Systolic Automata and P Systems

Roberto Barbuti[1], Andrea Maggiolo-Schettini[1], Paolo Milazzo[1], Giovanni Pardini[1(✉)], and Simone Tini[2]

[1] Dipartimento di Informatica, Università di Pisa,
Largo B. Pontecorvo 3, 56127 Pisa, Italy
{barbuti,maggiolo,milazzo,pardinig}@di.unipi.it
[2] Dipartimento di Scienza e Alta Tecnologia, Università dell'Insubria,
Via Valleggio 11, 22100 Como, Italy
simone.tini@uninsubria.it

Abstract. Systolic automata are models of highly-concurrent language acceptors based on identical processors with one-way flow of information, amenable to efficient hardware implementation as multiprocessor chips.

In this paper we investigate the relationship between Binary Systolic Tree Automata (BSTA), in which the underlying communication structure is an infinite complete binary tree with parallel bottom-up computation, and P systems, a biologically-inspired formalism based on rewrite rules acting upon multisets of symbols with a maximally-parallel semantics.

In particular, we propose a variant of BSTA as multiset languages acceptors, termed Multiset BSTA. By exploiting the similarity in the parallel computation as performed in both BSTA and P systems, we show how a Multiset BSTA can be simulated by a cooperative P system while preserving the computational efficiency of systolic automata.

1 Introduction

Systolic automata are highly parallel language acceptors inspired by the functioning of VLSI architectures [14,17]. A systolic automaton is an infinite tree associated with an input function g and a processing function f. Without loss of generality, the tree of a systolic automaton is often assumed to be binary, thus obtaining the class of Binary Systolic Tree Automata (BSTAs). The input function g maps each symbol of the considered input alphabet into a working symbol from an operating alphabet. The processing function f, instead, maps two working symbols into one. The way in which a BSTA processes a candidate string to determine whether it belongs or not to the accepted language is by feeding its tree at a suitable level with such a string, and then by applying the processing function at each level (from bottom to top) in order to produce, in the root of the tree, a single operating symbol. If such a symbol belongs to the accepting alphabet (subset of the operating alphabet) then the candidate string is accepted, otherwise it is not accepted. Note that a string w of length m has to be accepted at smallest level n of the tree such that $m \leq 2^n$. Moreover, if $m < 2^n$, then $2^n - m$ instances of the special symbol $\sharp$ are appended to the string.

C.S. Calude et al. (Eds.): Gruska Festschrift, LNCS 8808, pp. 17–31, 2014.
DOI: 10.1007/978-3-319-13350-8_2

It has been proved that BSTAs can accept all regular languages [17]. Moreover, by exploiting the tree structure they can accept higher level languages such as $a^{2^n}b^{2^n}$. The class of languages accepted by a BSTA is a subset of the class E0L [24] called *Systolic E0L* [13]. Many variants of systolic automata have been proposed (see, e.g., [15,16,19]), and many studies have been performed on them (see, e.g., [18,20,21])

The processing of a candidate string by a BSTA is performed in a highly parallel way. Symbols to be associated with nodes at level i of the tree can be obtained by the symbols associated with nodes at level $i+1$ by applying f, the processing function, 2^i times in parallel to each pair of children nodes of nodes at level i. This is repeated for each level of the tree until the root is reached. Hence, an execution of a BSTA can be seen as a sequence of highly parallel steps in which *all* of the currently available symbols are processed and "transformed" into symbols to be used in the next step. The form of parallelism at the basis of the functioning of BSTAs is hence very similar to the notion of *maximal parallelism* considered in the context of P Systems [23]. P Systems are a form of hierarchical multiset rewriting systems. Maximal parallelism in P systems states that, at each step of their execution, rewriting rules must be applied in parallel to a sub-multiset of the available symbols (possibly to *all* of them) so that no rule can be applied to the remaining symbols.

The aim of this paper is hence to investigate the relationship between BSTAs and P systems based on the similarity of the forms of parallelism they consider. Since P systems are used to process multisets, the first thing we do is to define a variant of BSTAs, called MBSTAs, that can be used to accept *multiset languages* rather than languages of strings. As usual when passing from string languages to multiset languages ([12]), we show that MBSTAs can accept every context-free multiset language. Then, we face the problem of translating a MBSTA into an equivalent P system (used as language acceptor [9]). To this aim we define another (equivalent) variant of MBSTAs, called Regular MBSTAs, in which some regularity conditions are assumed. Finally, we define a translation of RMBSTAs into P systems and prove that P systems obtained after translation are as efficient as the original RMBSTAs.

2 Background

Let $\mathbb{N}$ be the set of natural numbers and $\mathbb{N}^+$ denote $\mathbb{N} \setminus \{0\}$. Elements of sets are enumerated between { and }, while elements of multisets are enumerated between {| and |}. Given a finite alphabet Λ, we denote by Λ^* the set of all finite strings over Λ, namely $\epsilon \in \Lambda^*$, for ϵ the empty string, and $aw \in \Lambda^*$, for $a \in \Lambda$ and $w \in \Lambda^*$. Given two sets of strings Z_1, Z_2, their concatenation is denoted $Z_1.Z_2 = \{w_1w_2 \mid w_1 \in Z_1, w_2 \in Z_2\}$. The number of occurrences of a symbol a in a string w is denoted $|w|_a$; moreover, given a set $Z \subseteq \Lambda$, $|w|_Z = \sum_{a \in Z} |w|_a$. The length of w is denoted $|w|$. The i^{th} element of w is denoted w_i. We denote with Λ^+ the set $\Lambda^* \setminus \{\epsilon\}$. As usual, a language over Λ is a subset $L \subseteq \Lambda^*$. We denote $\mathcal{M}(\Lambda)$ the set of all the multisets with elements in Λ. The union of

multisets is denoted by $\oplus$, $\setminus$ denotes both the difference between sets and the difference between multisets, and $\emptyset$ denotes both the empty set and the empty multiset. Moreover, we denote with $\mathcal{P}(I)$ the powerset of I, that is the set of all subsets of the set I. Given a function $f : A \to A$, we define f^n as $f^0(x) = x$ and $f^n(x) = f(f^{n-1}(x))$.

