with $\phi(i) = \big(\mathrm{ADD}(r), l_j, l_k\big)$, we use the module depicted in fig. 3.2. Assume that σ_{l_i} contains a single spike, and that the only other non-empty neurons are of the form σ_r for some $r \in \underline{m}$. Then the only applicable rule in the system is the one in σ_{l_i}, and control transfers to the first component (if the first component is not already active). Thus the rule in σ_{l_i} will fire, and $\sigma_r, \sigma_{i,1}$, and $\sigma_{i,2}$ each receive a spike. Neither σ_r nor $\sigma_{i,2}$ can fire, however, because they only contain rules in the second component. Hence $\sigma_{i,1}$ will fire, and σ_r and $\sigma_{i,2}$ receive a second spike. Since now no rules in the first component are applicable, control transfers to the second component. Note that σ_r now contains two more spikes, which corresponds to incrementing the register r by one.

Now $\sigma_{i,2}$ has two applicable rules, $(a^2, a) \to a$ and $a^2 \to a$, one of which is non-deterministically selected and fired. In either case, $\sigma_{i,3}$ and $\sigma_{i,4}$ each receive a spike. If the first rule is fired, control remains in the first component as the $a \to a$ rule in $\sigma_{i,2}$ is still applicable and will fire, sending another spike to $\sigma_{i,3}$ and $\sigma_{i,4}$, and control transfers to the second component. Otherwise, control transfers to the second component immediately. Now $\sigma_{i,3}$ will forget its two spikes and $\sigma_{i,4}$ will emit a spike, or $\sigma_{i,3}$ will send out a spike and $\sigma_{i,4}$ will forget its spike, respectively. Control then transfers back to the second component, and either $\sigma_{i,5}$ or $\sigma_{i,6}$ will send a spike to $\sigma_{l_{l_j}}$ or $\sigma_{l_{l_k}}$, respectively.

We use the module depicted in fig. 3.3 to simulate an instruction l with $\phi(l) = (\mathrm{SUB}(r), l_j, l_k)$. Again, the only non-empty neurons are σ_{l_i} and possible $\sigma_{r'}$ for some register $r' \in \underline{m}$, and the only applicable rule is the one in the first component of σ_{l_i}, so control transfers to the first component. σ_{l_i} spikes, and the neurons $\sigma_{i,1}$, $\sigma_{i,2}$, and σ_r receive one spike each. Control then transfers to the second component, and $\sigma_{i,1}$, $\sigma_{i,2}$, and, possibly, σ_r spike, in no particular order. Note that the rule in σ_r is only applicable if it contains at least three spikes, i.e., only if it contained at least two spikes prior to receiving the spike from σ_{l_i}. Thus, σ_r will spike iff register r is non-empty.

For now, assume that σ_r does indeed spike. Then $\sigma_{i,3}$ and $\sigma_{i,4}$ each receive three spikes, so $\sigma_{i,3}$ will spike and $\sigma_{i,4}$ will forget these, and control transfers to the first component. Note that σ_r will send its spike not only to $\sigma_{i,3}$ and $\sigma_{i,4}$, but also to each $\sigma_{i',3}$ and $\sigma_{i',4}$ where $i' \in H$ is such that $\pi_1\big(\phi(i')\big) = \mathrm{SUB}(r)$, and we must ensure that these spikes are forgotten before the simulation proceeds. The $a \to \lambda$ rules in the first components of $\sigma_{i',3}$ and $\sigma_{i',4}$ accomplish just that. Also, $\sigma_{i,5}$ will send its spike to $\sigma_{i,6}$. Then, control transfers to the second component, $\sigma_{i,6}$ spikes, and $\sigma_{l_{l_j}}$ receives a spike.

If, on the other hand, σ_r does not spike, i.e., if the register r is empty, $\sigma_{i,3}$ and $\sigma_{i,4}$ each receive only two spikes. Control then transfers to the first component, $\sigma_{i,3}$ forgets its spikes, and $\sigma_{i,4}$ spikes, so $\sigma_{i,7}$ and σ_r each receive a spike (σ_r now contains two spikes). Then $\sigma_{i,7}$ spikes, both $\sigma_{i,8}$ and σ_r receive a spike, and control transfers to the second component. Now σ_r and $\sigma_{i,8}$ will both spike, so σ_r is empty again. As above, all neurons $\sigma_{i',3}$ and $\sigma_{i',4}$ for $i' \in H$ with $\pi_1\big(\phi(i')\big) = \mathrm{SUB}(r)$ will now forget their spikes, and $\sigma_{i,9}$ receives a spike, which is subsequently passed through $\sigma_{i,10}$ to $\sigma_{l_{l_k}}$.

We simulate the HALT instruction by the module shown in fig. 3.4. When $\sigma_{l_{l_h}}$ spikes (again, control must transfer to the first component because no other rules are applicable), both σ_1 and σ_{out} receive a spike, and σ_{out} will spike for the first time. Control then transfers to the first component. Assume that σ_1 now contains $2n+1$ spikes (for some $n \in \mathcal{N}(\mathcal{M})\setminus\{0\}$). Then, for the next n steps, two spikes are removed from σ_1, and σ_{out} receives a spike. Then control transfers to the first component, and σ_{out} spikes for the second time, producing a spike train of the form $k, k+n$ for some $k \in \mathbb{N}_{>0}$, hence we have $n \in C_2\mathcal{N}_t^{sseq}(\Pi)$.

Otherwise, if σ_1 contains only a single spike, no rule is applicable and the computation stops without producing a two-element spike train.

Clearly, we can apply the same approach to a register machine with multiple output registers generating some $L' \in \Psi\mathrm{RE}$. ■

Observe that the only part of the construction that is sensitive to timing is the output module. As only one rule is applicable at any time, the module behaves the same when run in the maximally parallel mode. Also notice that there are no interactions between neurons that depend on the strongly sequential mode (there is no interaction between $\sigma_{i,4}$ and σ_r be-