Definition 1 (Parikh mapping). *Let $\Lambda = \{a_1, a_2, \ldots, a_n\}$ be an ordered alphabet. The* Parikh mapping *over strings $\phi : \Lambda^* \to \mathbb{N}^n$ is defined as follows:*

$$\phi(w) = (|w|_{a_1}, |w|_{a_2}, \ldots, |w|_{a_n}).$$

The Parikh mapping of a language $L \subseteq \Lambda^*$ is defined as $\phi(L) = \{\phi(w) \mid w \in L\}$.

In the rest of the paper, we always assume alphabets to be ordered. Therefore there is a one-to-one correspondence between Parikh vectors in $\mathbb{N}^{|\Lambda|}$ and multisets over Λ. For this reason, with a slight abuse of notation, we assume to denote by $\phi(w)$, for any string $w \in \Lambda^*$, both the Parikh vector and the multiset over Λ described by w.

We recall from [22] the definition of Binary Systolic Tree Automata [14,17].

Definition 2 (Binary Systolic Tree Automaton). *A* Binary Systolic Tree Automaton (BSTA)[1] *is a construct $K = (\Lambda, Q, F, f, g)$, where Λ is the finite* input alphabet*, $Q \cup \{\sharp\}$ is the finite* operating alphabet *(with $\sharp$ being a special symbol outside of Λ, Q), $F \subseteq Q \cup \{\sharp\}$ is the* accepting alphabet*, $f : (Q \cup \{\sharp\}) \times (Q \cup \{\sharp\}) \to Q \cup \{\sharp\}$ is the* processing function *and $g : \Lambda \cup \{\sharp\} \to Q \cup \{\sharp\}$ is the* input function*. Moreover, the processing function is such that $f(x, y) = \sharp$ iff $x = y = \sharp$; while the input function is such that $g(x) = \sharp$ iff $x = \sharp$.*

A BSTA is interpreted as an infinite complete binary tree, in which the processing function f is associated with each node. A BSTA can accept strings on the alphabet Λ in the following way. Given a string $w \in \Lambda^*$ having length m, we take the smallest level n of the tree with at least m nodes. If $m < 2^n$, let $\ell = 2^n - m$. The string $w\sharp^\ell$ is transformed by means of the input function, by applying g to each one of its symbols, preserving the ordering. The string in $(Q \cup \{\sharp\})^*$ obtained is then fed to the level n of the tree. Precisely, the symbols in the transformed string are given as input, in order, to the nodes of the cut at level n, starting from the leftmost node.

At the first step, once each node of the cut has an input in $Q \cup \{\sharp\}$, all processing functions of the level $n-1$ get, in parallel, the two inputs from their children nodes and produce their results, a symbol in $Q \cup \{\sharp\}$ for each node. This process is iterated for n steps, resulting in a symbol $q \in Q \cup \{\sharp\}$ being produced in the root of the tree. If $q \in F$ then the string is accepted, otherwise it is rejected by the BSTA.

The definition of BSTAs reported here, from [22], includes constraints on the behaviour of the processing and input function when dealing with the special symbol $\sharp$ which were not assumed in the original definition ([14]). Note that

[1] Also known in the literature by the acronyms SBTA and BT-VLSI.

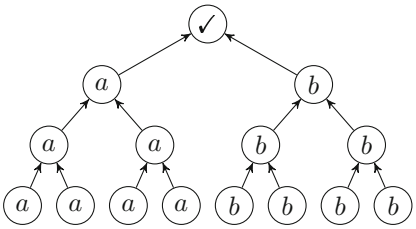

Fig. 1. BSTA accepting computation

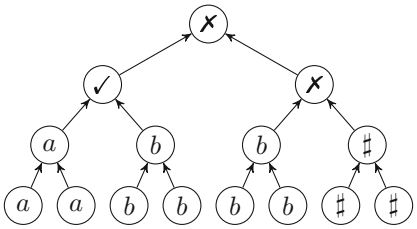

Fig. 2. BSTA rejecting computation

such constraints on functions f and g imply that the value of function $f(\sharp, x)$, for $x \in Q$, is irrelevant, since it is never used by a BSTA. In fact, at each level of the tree, any and all $\sharp$ symbols may only occur at the end of the string.

Definition 3 (Language accepted by a BSTA). *Consider a BSTA,* $K = (\Lambda, Q, F, f, g)$. *We define* $\overline{f} : (Q \cup \{\sharp\})^* \to (Q \cup \{\sharp\})^*$ *and* $\overline{g} : (\Lambda \cup \{\sharp\})^* \to (Q \cup \{\sharp\})^*$ *as follows:*

$$\overline{f}(w_1 w_2 \dots w_m) = f(w_1, w_2) f(w_3, w_4) \dots f(w_{m-1}, w_m) \qquad \text{if } m \text{ is even;}$$
$$\overline{g}(w_1 w_2 \dots w_m) = g(w_1) g(w_2) \dots g(w_m).$$

Given a string $w \in \Lambda^*$, *let* $n \in \mathbb{N}$ *be such that* $|w| \leq 2^n$. *Then* w is accepted by K at level n *iff* $\overline{f}^n(\overline{g}(w\sharp^\ell)) \in F$, *where* $\ell = 2^n - |w|$. *The string* w is accepted by K *iff it is accepted at the smallest level* n *with* $|w| \leq 2^n$. *Finally, the* language accepted by K *is the set of strings* $L(K) = \{w \in \Lambda^* \mid w \text{ is accepted by } K\}$.