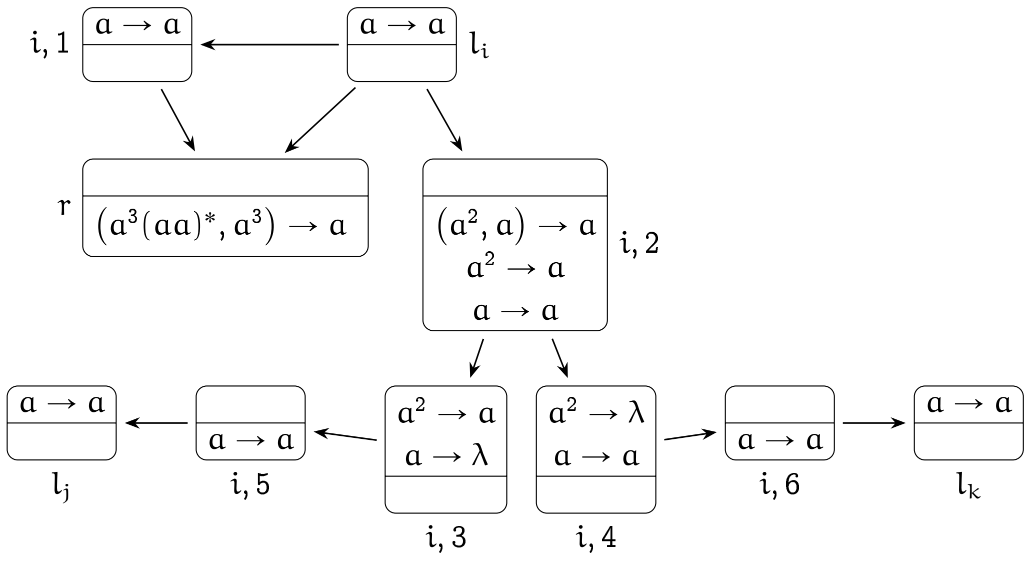

Figure 3.2.: The ADD module for the strongly sequential case

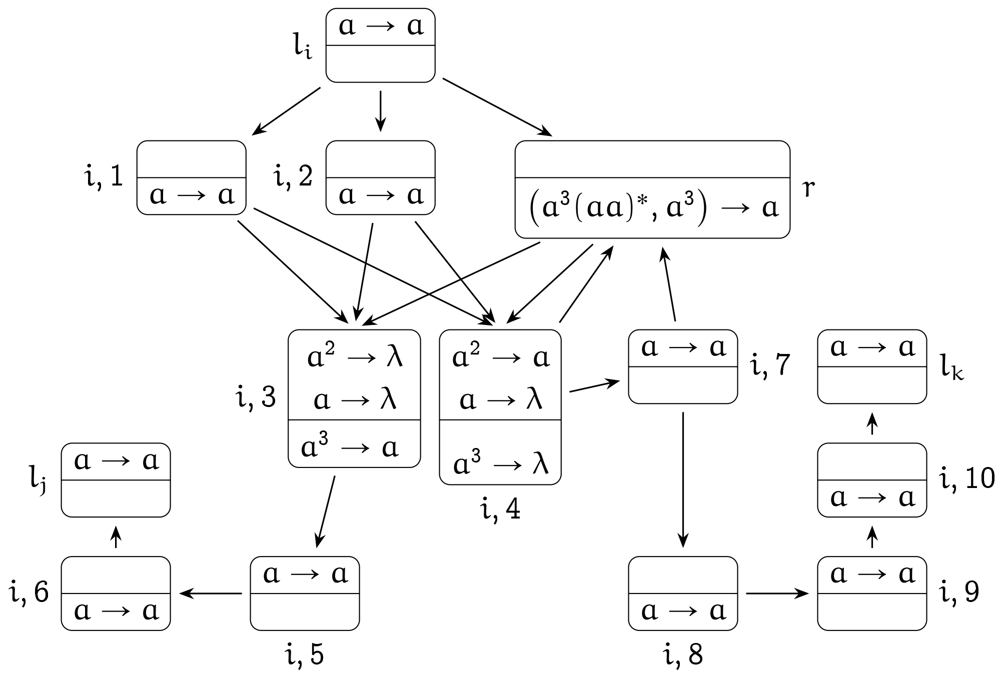

Figure 3.3.: The SUB module for the strongly-sequential case

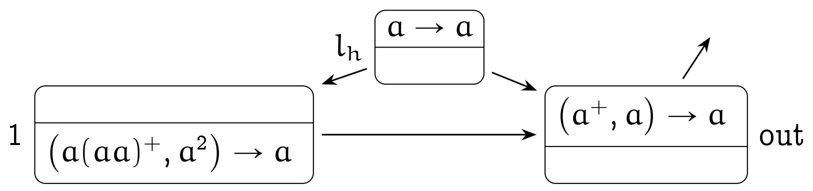

Figure 3.4.: The output module for the strongly sequential case

cause the rules are in different components). Hence, the system generates the same output, and we obtain the following corollary.

Corollary 3.11

$$\mathcal{N}C_2SN_2P_t^{maxpar}(unb) = \mathcal{N}RE, \textit{ and}$$
$$\Psi\, C_2SN_2P_t^{maxpar}(unb) = \Psi RE\,. \qquad \square$$

3.2.2. The arbitrary protocol

The terminating protocol allows us to delay the spiking of a neuron until no more rules are applicable in one component, and this is the basic idea behind the output module as used in the proof of Theorem 3.10. Switching to the arbitrary protocol, we lose that feature, and clearly we have to make up for this loss by using more powerful features for the remaining system (note how this parallels the fact that the terminating mode is more powerful for CD grammar systems as well, cf. Proposition 2.54). It turns out that using the maximally parallel mode instead of the strongly sequential mode, allowing for general neurons, and using extended rules is enough to recover universality.

Theorem 3.12

$$\mathcal{N}C_2SN_2P_*^{maxpar}(gene) = \mathcal{N}RE, \textit{ and}$$
$$\Psi\, C_2SN_2P_*^{maxpar}(gene) = \Psi RE\,. \qquad \square$$

PROOF (THEOREM 3.12) The ADD module almost works when using the arbitrary protocol in the maximally parallel mode. Consider, however, control transferring to the second component after σ_{l_i} has spiked. Then the neuron σ_r contains $2n + 1$ spikes (for some $n \in \mathbb{N}$) and hence its only rule is applicable, leaving the neuron with $2(n - 1)$ spikes. We can, however, easily fix this by introducing a delay neuron between σ_{l_i} and σ_r that contains a single $a \to a$ rule in its first component. Since we are working in the maximally parallel mode, this ensures that σ_r receives both spikes simultaneously, and the rule in its second component never becomes applicable. Similarly, we need another delay neuron between σ_{l_i} and $\sigma_{i,2}$ that ensures both spikes are received simultaneously in $\sigma_{i,3}$. Finally, we replace the rules $a \to a$ and $(a^2, a) \to a$ in the former $\sigma_{i,2}$ neuron by the extended rule $a^2 \to a^2$.

The SUB module works as before when using the arbitrary protocol and the maximally parallel mode, because between every transfer of control we have at least one rule firing in the newly active component, and due to the maximal parallelism, every applicable rule must fire. However, there is no neuron where a rule would be applicable if a rule in the same component were fired in the previous step.

The output module, however, needs to be adapted. Otherwise, control might transfer to component one while the neuron corresponding to the output register still contains three or more spikes (i.e., while the rule in the second component is still applicable), and the output neuron would spike prematurely.

Indeed, we modify the output neuron to contain a rule $a \to a$ in both components, and the neuron corresponding to the output register to have the two rules $\left(a^3(aa)^+, a^2\right) \to \lambda$ and $a^3 \to a$ in both components (note that this is now a general neuron). Assume that, after receiving the spike from $\sigma_{l_{l_h}}$, σ_1 contains $2n + 1$ spikes. When σ_{out} now spikes for the first time, σ_1 will forget two spikes in the same step (since we are now working in the maximally parallel mode), and thus contain $2n - 1$ neurons. During each of the next $n - 2$ steps, σ_1 forgets two spikes. Then, the last three spikes are removed and a spike is sent to σ_{out}, which spikes in the next step for the second time, exactly n steps after the first spike.

If σ_1 contains only a single spike, no rule is applicable, and the computation stops without producing a result. ■