Example 1. Let us consider the BSTA $K = (\Lambda, Q, F, f, g)$ where $\Lambda = \{a, b\}$, $Q = \{a, b, ✓, ✗\}$, $F = \{✓\}$, g is the identity function, and f is defined as follows:

$$f(a,a) = a \quad f(b,b) = b \quad f(a,b) = ✓ \quad f(\sharp,\sharp) = \sharp$$

and $f(x, y) = ✗$ for any other pair of symbols not defined above. Two possible computations of K are shown in Figure 1 and 2. The language accepted by the BSTA is $L(K) = \{a^{2^n} b^{2^n} \mid n \in \mathbb{N}\}$.

A useful property for the definition of BSTAs is to be able to give the string as input to any level of the tree with enough nodes, by relaxing the constraint that requires it to be fed to the smallest possible level. We recall the definition of *stable* BSTA, for which the result of either acceptance or rejection of a string w is independent of the actual level n ($|w| \leq 2^n$) to which the string is fed. Moreover, each BSTA can be transformed into an equivalent *stable* BSTA, as shown by the theorem which follows.

Definition 4 (Stable BSTA). *A BSTA* $K = (\Lambda, Q, F, f, g)$ *is* stable *iff for each string* $w \in \Lambda^*$ *and for all* $n_1, n_2 \in \mathbb{N}$ *with* $|w| \leq 2^{n_1} \leq 2^{n_2}$, *it holds that*

$$w \text{ is accepted by } K \text{ at level } n_1 \iff w \text{ is accepted by } K \text{ at level } n_2.$$

Theorem 1 ([14,22]). *For every BSTA* K *there exists a stable BSTA* K' *such that* $L(K) = L(K')$.

3 Multiset Binary Systolic Tree Automata

A *Multiset Binary Systolic Tree Automaton* (MBSTA) has the same structure of a BSTA, but it accepts multisets. Because multisets have no order among the elements, the elements of the multiset can be given as input to the nodes of the cut in any order. For a multiset to be accepted by a MBSTA there needs to be some order of the symbols yielding a final symbol in the root of the tree.

Definition 5 (Multiset language accepted by a MBSTA). *Consider a MBSTA, $M = (\Lambda, Q, F, f, g)$. Given a multiset $\mu \in \mathcal{M}(\Lambda)$, let $n \in \mathbb{N}$ be such that $|\mu| \leq 2^n$. Then μ* is accepted by M at level n *iff there exists a string $w \in \Lambda^*$ such that $\phi(w) = \mu$ and $\overline{f}^n(\overline{g}(w\sharp^\ell)) \in F$, where $\ell = 2^n - |\mu|$. The multiset μ* is accepted by M *iff it is accepted at the smallest level n such that $|\mu| \leq 2^n$. Finally, the* multiset language accepted by M *is set of multisets $L(M) = \{\mu \in \mathcal{M}(\Lambda) \mid \mu$ is accepted by $M\}$.*

The multiset language accepted by a MBSTA is obtained from the language accepted by the BSTA having the same structure through the Parikh mapping.

Proposition 1. *Assume a BSTA $K = (\Lambda, Q, F, f, g)$ and the MBSTA $M = (\Lambda, Q, F, f, g)$ having the same structure. Then*

1. *If a string $w \in \Lambda^*$ is accepted by K at level n then the multiset $\phi(w) \in \mathcal{M}(\Lambda)$ is accepted by M at level n;*
2. *If a multiset $\mu \in \mathcal{M}(\Lambda)$ is accepted by M at level n then there is a string $w \in \Lambda^*$ such that $\phi(w) = \mu$ and w is accepted by K at level n.*

Proof. Directly by Definition 5.

Corollary 1. *Assume a BSTA $K = (\Lambda, Q, F, f, g)$ and the MBSTA $M = (\Lambda, Q, F, f, g)$ having the same structure. Then $L(M) = \{\phi(w) \mid w \in L(K)\}$.*

The notion of stability is trivially extended to MBSTAs. Moreover, analogously to BSTAs, a MBSTA can always be transformed into an equivalent stable MBSTA.

Definition 6 (Stable MBSTA). *A MBSTA $M = (\Lambda, Q, F, f, g)$ is* stable *iff for each multiset $\mu \in \mathcal{M}(\Lambda)$ and $n_1, n_2 \in \mathbb{N}$ with $|\mu| \leq 2^{n_1} \leq 2^{n_2}$, it holds that*

$$\mu \text{ is accepted by } M \text{ at level } n_1 \iff \mu \text{ is accepted by } M \text{ at level } n_2.$$

Proposition 2. *Let $K = (\Lambda, Q, F, f, g)$ be a stable BSTA. Then the MBSTA $M = (\Lambda, Q, F, f, g)$ having the same structure is stable.*

Proof. Assume any multiset $\mu \in \mathcal{M}(\Lambda)$ and any pair of naturals $n_1, n_2 \in \mathbb{N}$ such that $|\mu| \leq 2^{n_1} \leq 2^{n_2}$. We prove that if μ is accepted by M at level n_1 then μ is accepted by M at level n_2, the converse case is analogous. If μ is accepted by M at level n_1, then by Proposition 1 (case 2) there exists some $w \in \Lambda^*$ such that $\phi(w) = \mu$ and w is accepted by K at level n_1. Since K is stable we infer that w is accepted by K at level n_2, thus implying, through Proposition 1 (case 1), that μ is accepted by M at level n_2.