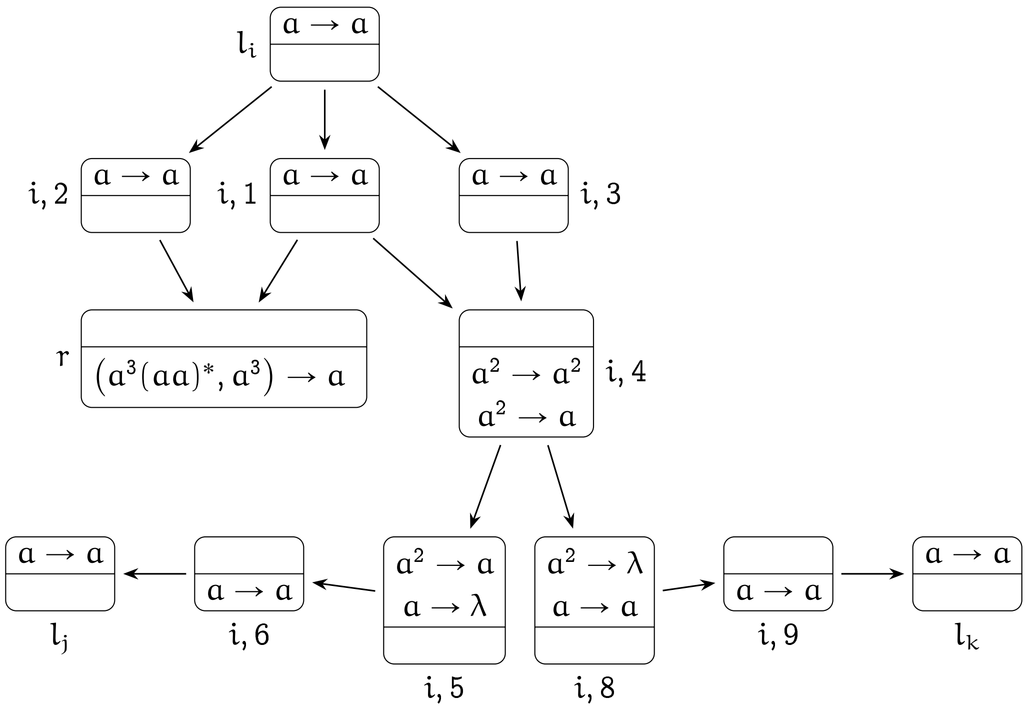

Figure 3.5.: The ADD module for the maximally parallel case

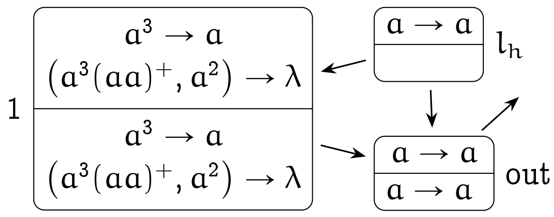
1
$a^3 \to a$
$(a^3(aa)^+, a^2) \to \lambda$
$a^3 \to a$
$(a^3(aa)^+, a^2) \to \lambda$
$a \to a$
l_h
$a \to a$
$a \to a$
out

Figure 3.6.: Output module for the maximally parallel case

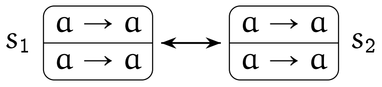

Figure 3.7.: Stepper module for the maximally parallel case

3.2.3. The exactly-k-steps protocol

Theorem 3.13

Let $k \in \mathbb{N}_{>0}$. *Then we have*

$$\mathcal{N}C_2SN_2P^{maxpar}_{=k}(\text{gene}) = \mathcal{N}RE, \quad \textit{and}$$
$$\Psi\, C_2SN_2P^{maxpar}_{=k}(\text{gene}) = \Psi RE\,. \qquad \square$$

Proof (Theorem 3.13) Consider again the construction from the proof of Theorem 3.12. While this works in the presence of arbitrary transfers of control between components, the problem is that when transferring control exactly every k steps, there might not be an applicable rule in the newly activated component and the computation will halt. We can, however, easily extend the construction by adding two neurons that continually exchange a single spike among them. This ensures that there is always an applicable rule. Hence, computation will not stop prematurely (in fact, it never will, but that is not a problem since we are only interested in the spike trains of the output neurons). Figure 3.7 shows such a "stepper" module. ■

3.2.4. The other protocols

As an immediate corollary to Theorem 3.13, we obtain universality for the remaining two cooperation protocols, as both "at least k steps" and "at most k steps" are subsumed by "exactly k steps."

Corollary 3.14

Let $k \in \mathbb{N}_{>0}$. *Then we have*

$$\mathcal{N}C_2SN_2P^{maxpar}_{\leqslant k}(\text{gene}) = \mathcal{N}RE,$$
$$\mathcal{N}C_2SN_2P^{maxpar}_{\geqslant k}(\text{gene}) = \mathcal{N}RE,$$
$$\Psi\, C_2SN_2P^{maxpar}_{\leqslant k}(\text{gene}) = \Psi RE, \quad \textit{and}$$
$$\Psi\, C_2SN_2P^{maxpar}_{\geqslant k}(\text{gene}) = \Psi RE\,. \qquad \square$$

3.2.5. The asynchronous case

Instead of using the maximally parallel or the strongly sequential mode, we may also consider asynchronous systems. Introduced by [Cav+09], these systems feature an additional source of non-determinism in that an applicable rule must not necessarily fire.

Definition 3.15 (Asynchronous mode, [Cav+09])
In the *asynchronous mode* of operation, at most one rule is applied in each neuron. While only applicable rules are allowed to be fired, a rule need not be fired even if it is the only applicable rule in the neuron. Since the timing between a neuron spiking is now sensitive to when rules are applied, we use the total result mode for asynchronous SN P systems. We denote the asynchronous mode by async. □

Theorem 3.16 ([MRK14b])

$$\begin{aligned} &\mathcal{N}C_2SN_{tot}P_t^{async}(\text{gene}) = \mathcal{N}RE, \ \textit{and} \\ &\Psi\, C_2SN_{tot}P_t^{async}(\text{gene}) = \Psi RE\,. \end{aligned}$$ □

Proof (Theorem 3.16) This proof is similar in structure to that of Theorem 3.10. Again, we construct an asynchronous SN P system with cooperating rules working according to the terminating protocol. We do, however, allow for general neurons.

The module for simulating ADD instructions (cf. fig. 3.8) differs from the strongly sequential mode only by the added $a \to \lambda$ rule in the second component of σ_r. Since this rule is never fired through the simulation of an ADD instruction, this does not change the behavior.

To simulate an instruction $i \in H$ with $\phi(i) = (\text{SUB}(r), l_j, l_k)$, we use the module depicted in fig. 3.9. If register r is empty, the single spike received by σ_r will be forgotten using the $a \to \lambda$ rule, and $\sigma_{i,3}$ and $\sigma_{i,4}$ each receive two spikes. Conversely, if r is not empty, then σ_r will spike, and $\sigma_{i,3}$ and $\sigma_{i,4}$ each receive three spikes. From there, the computation proceeds in a straightforward fashion.

The output module becomes simpler, since we are now working in the total result mode. We depict the new module in fig. 3.10. Clearly, σ_{out} emits one spike for each two spikes it contains.

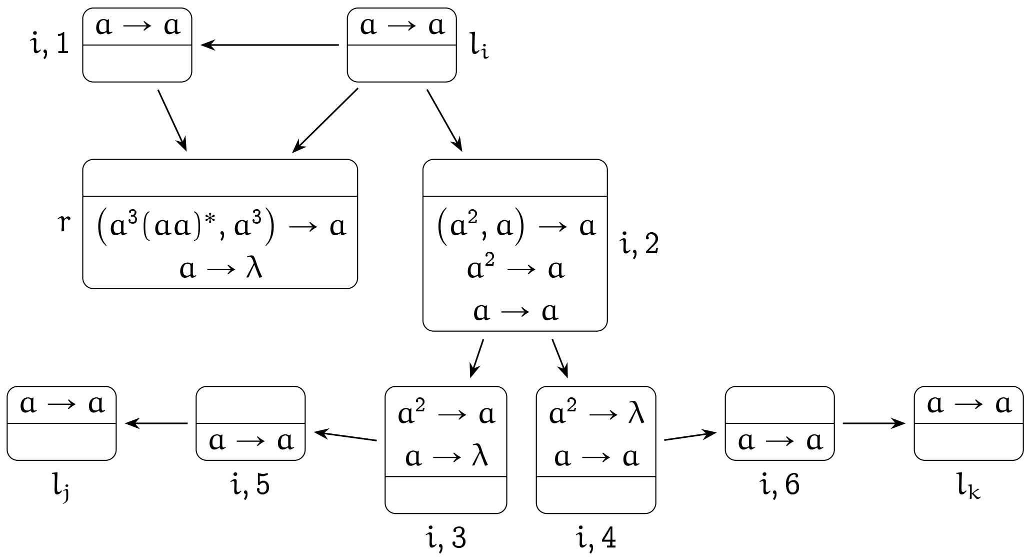

Figure 3.8.: The ADD module for the asynchronous case

Again, we may extend the approach to systems with multiple output neurons. ■

[SPP13] notes that in biological neural systems, small groups of 4–5 or 12–15 neurons will often work in a synchronous fashion while the system as a whole works in the asynchronous mode. Based on this observation, they introduce the concept of local synchronization, and this concept easily generalizes to the setting of SN P systems with cooperating rules.