Theorem 2. *For every MBSTA M there exists a stable MBSTA M' such that $L(M) = L(M')$.*

Proof. Given the MBSTA $M = (\Lambda, Q, F, f, g)$, we consider the BSTA $K = (\Lambda, Q, F, f, g)$ having the same structure of M. From Theorem 1 there exists a stable BSTA $K' = (\Lambda, Q', F', f', g')$ with $L(K) = L(K')$. Consider the MBSTA $M' = (\Lambda, Q', F', f', g')$ having the same structure of K'. From Proposition 2 we have that M' is stable. It remains to be shown that $L(M') = L(M)$. We prove that for any $\mu \in \mathcal{M}(\Lambda)$, $\mu \in L(M)$ implies $\mu \in L(M')$, the converse is analogous. If $\mu \in L(M)$, then by Corollary 1 we infer $w \in L(K)$ for some $w \in \Lambda^*$ with $\phi(w) = \mu$. From $L(K) = L(K')$ we infer $w \in L(K')$. Since K' and M' have the same structure, by Corollary 1 it follows $\mu \in L(M')$.

3.1 MBSTAs and MFAs

In this section we briefly study the expressive power of MBSTAs, by showing that they can accept all the languages recognized by Multiset Finite Automata. A *Multiset Finite automaton (MFA)* [12] is a finite automaton in which the input is given by a multiset of symbols of the alphabet. It starts in its initial state with the whole multiset of symbols, and changes its current state based on the state itself and on a symbol which is chosen from the multiset. The chosen symbol is removed from the multiset. The MFA stops when either no move is possible or the multiset is empty. We also recall the important fact that MFAs have the same expressive power as Multiset Context-Free Grammars ([12]), therefore they accept the class of *context-free multiset languages*.

Definition 7 (Multiset Finite Automaton). *A* Multiset Finite Automaton *is a construct $A = (\Lambda, Z, W, z_0, t)$, where Λ is the* input alphabet*, Z is the finite set of* states*, W is the set of* accepting states *($W \subseteq Z$), $z_0 \in Z$ is the* initial state*, and t is the* transition function *($t : Z \times \Lambda \to \mathcal{P}(Z)$).*

Definition 8 (Multiset language accepted by a MFA). *Given the* Multiset Finite Automaton *$A = (\Lambda, Z, W, z_0, t)$, a* configuration *of A is a pair (z, μ), where z is a state, $z \in Z$, and μ is a multiset of symbols, $\mu \in \mathcal{M}(\Lambda)$. Let $\vdash$ denote the following relation on configurations, whose reflexive and transitive closure is denoted $\vdash^*$:*

$$(z, \mu) \vdash (z', \mu') \iff z' \in t(z, a),\ a \in \Lambda,\ \mu' = \mu \backslash \{\!|a|\!\}.$$

A multiset $\mu \in \mathcal{M}(\Lambda)$ is accepted by A *iff* $(z_0, \mu) \vdash^* (z, \emptyset)$ *for some $z \in W$. The* multiset language accepted by A *is* $L(A) = \{\mu \in \mathcal{M}(\Lambda) \mid \mu$ *is accepted by* $A\}$.

Theorem 3. *Given an alphabet Λ, every context-free multiset language L is accepted by a MBSTA.*

Proof. The proof follows that in [17]. Consider the MFA $A = (\Lambda, Z, W, z_0, t)$ with $L(A) = L$, and build a MBSTA $K = (\Lambda, Q, F, f, g)$ with $L(K) = L$ as follows.

The processing alphabet Q of K is defined as $Q = \mathcal{P}(Z \times Z)$, where each symbol in Q corresponds therefore to a set of pairs (z_1, z_2) of states of A.

The input function $g : \Lambda \cup \{\sharp\} \to Q \cup \{\sharp\}$ is defined as follows:

$$g(a) = \{(z_i, z_j) \mid z_i, z_j \in Z,\ z_j \in t(z_i, a)\} \qquad a \in \Lambda;$$
$$g(\sharp) = \sharp;$$

while the processing function $f : Q \cup \{\sharp\} \times Q \cup \{\sharp\} \to Q \cup \{\sharp\}$ is defined as

$$f(I, J) = \{(z_i, z_j) \mid (z_i, z_k) \in I,\ (z_k, z_j) \in J\};$$
$$f(I, \sharp) = I;$$
$$f(\sharp, \sharp) = \sharp.$$

Finally, let $F_0 = \{I \in Q \mid \exists z \in W.\ (z_0, z) \in I\}$; then the accepting alphabet is

$$F = \begin{cases} F_0 \cup \{\sharp\} & \text{if } z_0 \in W; \\ F_0 & \text{if } z_0 \notin W. \end{cases}$$

Let us consider any node of the binary tree computed according to f and g from a string $w\sharp^\ell$ being fed at some level n, with $w \in \Lambda^*$ and $\ell = 2^n - |w|$. Assume $I \neq \sharp$ and let $\sigma(I)$ be the substring of w corresponding to the ordered sequence of leaves being descendants of I. It is easy to see that I contains precisely the pairs of states (z_1, z_2) such that there exists a path in A labelled by the symbols in $\sigma(I)$; i.e., given $\sigma(I) = a_1 a_2 \dots a_k$, then $(z_1, z_k) \in I$ iff there exist states $z_1, z_2, \dots, z_{k+1}$ such that for all $i < k$ we have $z_{i+1} \in t(z, a_i)$.