Definition 3.17 (Local synchronization, [SPP13])
An asynchronous SN P system with cooperating rules and *local synchronization* is a structure

$$\Pi \coloneqq (O, \sigma_1, \sigma_2, \ldots, \sigma_m, \mathrm{loc}, \mathrm{syn}, i_0),$$

where

- $(O, \sigma_1, \sigma_2, \ldots, \sigma_m, \mathrm{syn}, i_0)$ is an SN P system with cooperating rules, and
- $\mathrm{loc} \subseteq \mathfrak{P}(\{\sigma_i \mid i \in \underline{m}\})$ is the set of *locally synchronous* neurons.

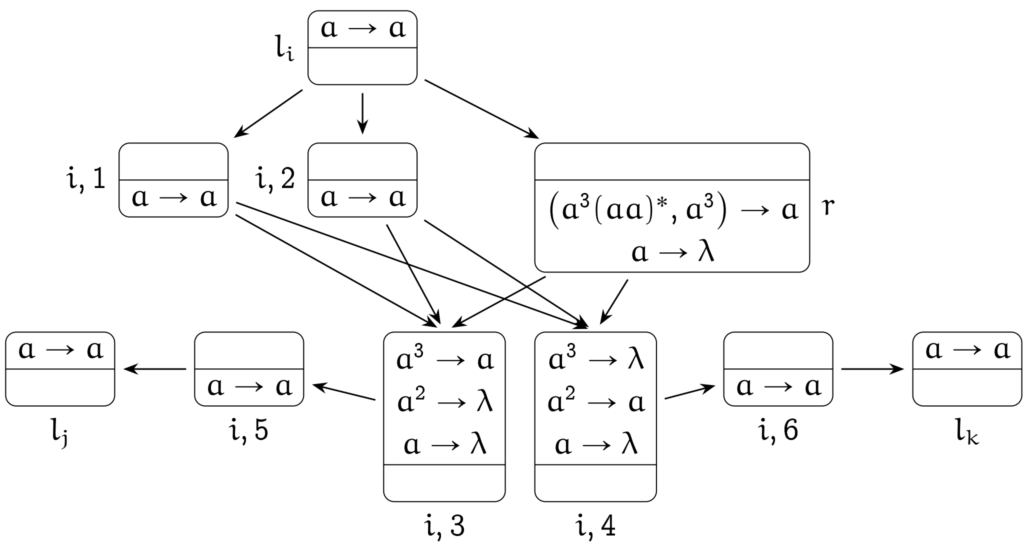

Figure 3.9.: The SUB module for the asynchronous case

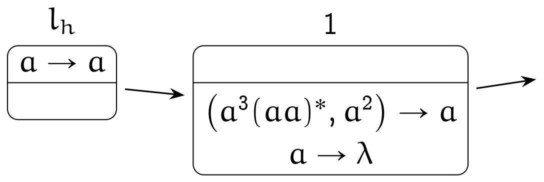

Figure 3.10.: The output module for the asynchronous case

We refer to the elements of loc as *LS-sets*, and denote the use of local synchronization by adding the lsync superscript. □

It turns out that local synchronization suffices to recover the loss of synchronization incurred by switching to the arbitrary cooperation protocol. Hence, we obtain the following theorem.

Theorem 3.18

$$\mathcal{N}C_2SN_{tot}P_*^{lsync}(\text{gene}) = \mathcal{N}RE$$
$$\Psi\, C_2SN_{tot}P_*^{lsync}(\text{gene}) = \Psi RE$$ □

Proof (Theorem 3.18) Consider the ADD module from the proof of Theorem 3.12 enriched with the $a \to \lambda$ rule in the neuron σ_r. We depict such a module in fig. 3.11. Using the LS-sets $\{\sigma_{i,1}, \sigma_{i,2}, \sigma_{i,3}\}$ and $\{\sigma_{i,5}, \sigma_{i,8}\}$, this module works as before.

The SUB module from the proof of Theorem 3.16 (fig. 3.9) works as before when we add the two LS-sets $\{\sigma_{i,1}, \sigma_{i,2}, \sigma_r\}$ and

$$\{\sigma_{i',3}, \sigma_{i',4} \mid i' \in H \text{ is such that } \phi(i') = (\text{SUB}(r), l', l'')\}.$$

Similarly, the output module (fig. 3.10) continues to work. ■

The size of the LS-set for the SUB module depends on the number of SUB instructions decrementing the same register. While the definition of the locally synchronous mode does not impose any bounds on the size of the LS-sets, we feel that it goes against the spirit of local synchronization if LS-sets grow too big.

The universal register machine simulated in [MRK14a] has 14 SUB instructions with at most four instructions sharing a register. Thus, we may conclude that there are universal locally synchronous SN P systems with cooperating rules where, keeping in line with the biological inspiration, no LS-set contains more than 15 neurons.

In a similar fashion, we can adapt the stepper approach to recover universality in the other modes.

Theorem 3.19
Let $k \in \mathbb{N}_{>0}$. *Then we have*

$$\begin{aligned}
\mathcal{N}C_2SN_{tot}P^{lsync}_{=k}(\text{gene}) &= \mathcal{N}RE\\
\Psi\, C_2SN_{tot}P^{lsync}_{=k}(\text{gene}) &= \Psi RE\\
\mathcal{N}C_2SN_{tot}P^{lsync}_{<k}(\text{gene}) &= \mathcal{N}RE\\
\Psi\, C_2SN_{tot}P^{lsync}_{<k}(\text{gene}) &= \Psi RE\\
\mathcal{N}C_2SN_{tot}P^{lsync}_{>k}(\text{gene}) &= \mathcal{N}RE\\
\Psi\, C_2SN_{tot}P^{lsync}_{>k}(\text{gene}) &= \Psi RE
\end{aligned}$$

□

Proof (Theorem 3.19) We extend the construction from the proof of Theorem 3.18 with a stepper module. Since we are now concerned with terminating computations, we need to make sure that the stepper module will not keep the computation running indefinitely. Furthermore, we require both σ_{s_1} and σ_{s_2} to contain a spike in the initial configuration, and that $\{\sigma_{s_1}, \sigma_{s_2}\}$ is an LS-set. Otherwise, only one of s_1 and s_2 could spike, which would lead to the spike being deleted and the computation stopping prematurely. Figure 3.12 depicts the combined output/stepper module. Clearly, this asserts that a rule in every component is applicable until σ_{l_h} spikes. Since σ_1 contains the same rules in both components, the stepper functionality is no longer needed. ■