Assume now an arbitray multiset $\mu \in L(A)$. Then there exist a string $w = a_1 \dots a_k \in \Lambda^*$ with $\phi(w) = \mu$ and configurations $(z_0, \mu_0) \vdash (z_1, \mu_1) \vdash \cdots \vdash (z_k, \mu_{k+1})$ such that for all i we have $\mu_{i+1} = \mu_i \setminus \{a_i\}$, and $z_k \in W$ is an accepting state. For n such that $k \leq 2^n$ consider the node $\widetilde{I} = \overline{f}^n(\overline{g}(w\sharp^{2^n-k}))$. We have $(z_0, z_k) \in \widetilde{I}$ and, since $x_k \in W$, we have $\widetilde{I} \in F$. Therefore $\mu \in L(K)$.

Assume $\mu \in L(K)$. Then there exist a n such that $k \leq 2^n$ and a string $w = a_1 \dots a_k \in \Lambda^*$ with $\phi(w) = \mu$, a node $\widetilde{I} = \overline{f}^n(\overline{g}(w\sharp^{2^n-k})) \in F$ and configurations $(z_0, \mu_0) \vdash (z_1, \mu_1) \vdash \cdots \vdash (z_k, \mu_{k+1})$ such that for all i we have $\mu_{i+1} = \mu_i \setminus \{a_i\}$. From $\widetilde{I} \in F$ we infer $z_k \in W$. Therefore $\mu \in L(A)$. We conclude $L(K) = L(A)$.

3.2 Regular MBSTAs

We introduce variants of BSTAs and MBSTAs, called Regular BSTAs (RBSTAs) and Regular MBSTAs (RMBSTAs) respectively, in which some regularity conditions are assumed. Regular MBSTA will be used as an intermediate formalism to ease the construction of a P system which accepts the same multiset language as a given MBSTA.

Definition 9 (Regular (M)BSTA). *A* Regular BSTA *(resp.* Regular MBSTA*) is a BSTA $C = (\Lambda, Q, F, f, g)$ (resp. MBSTA $M = (\Lambda, Q, F, f, g)$) such that Q can be partitioned into the sets $Q_o = \{q_1, \dots, q_h\}$ of plain symbols, and $Q_o^\sharp = \{q_{h+1}^\sharp, \dots, q_n^\sharp\}$ of tagged symbols, and the following regularity conditions for the functions f and g are satisfied:*

- $g(x) \in Q_o$, *for all* $x \in \Lambda$;
- $f(q_1, q_2) \in Q_o$, *for all* $q_1, q_2 \in Q_o$;
- $f(q_1, \sharp) \in Q_o^\sharp$, *for all* $q_1 \in Q_o \cup Q_o^\sharp$;
- $f(q_1, q_2) \in Q_o^\sharp$, *for all* $q_1 \in Q_o$, $q_2 \in Q_o^\sharp$.

Theorem 4. *Let* $K = (\Lambda, Q, F, f, g)$ *(resp.* $M = (\Lambda, Q, F, f, g)$*) be a stable BSTA (resp. stable MBSTA). Then we can effectively construct a stable RBSTA* $K' = (\Lambda, Q', F', f', g)$ *(resp. stable RMBSTA* $M' = (\Lambda, Q', F', f', g)$*) such that*

- $L(K) = L(K')$ *(resp.* $L(M) = L(M')$*);*
- Q' *can be partitioned into the sets* $Q_o = Q$ *and* $Q_o^\sharp = \{x^\sharp \mid x \in Q\}$.

Proof. Let $F' = F \cup \{x^\sharp \mid x \in F\}$, and let f' be defined as follows:

- for $x, y, z \in Q_o$, if $f(x, y) = z$ then $f'(x, y) = z$ and $f'(x, y^\sharp) = z^\sharp$;
- for $x, z \in Q_o$, if $f(x, \sharp) = z$ then $f'(x, \sharp) = z^\sharp$ and $f'(x^\sharp, \sharp) = z^\sharp$;
- $f'(\sharp, \sharp) = \sharp$.

We prove that the thesis holds for BSTAs and MBSTAs by showing that for all strings $w \in \Lambda^*$ and n such that $|w| \leq 2^n$, w can be accepted by K (resp. M) at level n iff w can be accepted by K' (resp. M') at level n.

Let $\ell = 2^n - |w|$, and let $u_i = \left(\overline{f}\right)^i (\overline{g}(w\sharp^\ell))$, $u'_i = \left(\overline{f'}\right)^i (\overline{g}(w\sharp^\ell))$, for $i \in \{0, \ldots, n\}$. Note that $|u_i| = |u'_i| = 2^{n-i}$. Moreover, let us define the function $\theta : (Q_o \cup Q_o^\sharp \cup \{\sharp\})^+ \to (Q_o \cup \{\sharp\})^+$ as follows:

$$\begin{aligned} &\theta(x) = \theta(x^\sharp) = x && x \in Q_o \\ &\theta(\sharp) = \sharp \\ &\theta(x_1 \ldots x_k) = \theta(x_1) \ldots \theta(x_k) \end{aligned}$$

It suffices to show that for all $i \in \{0, \ldots, n\}$, $u_i = \theta(u'_i)$, which can be proved by induction. As regards the base case, $u_0 = \theta(u'_0)$, since $u'_0 = u_0 \in (Q_o \cup \{\sharp\})^+$.