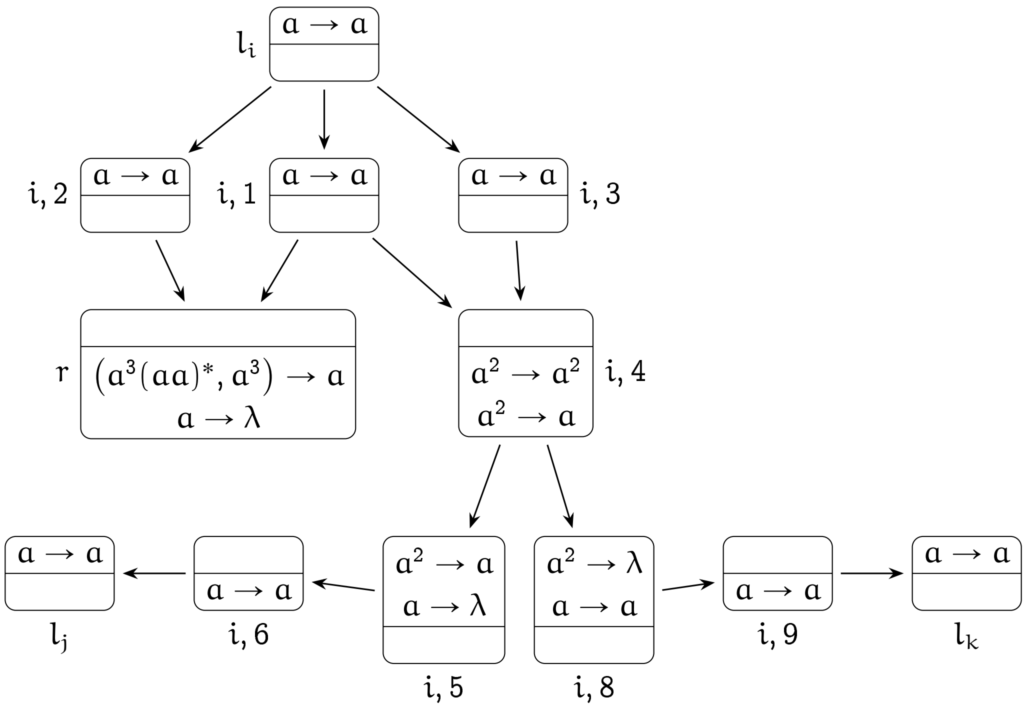

Figure 3.11.: The ADD module for the locally synchronous case

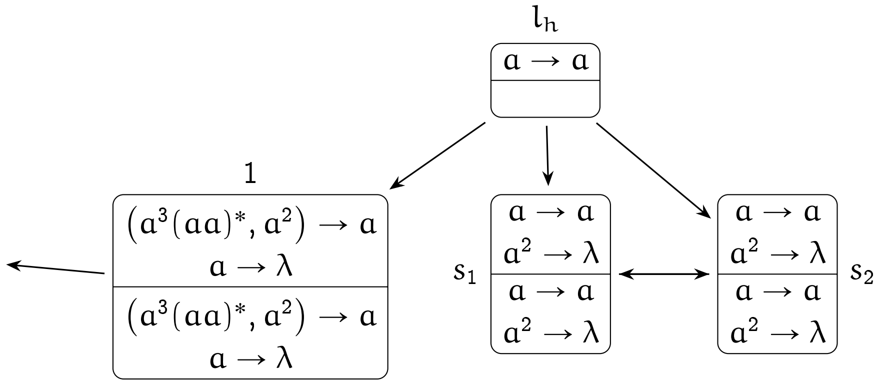

Figure 3.12.: The combined output/stepper module for the locally synchronous case

4. Generating a non-semi-linear set: An example

To demonstrate the usefulness as a modeling tool, we now take a CD grammar system generating a non-context-free language and show how to construct an equivalent SN P system with cooperating rules.

4.1. The set $\{2^n \mid n \in \mathbb{N}\}$

Lemma 4.1
The set

$$P := \{2^n \mid n \in \mathbb{N}\}$$

is not semi-linear. □

Proof (Lemma 4.1) Let $S \subseteq P$ be a linear subset of P, i.e.,

$$S = \{a + ib \mid i \in \mathbb{N}\}$$

for some $a, b \in \mathbb{N}$. Clearly, either $|S| = 1$ or S is infinite. Assume now that S is infinite.

Hence, we have that for any $i \in \mathbb{N}$, there is an $m \in \mathbb{N}$ satisfying

$$a + ib = 2^m.$$

Thus, considering $i = 0$, we obtain $a = 2^{m_a}$ for some $m_a \in \mathbb{N}$. Since $a + ab \in S$, we have $a + ab = 2^{m_b}$ for some $m_b \in \mathbb{N}_{>0}$, and hence

$$\begin{aligned} a + ab &= 2a + b \\ &= 2^{m_a+1} + b \\ &= 2^{m_b}, \text{ i.e.,} \\ b &= 2^{m_b} - 2^{m_a+1}. \end{aligned}$$

Since also $a + b \in S$, we have

$$\begin{aligned} 2^{m_c} &= a + b \\ &= 2^{m_a} + \left(2^{m_b} - 2^{m_a+1}\right) \\ &= 2^{m_a} + (2^{m_b} - 2 \cdot 2^{m_a}) \\ &= 2^{m_b} - 2^{m_a}, \text{ i.e.,} \\ 2^{m_b} &= 2^{m_c} + 2^{m_a} \end{aligned}$$

for some $m_c \in \mathbb{N}_{>0}$. Hence we must have $m_a \geqslant 1$, and furthermore $m_c = m_a$. But then $m_b = m_a + 1$, and we obtain

$$\begin{aligned} S \ni a + 2b &= 2^{m_a} + 2 \cdot 2^{m_a+1} \\ &= 2^{m_a} + 2^{m_a+2} \notin P. \end{aligned}$$

This contradicts $S \subseteq P$, hence S cannot be infinite. Since P, however, is infinite, and there are no infinite linear subsets of P, we immediately obtain that P cannot be a finite union of linear sets. Hence, we conclude that P is not semi-linear. ■

Lemma 4.2
We have

$$P \notin \Psi\mathrm{CF}\,. \qquad \square$$

Proof (Lemma 4.2) We assume to the contrary that $P \in \Psi\mathrm{CF}$. Then there is a language $L \in \mathrm{CF}$ such that $\Psi[L] = P$. By Theorem 2.32 P must be semi-linear, but this contradicts Lemma 4.1. Hence, we conclude $P \notin \Psi\mathrm{CF}$. ■

Corollary 4.3
Since $P \subseteq \mathbb{N}^1$*, we immediately obtain*

$$P \notin \mathcal{N}\mathrm{CF}\,. \qquad \square$$

4.2. A CD grammar system

Example 4.4 ([Csu+94, Example 3.2])
Consider the CD grammar system

$$\begin{aligned} \Gamma &= \big(\{A, S\}, \{a\}, S, G_1, G_2, G_3\big), \text{ where} \\ G_1 &= \{S \to AA\}, \\ G_2 &= \{A \to S\}, \text{ and} \\ G_2 &= \{A \to a\}. \end{aligned}$$

Clearly, any derivation in the terminating cooperation protocol for Γ proceeds as follows:

$$\begin{aligned} S &\Longrightarrow^t_{G_1} AA \\ &\Longrightarrow^t_{G_2} SS \\ &\Longrightarrow^t_{G_1} A^4 \\ &\Longrightarrow^t \dots \\ &\Longrightarrow^t_{G_2} S^{2^{n-1}} \\ &\Longrightarrow^t_{G_1} A^{2^n} \\ &\Longrightarrow^t_{G_3} a^{2^n} \end{aligned}$$

for some $n \in \mathbb{N}_{>0}$.

Thus, the language generated by Γ is

$$\begin{aligned} \mathcal{L}_t(\Gamma) &= \big\{a^{2^n} \bigm| n \in \mathbb{N}_{>0}\big\}, \text{ and} \\ \Psi\big[\mathcal{L}_t(\Gamma)\big] &= \{2^n \mid n \in \mathbb{N}_{>0}\} \subsetneq P. \end{aligned}$$ □

In order to generate P, i.e., adapting Γ such that it may also generate 1, we need to introduce another non-terminal symbol: A rule $S \to a$ would have to be added to the first component G_1 (otherwise, it would never be applicable, as the system starts in the first component and only transfers if no rule in G_1 is applicable, i.e., only after every S has been replaced by AA), but then we could also derive a^3, but $\Psi(a^3) = 3 \notin P$.

Example 4.5
Consider the following CD grammar system Γ':

$$\begin{aligned}\Gamma' &= \big(\{A,S,T\},\{a\},T,G_1,G_2,G_3\big), \text{ where}\\ G_1 &= \{T \to a, T \to S, S \to AA\},\\ G_2 &= \{A \to S\}, \text{ and}\\ G_2 &= \{A \to a\}.\end{aligned}$$

Clearly, we have

$$\begin{aligned}\mathcal{L}_t(\Gamma') &= \big\{a^{2^n} \,\big|\, n \in \mathbb{N}\big\}, \text{ and}\\ \Psi\big[\mathcal{L}_t(\Gamma')\big] &= \{2^n \mid n \in \mathbb{N}\} = P.\end{aligned}$$ □

4.3. An SN P system with cooperating rules

We now aim to construct an SN P system with cooperating rules generating P. Our approach is to start off with a single spike and to continue doubling the number of spikes until, at some point, the system decides to stop. This suggests to first look at two smaller systems that handle the doubling and the stopping, respectively, and indeed we shall see that the structure of SN P systems makes it easy to combine these two modules into a single system afterwards. We will use the terminating protocol and the maximally parallel mode for this example, taking the number of spikes sent to the environment as the generated number.