In the inductive case, assume $u_i = \theta(u'_i)$; we need to prove that $u_{i+1} = \theta(u'_{i+1})$. By the definitions of g and f', for all i, each string u'_i is of the form $q_1 \ldots q_k \sharp \ldots \sharp$ with $q_i \in Q_o$ for all $i < k$ and $q_k \in Q_o \cup Q_o^\sharp$. As a consequence, only either one of the following two cases may occur (in the following, we assume $\overline{f}(\epsilon) = \epsilon$):

- *Case* $u'_i = \alpha v$, *with* $\alpha \in Q_o^*$, $v \in \{\sharp\}^*$. It holds that $u_i = \theta(u'_i) = u'_i$.
  If $|\alpha|$ is even, then $u'_{i+1} = \overline{f'}(u'_i) = \overline{f}(u_i) = u_{i+1}$, and hence $\theta(u'_{i+1}) = \theta(u_{i+1}) = u_{i+1}$.
  If $|\alpha|$ is odd, let $u'_i = \alpha' x \sharp v'$, with $\alpha' \in Q_o^*$, $x \in Q_o$, $v' \in \{\sharp\}^*$. Then $u'_{i+1} = \overline{f'}(\alpha' x \sharp v') = \overline{f}(\alpha') f'(x, \sharp) \overline{f}(v') = \overline{f}(\alpha') \left(f(x, \sharp)\right)^\sharp \overline{f}(v')$, and hence $\theta(u'_{i+1}) = \overline{f}(\alpha')\, f(x, \sharp)\, \overline{f}(v') = \overline{f}(u_i) = u_{i+1}$.
- *Case* $u'_i = \alpha x^\sharp v$, *with* $\alpha \in Q_o^*$, $x \in Q_o$, $v \in \{\sharp\}^*$. It holds that $u_i = \theta(u'_i) = \alpha x v$.

If $|\alpha|$ is odd, let $u'_i = \alpha' y x^\sharp v$, with $\alpha' \in Q_o^*$, $y \in Q_o$. Then $u'_{i+1} = \overline{f'}(\alpha' y x^\sharp v) = \overline{f}(\alpha') f'(y, x^\sharp) \overline{f}(v) = \overline{f}(\alpha') \, (f(y,x))^\sharp \, \overline{f}(v))$; hence $\theta(u'_{i+1}) = \overline{f}(\alpha') \, f(y,x) \, \overline{f}(v) = \overline{f}(u_i) = u_{i+1}$.
If $|\alpha|$ is even, let $u'_i = \alpha x^\sharp \sharp v'$, with $v' \in \{\sharp\}^*$. Then $u'_{i+1} = \overline{f'}(\alpha x^\sharp \sharp v') = \overline{f}(\alpha) f'(x^\sharp, \sharp) \overline{f}(v') = \overline{f}(\alpha) \, (f(x,\sharp))^\sharp \, \overline{f}(v))$ (recall that $f(x,\sharp) \neq \sharp$ by definition). Hence $\theta(u'_{i+1}) = \overline{f}(\alpha) \, f(x,\sharp) \, \overline{f}(v') = \overline{f}(u_i) = u_{i+1}$.

4 P Systems

P systems [23] are a bio-inspired computational formalism, where the behaviour is driven by evolution rules applied to multisets of objects. A P system is composed of a hierarchy of membranes, each containing a multiset of objects and a set of evolution rules. Evolution rules describe how the objects of the system evolve, for example they can be used to describe chemical reactions, i.e. rules in which some objects interact and, as a result, they are transformed into some other objects. Given a membrane m, its evolution rules in the set R_m can be applied only to the objects contained in the same membrane, and not in any other membrane. Many versions of P systems have been defined [2,4,7]. Formal semantics of different versions of P systems are presented in [1,3,5,8,10,11].

An evolution rule is of the form $u \to v$, where u and v are multisets whose elements are called *reactants* and *products*, respectively. When a rule is applied, the reactants are removed from the membrane and the products are added to the target membrane, which could be a different membrane than the one in which the rule is applied. Membranes are univocally labelled with natural numbers. Given a membrane m, the products of a rule associated with m are described by a multiset of (possibly) labelled objects having the following forms: a, meaning that the object a is added to the same membrane m; a_{out}, meaning that the object a is to be sent out of the membrane; a_{in_x}, meaning that the object a is to be sent into the child membrane labelled by x.

An evolution rule is said to be *cooperative* if it contains more than one reactant, otherwise the rule is called *non-cooperative*. This naming is also extended to P system models, that is, a *non-cooperative* P system is such that all its rules are non-cooperative, otherwise it is a *cooperative* P system.

A formal definition of P systems follows.

Definition 10. *A* P system *is a tuple* $\Pi = (V, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n)$ *where:*

- V *is a finite* alphabet *whose elements are called* objects*;*
- $\mu \subset \mathbb{N} \times \mathbb{N}$ *describes the tree-structure of membranes, where* $(i,j) \in \mu$ *denotes that the membrane labelled by* j *is contained in the membrane labelled by* i*;*
- w_i*, with* $1 \leq i \leq n$*, are strings from* V^* *representing multisets over* V *associated with membranes* $1, 2, \ldots, n$ *of* μ*;*
- R_i*, with* $1 \leq i \leq n$*, are finite sets of* evolution rules *associated with membranes* $1, 2, \ldots, n$ *of* μ*.*

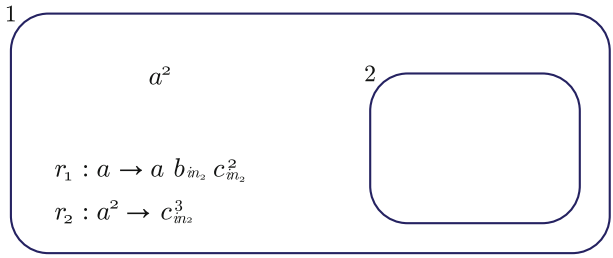

Fig. 3. An example of P system model

A characteristic of P systems is the way in which rules are applied in each step, namely with *maximal parallelism*. In each step, evolution rules are applied in a maximal non-deterministic way in all membranes, that is, in each membrane, a multiset of rules is selected non-deterministically to consume the membrane objects, in such a way that no other rule can be applied to the objects which are not involved in any rule application.

A *configuration* of a P system is given by an association of its membranes with multisets of objects. The multisets of objects $w_1, \ldots, w_n$ in the definition of a P system Π represent the initial configuration of Π. A *computation* is a sequence of transitions between configurations of a given P system Π starting from the initial configuration. Each transition of a computation describes a maximally parallel step. A computation is *successful* if and only if it reaches a configuration in which no rule is applicable. The result of a successful computation can be defined in different ways. Unsuccessful computations are those computations which never halt, thus yielding no result.

Example 2. Figure 3 depicts a P system with two membranes, labelled 1 and 2. The rules r_1 and r_2 are associated with membrane 1, while membrane 2 has no rules associated with it. An application of rule $r_1 = a \to a\, b_{in_2}\, c^2_{in_2}$ causes a copy of object b and two copies of object c to be sent into the inner membrane 2. The object a is still present after the application, since it appears in the right-hand part of the rule. Rule $r_2 = a^2 \to c^3_{in_2}$, instead, can be applied to a pair of objects a, and results in sending three copies of the object c into membrane 2. The initial state, as depicted, contains two copies of object a in membrane 1, and no objects in membrane 2.

At the first step, either rule r_1 or r_2 is applied. In fact, both the rules are enabled, since their reactants are present in the membrane. Actually, if r_1 is applied to an object a, then the maximality requires it to be applied also to the other copy of a. This application sends the objects b^2c^4 into membrane 2. The objects contained in membrane 1 remain aa after the application, therefore the double application of rule r_1 can be repeated in the subsequent step. Whenever rule r_2 is applied, it causes the two copies of a in membrane 1 to disappear, thus terminating the computation. In such a case, the objects ccc are sent into membrane 2. Therefore, any computation of this P system is composed of a

sequences of steps in which only r_1 is applied (twice per step), followed by a last step in which rule r_2 is applied once. Therefore, whenever the P system terminates, membrane 2 contains a multiset of objects $b^k\, c^{2k+3}$, for some $k \geq 0$.

A P system can be used either as an acceptor or as a generator of a multiset language over an alphabet Λ [9]. In the first case, a multiset over Λ is inserted in the outmost membrane of the P system and the result of its computations says whether such a multiset belongs to the multiset language accepted by the P system or not. In the second case the P system has a fixed initial configuration and can give as results (possibly in a non-deterministic way) all the possible multisets belonging to a given multiset language.

In order to investigate the relationship between (R)MBSTAs and P systems let us recall from [9] the definition of P system used as language acceptor. From [6] it follows that any P system Π can be translated into an equivalent P system Π' having a (flat) membrane structure that consists only of one membrane. Hence, we consider in this paper only *flat acceptor P Systems.*

Definition 11. *A* flat acceptor P system *over an alphabet Λ is a P system $\Pi = (\Lambda \cup \mathcal{C} \cup \{\mathbf{T}\}, \emptyset, \mu_1, R_1)$, where:*

- *$\mathcal{C}$ is a set of* control objects *such that $\Lambda \cap \mathcal{C} = \emptyset$;*
- *$\mathbf{T}$ is a special object not contained in $\Lambda \cup \mathcal{C}$;*
- *μ_1 is a multiset of objects in $\mathcal{C}$.*

A multiset μ of objects over Λ is accepted by Π iff, by adding μ to μ_1, then a final configuration can be reached with $\mathbf{T}$ occurring in the membrane.

The multiset language accepted by a flat acceptor P system Π is denoted $Ps(\Pi)$ (as Parikh set).

4.1 Simulating RMBSTAs with P Systems

Let $R = (\Lambda, Q, F, f, g)$ be a RMBSTA. Recall that Q can be partitioned into the sets $Q_o = \{q_1, \ldots, q_h\}$ of plain symbols, and $Q_o^\sharp = \{q_{h+1}^\sharp, \ldots, q_n^\sharp\}$ of tagged symbols. Moreover, assume, without loss of generality, that $\Lambda \cap Q = \emptyset$.

Let us consider two fresh symbols: $\diamondsuit, \mathbf{F} \notin \Lambda \cup Q$. Symbol $\diamondsuit$ is a special *trap* symbol that will be used to denote invalid computations, while $\mathbf{F}$ (dual to $\mathbf{T}$) will be used to denote computations that do not accept the input multiset.

We construct an acceptor P system $\Pi_R = (\Lambda \cup \mathcal{C} \cup \{\mathbf{T}\}, \emptyset, \mu_1, R_1)$ where $\mathcal{C} = Q \cup \{\mathbf{F}, \sharp, \diamondsuit\}$, $\mu_1 = \{\!|\sharp|\!\}$, and R_1 is composed of the rules shown in Figure 4.