4.3.1. Doubling the spikes

We can easily achieve duplication of spikes by having a neuron send its spikes to two other neurons, and both of them sending the spikes back to the first neuron. Figure 4.1 depicts such a module that will repeatedly double the number of spikes in the acc neuron.

While it would suffice to have a single rule $(a^+, a) \to a$ in each of the neurons for the purpose of doubling the number of spikes, this would deprive us of the ability to stop the process through the introduction of additional spikes into the module. Hence, we have rules handling either a single spike, two spikes, or a multiple of four spikes. This allows us to stop the process by introducing five additional spikes into the system.

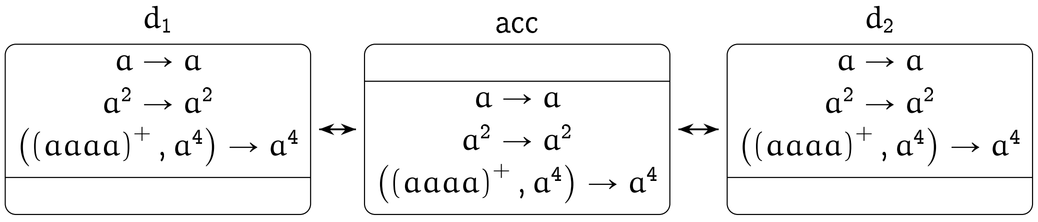

Figure 4.1.: Doubling of spikes

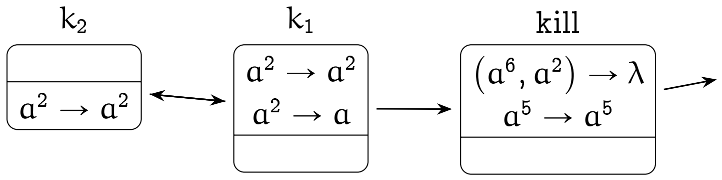

Figure 4.2.: The non-deterministic kill-switch

4.3.2. The kill-switch

As the only source of non-determinism in the system is the selection of the rule to be fired from among the applicable rules, it is exactly this mechanism we need to exploit to allow the system to decide which number to generate, the idea being that, depending on the rule chosen, the computation is either stopped or continues.

We depict such a module in fig. 4.2. Initially, the k_1 neuron contains two spikes and the kill neuron starts with four spikes, whereas k_2 remains empty. If k_1 emits two spikes, these two are then forgotten in kill. Control transfers to the second component, k_2 emits the two spikes, control transfers back to the first component, and the module has returned to its initial configuration. If k_1 emits only a single spike, however, kill emits five spikes, and computation then stops as none of the neurons contains an applicable rule anymore.

4.3.3. Putting it all together

Clearly, we need some kind of marker that signals the rest of the system that no further doubling of spikes should occur. It turns out that neither a nor a^3 are sufficient, since then a^2 and a^4 may arise either through the doubling of a and a^2, respectively, or through receiving the a or a^3 marker while there is already a single spike present in the neuron, and we have no way of distinguishing these two possibilities. This is not the case for a^5, however, as a^6 will never arise through the doubling. We also need to add one more rule to the acc neuron to output the last spike after receiving the a^5 marker.

Example 4.6
Figure 4.3 depicts the full system. We start with one spike in the acc neuron, two spikes in the k_1 neuron, and four spikes in the kill neuron.

Initially, the first component is active, and k_1 is the only neuron where a rule can be applied. One of the two rules is chosen non-deterministically, and either one or two spikes are emitted. Then, only kill has an applicable rule. For now, assume k_1 indeed sends out two spikes. These two spikes are then deleted in kill, and no more rules in the first component are applicable. Thus, control transfers to the second component. The neuron acc sends its contents (only a single spike for now) to d_1, d_2, and the environment, and then k_2 sends two spikes back into k_1. Then control transfers back to the first component. This process repeats until k_1 emits only a single spike. Assume that this happens after $n \in \mathbb{N}$ cycles of the above process. If $n > 0$, then acc has already sent

$$\sum_{i=0}^{n} 2^i = 2^n - 1$$

spikes to the environment. acc will now send out a final spike, bringing the total to 2^n (note that $2^0 = 1$, so this also works for $n = 0$). None of the rules in d_1 and d_2 are applicable (in fact, no rule in the system at all is applicable), so the computation stops.

Clearly, the language generated by Π working according to the terminating protocol in the maximally-parallel mode and taking the total number of spikes sent to the environment is

$$C_{tot}\mathcal{N}_t^{maxpar}(\Pi) = \{2^n \mid n \in \mathbb{N}\} = P. \qquad \square$$

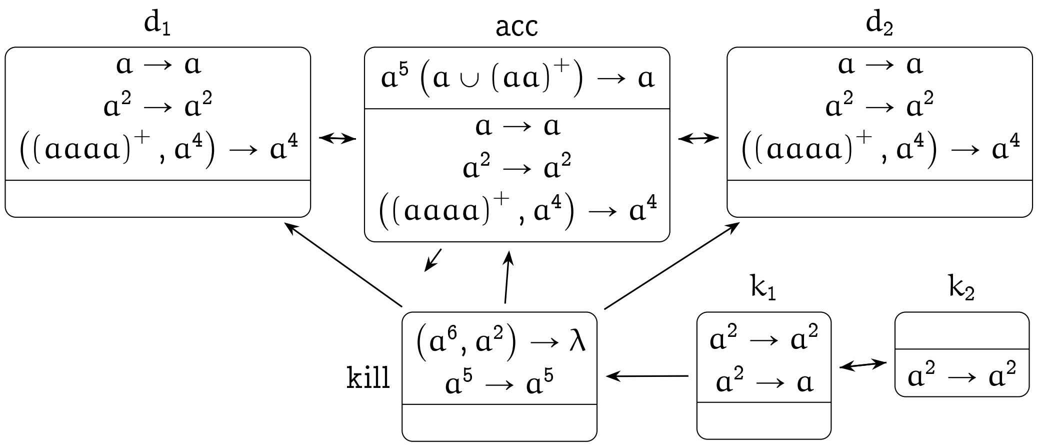

Figure 4.3.: The complete system Π

4.4. Comparing the two approaches

While the CD grammar system Γ' is ostensibly simpler in terms of the number of rules, it needs three components, whereas for the SN P system with cooperating rules Π, two components suffice. Furthermore, Π uses only a single symbol (and no non-terminal symbols at all), whereas Γ' needs four. If we were to add a second symbol to Π serving as the termination marker, we could easily eliminate the kill neuron and greatly simplify the rules in the d_1, d_2, and acc neurons. This approach somewhat resembles the "toxic objects" introduced in [AF14] for regular P systems, and lifting this concept to SN P systems may be a valuable approach in constructing smaller systems in general.

Another aspect worth considering is that the generative capacity of CD grammar systems is limited to the ET0L family of languages, whereas SN P systems with cooperating rules can generate any language in RE:

$$\mathcal{N}CD_\infty(t) = \mathcal{N}ET0L \subsetneq \mathcal{N}CS \subsetneq \mathcal{N}RE = \mathcal{N}C_2SN_{tot}P_t^{maxpar}(gene)\,.$$

Finally, SN P systems with cooperating rules can be composed in a natural fashion: We have already seen how bigger systems can be assembled from smaller modules like building blocks, and Example 4.7 shows how we can compute the intersection of the languages generated by two systems. For CD grammar systems, there is no comparable construction. Indeed,

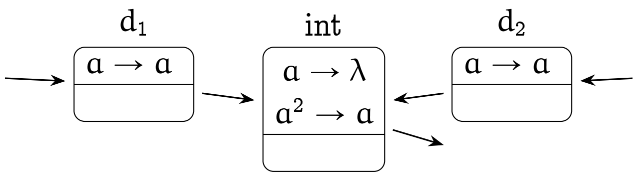

Figure 4.4.: Intersection module

the family ET0L is not even closed under intersection (this follows by the same argument given for indexed languages in [Aho68, Theorem 4.4], since ET0L is also closed under morphisms (cf. [KRS97, Theorem 2.8]) and contains CF).

Example 4.7
Let Π_1, Π_2 be two-component SN P systems with cooperating rules.

Then we can construct a system Π such that

$$C_{tot}\mathcal{N}_t^{maxpar}(\Pi) = C_{tot}\mathcal{N}_t^{maxpar}(\Pi_1) \cap C_{tot}\mathcal{N}_t^{maxpar}(\Pi_2)$$

by adding three neurons and connecting the output neuron of Π_i to the respective neuron d_i of Π. Figure 4.4 depicts the additional neurons, where int becomes the new output neuron. □

A similar module allows for the computation of the concatenation of the languages computed by two systems. While this is a construction that is also possible with CD grammar systems, it involves modifying existing rules (at the very least the production rule for the starting symbol has to be changed) and possibly the renaming of non-terminal symbols. In contrast, we only need to add neurons and synapses to the SN P system, leaving the rest of the system unchanged.

5. Conclusion

5.1. What we have done

We have provided a comprehensive introduction into the theory of SN P systems with cooperating rules and their predecessors, CD grammar systems and SN P systems, and we have shown that the universality result from [MRK14b] for the terminating protocol can be retained for the other protocols, answering one of the open questions of that paper, and which of the restrictions on the types of rules or the working mode of the system need to be relaxed in order to do so. Furthermore, we have designed an SN P system with cooperating rules that generates a non-context-free language, and have shown that such system are superior to CD grammar systems in terms of composability.

5.2. Future work

From the open questions in [MRK14b], we have answered the question of universality in the face of other cooperation protocols. Still, we feel that a critical examination of whether all of the extensions used in our proofs are indeed necessary to obtain universality. We suspect that while we cannot avoid using extended rules, or the maximally parallel or the locally synchronous mode, respectively, there might be a construction avoiding the use of general neurons for at least the maximally parallel case.

Small examples of universal systems are given by [MRK14a] and [SP14]. Since two components suffice for universality using any of the cooperation protocols, we do not expect studying systems with more components to provide further insights, and single-component systems are just ordinary SN P systems.

Several extensions of CD grammar systems have been extensively studied, among them hybrid CD grammar systems, where different components are allowed to work according to different cooperation protocols. While this concept clearly would have no impact on the computational power of SN P systems with cooperating rules, we feel that consideration of such hybrid systems may aid in finding smaller universal systems. Another concept that has been studied for CD grammar systems is that of a controlled system, where the active component is not chosen non-deterministically, but according to some external or internal control mechanism (cf. [Csu+94]). However, an *external* control mechanism (i.e., a mechanism where the allowed control transfers are fixed in advance) does not seem to be of much use in the case of SN P systems with cooperating rules (since two components suffice for universality, for, e.g., the terminating protocol, the non-deterministic choice degrades to simply switching to the other component), and we suspect that most non-trivial examples may be more suitable expressed as a hybrid system. On the other hand, an *internal* control mechanism (where the active component is determined by some predicate on the current state of the system) promises to lead to a further reduction in system size. As a starting point, we suggest looking into systems where control transfer happens whenever a certain neuron (or a set of neurons) contains no spikes.

Clearly, *SN P system with cooperating rules* is too unwieldy a term to repeat throughout, and we propose that a suitable abbreviation be introduced. [Wu+16b] introduces cooperating rules to (catalytic) P systems and proposes the term *colored P system*. We thus propose adopting the name *colored SN P system*.

[Wu+16a] introduces *cell-like SN P systems*, which bring the nested, tree-like *membrane structure* of conventional P systems to the world of SN P systems. We assume that combining this approach with cooperating rules may lead to a powerful tool for the modeling of biological processes, and suggest further investigation.

We strongly feel that modeling a real-world biological process using the framework of SN P systems with cooperating rules (and quite possibly other extensions) would provide insights into both the theory of SN P systems with cooperating rules and the process itself.

As a starting point, it may be a good idea to focus on a process already modeled using other variants of P systems, e.g., the quorum sensing behavior in the vibrio fischeri bacteria (cf. [RP08]) or the photosynthesis modeled in [Nis06] (or any of the other applications described in [CPP06]), although in all probability neither process can be modeled sufficiently without further extending the framework.

In any case, certain qualities of a process lend themselves to modeling using SN P systems with cooperating rules, among them

- being discrete in nature,
- involving relatively few different kinds of objects, and
- behaving in a locally synchronized and globally asynchronous way.

Extending this list is a further opportunity for future work. Ideally, we would want a checklist allowing us to decide whether a certain process is amenable to modeling using SN P systems with cooperating rules (or any other extension, for that matter).

A. Bibliography

[ABB97] Jean-Michel Autebert, Jean Berstel, and Luc Boasson. “Context-Free Languages and Pushdown Automata.” English. In: *Handbook of formal languages, Volume 1: Word, language, grammar*. Ed. by Grzegorz Rozenberg and Arto Salomaa. Springer, 1997. Chap. 3, pp. 111–174.

[AF14] Artiom Alhazov and Rudolf Freund. “P Systems with Toxic Objects.” English. In: *Proceedings of the 15th International Conference on Membrane Computing, August 2014, Prague*. Ed. by Marian Gheorghe et al. Vol. 8961. Lecture Notes in Computer Science. Springer, 2014, pp. 99–125.

[Aho68] Alfred V. Aho. “Indexed Grammars—An Extension of Context-Free Grammars.” English. In: *Journal of the ACM* 15.4 (1968), pp. 647–671.

[Awo06] Steve Awodey. *Category Theory*. English. Vol. 49. Oxford logic guides. Oxford University Press, 2006.

[BCF89] Avron Barr, Paul R. Cohen, and Edward A. Feigenbaum, eds. *The handbook of artificial intelligence, Volume 4*. English. Addison-Wesley, 1989.

[BN98] Franz Baader and Tobias Nipkow. *Term rewriting and all that*. English. Cambridge University Press, 1998.

[BPS61] Yehoshua Bar-Hillel, Micha Perles, and Eli Shamir. “On formal properties of simple phrase-structure grammars.” English. In: *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung* 14.1-4 (1961), pp. 143–172.

[Cav+09] Matteo Cavaliere et al. “Asynchronous spiking neural P systems.” English. In: *Theoretical Computer Science* 410.24–25 (2009), pp. 2352–2364.

[CD90] Erzsébet Csuhaj-Varjú and Jürgen Dassow. “On Cooperating/Distributed Grammar Systems.” English. In: *Journal of Information Processing and Cybernetics* 26.1-2 (1990), pp. 49–63.

[Chu36] Alonzo Church. “An Unsolvable Problem of Elementary Number Theory.” English. In: *American Journal of Mathematics* 58.2 (1936), pp. 345–363.

[CPP06] Gabriel Ciobanu, Gheorghe Păun, and Mario J. Pérez-Jiménez, eds. *Applications of Membrane Computing.* English. Natural computing series. Springer, 2006.

[Csu+94] Erzsébet Csuhaj-Varjú et al. *Grammar Systems: A Grammatical Approach to Distribution and Cooperation.* English. Gordon and Breach Science Publishers, 1994.

[DPR97] Jürgen Dassow, Gheorghe Păun, and Grzegorz Rozenberg. “Grammar Systems.” English. In: *Handbook of formal languages, Volume 2: Linear modeling: background and application.* Ed. by Grzegorz Rozenberg and Arto Salomaa. Springer, 1997. Chap. 4, pp. 155–214.

[EP02] Katrin Erk and Lutz Priese. *Theoretische Informatik: eine umfassende Einführung.* German. 2., erw. Aufl. Springer, 2002.

[Fri09] Pierluigi Frisco. *Computing with Cells: Advances in Membrane Computing.* English. Oxford University Press, 2009.

[Ghe+14] Marian Gheorghe et al., eds. *Proceedings of the 15th International Conference on Membrane Computing, August 2014, Prague.* English. Vol. 8961. Lecture Notes in Computer Science. Springer, 2014.

[Gol77] Jonathan Goldstine. “A simplified proof of Parikh’s theorem.” English. In: *Discrete Mathematics* 19.3 (1977), pp. 235–239.

[Grä08] George A. Grätzer. *Universal algebra.* English. 2. ed., with updates. Springer, 2008.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation*. English. Addison-Wesley, 1979.

[IPY06] Mihai Ionescu, Gheorghe Păun, and Takashi Yokomori. "Spiking Neural P Systems." English. In: *Fundamenta Informaticae* 71.2-3 (2006), pp. 279–308.

[Kor96] Ivan Korec. "Small universal register machines." English. In: *Theoretical Computer Science* 168.2 (1996), pp. 267–301.

[KRS97] Lila Kari, Grzegorz Rozenberg, and Arto Salomaa. "L Systems." English. In: *Handbook of formal languages, Volume 1: Word, language, grammar*. Ed. by Grzegorz Rozenberg and Arto Salomaa. Springer, 1997. Chap. 5, pp. 253–328.

[Kui97] Werner Kuich. "Semirings and Formal Power Series." English. In: *Handbook of formal languages, Volume 1: Word, language, grammar*. Ed. by Grzegorz Rozenberg and Arto Salomaa. Springer, 1997. Chap. 9, pp. 609–678.

[Lin68a] Aristid Lindenmayer. "Mathematical models for cellular interactions in development I. Filaments with one-sided inputs." English. In: *Journal of Theoretical Biology* 18.3 (1968), pp. 280–299.

[Lin68b] Aristid Lindenmayer. "Mathematical models for cellular interactions in development II. Simple and Branching Filaments with two-sided inputs." English. In: *Journal of Theoretical Biology* 18.3 (1968), pp. 300–315.

[Min67] Marvin L. Minsky. *Computation: finite and infinite machines*. English. Prentice-Hall, 1967.

[MRK14a] Venkata Padmavati Metta, Srinivasan Raghuraman, and Kamala Krithivasan. "Small Universal Spiking Neural P Systems with Cooperating Rules as Function Computing Devices." English. In: *Proceedings of the 15th International Conference on Membrane Computing, August 2014, Prague*. Ed. by Marian Gheorghe et al. Vol. 8961. Lecture Notes in Computer Science. Springer, 2014, pp. 300–313.

[MRK14b] Venkata Padmavati Metta, Srinivasan Raghuraman, and Kamala Krithivasan. “Spiking Neural P Systems with Cooperating Rules.” English. In: *Proceedings of the 15th International Conference on Membrane Computing, August 2014, Prague*. Ed. by Marian Gheorghe et al. Vol. 8961. Lecture Notes in Computer Science. Springer, 2014, pp. 314–329.

[MS97a] Alexandru Mateescu and Arto Salomaa. “Aspects of Classical Language Theory.” English. In: *Handbook of formal languages, Volume 1: Word, language, grammar*. Ed. by Grzegorz Rozenberg and Arto Salomaa. Springer, 1997. Chap. 4, pp. 175–252.

[MS97b] Alexandru Mateescu and Arto Salomaa. “Formal Languages: an Introduction and a Synopsis.” English. In: *Handbook of formal languages, Volume 1: Word, language, grammar*. Ed. by Grzegorz Rozenberg and Arto Salomaa. Springer, 1997. Chap. 1, pp. 1–40.

[Nii89] H. Penny Nii. “Blackboard Systems.” English. In: *The handbook of artificial intelligence, Volume 4*. Ed. by Avron Barr, Paul R. Cohen, and Edward A. Feigenbaum. Addison-Wesley, 1989. Chap. XVI, pp. 1–82.

[Nis06] Taishin Yasunobu Nishida. “A Membrane Computing Model of Photosynthesis.” English. In: *Applications of Membrane Computing*. Ed. by Gabriel Ciobanu, Gheorghe Păun, and Mario J. Pérez-Jiménez. Natural computing series. Springer, 2006. Chap. 6, pp. 181–202.

[Par66] Rohit J. Parikh. “On Context-Free Languages.” English. In: *Journal of the ACM* 13.4 (1966), pp. 570–581.

[Pău00] Gheorghe Păun. “Computing with Membranes.” English. In: *Journal of Computer and System Sciences* 61.1 (2000), pp. 108–143.

[Pău02] Gheorghe Păun. *Membrane Computing: An Introduction*. English. Natural computing series. Springer, 2002.

[PP09] Gheorghe Păun and Mario J. Pérez-Jiménez. “Spiking Neural P Systems. Recent Results, Research Topics.” English. In: *Algorithmic Bioprocesses*. Ed. by Anne Condon et al. Natural Computing Series. Springer, 2009, pp. 273–291.

[PPR06] Gheorghe Păun, Mario J. Pérez-Jiménez, and Grzegorz Rozenberg. “Spike trains in spiking neural P systems.” English. In: *International Journal of Foundations of Computer Science* 17.04 (2006), pp. 975–1002.

[PR02] Gheorghe Păun and Grzegorz Rozenberg. “A guide to membrane computing.” English. In: *Theoretical Computer Science* 287.1 (2002), pp. 73–100.

[PRS10] Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa. *DNA computing: New computing paradigms*. English. Springer, 2010.

[RP08] Francisco J. Romero-Campero and Mario J. Pérez-Jiménez. “A model of the quorum sensing system in Vibrio fischeri using P systems.” English. In: *Artificial Life* 14.1 (2008), pp. 95–109.

[RS97a] Grzegorz Rozenberg and Arto Salomaa, eds. *Handbook of formal languages, Volume 1: Word, language, grammar*. English. Springer, 1997.

[RS97b] Grzegorz Rozenberg and Arto Salomaa, eds. *Handbook of formal languages, Volume 2: Linear modeling: background and application*. English. Springer, 1997.

[Sip97] Michael Sipser. *Introduction to the theory of computation*. English. PWS Publishing Company, 1997.

[SP14] Tao Song and Linqiang Pan. “A Small Universal Spiking Neural P System with Cooperating Rules.” English. In: *Romanian Journal of Information Science and Technology* 17.2 (2014), pp. 177–189.

[SPP13] Tao Song, Linqiang Pan, and Gheorghe Păun. “Asynchronous spiking neural P systems with local synchronization.” English. In: *Information Sciences* 219 (2013), pp. 197–207.

[Tur36] Alan Mathison Turing. “On computable numbers, with an application to the Entscheidungsproblem.” English. In: *Journal of Mathematics* 58.5 (1936), pp. 345–363.

[Weg05] Ingo Wegener. *Theoretische Informatik: eine algorithmenorientierte Einführung*. German. 3., überarb. Aufl. Teubner, 2005.

[Wu+16a] Tingfang Wu et al. “Cell-like spiking neural P systems.” English. In: *Theoretical Computer Science* (2016). to appear.

[Wu+16b] Tingfang Wu et al. “On the Universality of Colored One-Catalyst P Systems.” English. In: *Fundamenta Informaticae* 144.2 (2016), pp. 205–212.

B. Index

C. Declaration of Authorship

Hiermit erkläre ich, dass ich die am heutigen Tag eingereichte Diplomarbeit zum Thema „Universality Results for Spiking Neural P Systems with Cooperating Rules“ unter Betreuung von Prof. Dr.-Ing. Franz Baader und Dr.-Ing. Monika Sturm selbständig erarbeitet, verfasst, und Zitate kenntlich gemacht habe. Andere als die angegebenen Hilfsmittel wurden von mir nicht benutzt.

I hereby certify that I am the sole author of this diploma thesis on "Universality Results for Spiking Neural P Systems with Cooperating Rules" that I submit today and wrote under supervision of Prof. Dr.-Ing. Franz Baader und Dr.-Ing. Monika Sturm, and that I designated every citation. I have used only the listed references.

Dresden,	....
2016-03-14	Maximilian Marx