Theorem 5. *Let $R = (\Lambda, Q, F, f, g)$ be a RMBSTA, and let $\Pi_R = (\Lambda \cup \mathcal{C} \cup \{\mathbf{T}\}, \emptyset, \mu_1, R_1)$ be the corresponding acceptor P system. Then, for any multiset $\mu \in \mathcal{M}(\Lambda)$ the following implications hold:*

1. *μ is accepted by R at level $n \implies$ μ can accepted by Π_R in no more than $n + 2$ steps;*

$x \to y$	if $y = g(x)$, $x \in \Lambda$
$xy \to z$	if $z = f(x, y)$; $x \in Q_o$, $y, z \in Q_o \cup Q_o^\sharp$
$x\sharp \to z\sharp$	if $z = f(x, \sharp)$; $x, z \in Q_o \cup Q_o^\sharp$
$x\sharp \to \mathbf{T}$	if $x \in F \cap (Q_o \cup Q_o^\sharp)$
$x\sharp \to \mathbf{F}$	if $x \in (Q_o \cup Q_o^\sharp) \setminus F$
$\sharp \to \mathbf{T}$	if $\sharp \in F$
$\sharp \to \mathbf{F}$	if $\sharp \notin F$
$\sharp \to \sharp$	
$x \to \diamondsuit$	if $x \in Q_o \cup Q_o^\sharp$
$x\mathbf{T} \to \diamondsuit$	if $x \in Q_o \cup Q_o^\sharp$
$x\mathbf{F} \to \diamondsuit$	if $x \in Q_o \cup Q_o^\sharp$
$\diamondsuit \to \diamondsuit$	

Fig. 4. Evolution rules of a P system which simulates a RMBSTA

2. *μ is accepted by Π_R in 1 step $\implies$ μ is accepted by R at level 0;*
3. *μ is accepted by Π_R in $n+2$ steps, with $n \geq 0$ $\implies$ μ is accepted by R at level n.*

As a consequence of these implications we have $L(R) = Ps(\Pi_R)$.

Proof. Item 1. Let us consider an accepting computation of the RMBSTA R, starting from a string $w \in \Lambda^*$ such that $\phi(w) = \mu$. Let $\ell = 2^n - |w|$, and $w_h = \overline{f}^{(n-h)}(\overline{g}(w\sharp^\ell))$, for $h \in \{0, \ldots, n\}$. It holds $w_0 \in F$.

If $|w| = 0$, then $\forall h \in \{0, \ldots, n\}$. $w_h \in \{\sharp\}^+$. Since μ is accepted, $w_0 = \sharp \in F$. The initial configuration of Π_R contains only $\sharp$, from which the transition $\sharp \longrightarrow \mathbf{T}$ can be performed, reaching a final accepting configuration.

Assume $|w| > 0$. For all $h \in \{0, \ldots, n\}$, $w_h = w'_h\sharp \ldots \sharp$, with $|w_h| = 2^h$, $w'_h = q_1 \ldots q_k$, $\forall j < k. q_j \in Q_o$, and $q_k \in Q_o \cup Q_o^\sharp$. Each application of the processing function f may only involve pairs of symbols (x, y) from one of these sets: $Q_o \times Q_o$; $Q_o \times Q_o^\sharp$; $Q_o \times \{\sharp\}$; $\{\sharp\} \times \{\sharp\}$. Note that, for the above cases (except for $f(\sharp, \sharp)$), Π_R contains evolution rules which simulate the behaviour of f. In fact, for the first two cases, Π_R contains an evolution rule $xy \to z$, with $z = f(x, y)$, while in the third case a rule $x\sharp \to z\sharp$, with $z = f(x, \sharp)$, is present. Since each symbol in w_h is used exactly in one application of f, according to the definition of the evolution rules of the P system Π_R, the resulting string w_{h+1} is such that the transition $\phi(w'_h\sharp) \longrightarrow \phi(w'_{h+1}\sharp)$ can be performed by Π_R. Therefore, the P system can perform, from the initial state $\phi(w\sharp)$, the sequence of $n+2$ transitions $\phi(w\sharp) \longrightarrow \phi(w'_n\sharp) \longrightarrow \cdots \longrightarrow \phi(w'_0\sharp) \longrightarrow \phi(\mathbf{T})$, where the first transition corresponds to the application of function g, and the last transition is made possible because $w'_0 = w_0 \in F$.

Item 2. In this case, only rule $\sharp \to \mathbf{T}$ may have been applied to the initial state $\mu \cup \{\!|\sharp|\!\}$, for the input multiset $\mu = \emptyset \in \mathcal{M}(\Lambda)$, which implies $\sharp \in F$; hence μ can be accepted by R at level 0. Otherwise, if $|\mu| > 0$, then the P system may not have reached a final accepting configuration in one step. In fact, the only way to yield $\mathbf{T}$ is to apply rule $\sharp \to \mathbf{T}$ while, at the same time, rules $x \to y$ with $y = g(x)$ are applied to each symbol in μ, thus yielding a non-final configuration.

Item 3. Let us consider an accepting computation $\gamma_0 \to \gamma_1 \to \cdots \to \gamma_{n+2}$, $n \geq 0$. It holds $\gamma_{n+2} = \{\!|\mathbf{T}|\!\}$, since no symbol in $Q \cup \{\mathbf{F}, \sharp, \Diamond\}$ may be present (note that no more than one of either $\mathbf{T}$ or $\mathbf{F}$ may be present in a configuration). In turn, either $\gamma_{n+1} = \{\!|\sharp|\!\}$ or $\gamma_{n+1} = \{\!|x\sharp|\!\}$. The former case implies that $\forall i \in \{0, \ldots, n\}$. $\gamma_i = \{\!|\sharp|\!\}$, since the only rule which may have been applied is $\sharp \to \sharp$. Hence $\mu = \emptyset$, which can be accepted by R at any level $n \geq 0$.

Let us consider the case $\gamma_{n+1} = \{\!|x\sharp|\!\}$, with $x \in F \cap Q$. Let $\widetilde{Q} = Q_o^* \cup (Q_o^* . Q_o^\sharp)$, namely $\widetilde{Q}$ contains all strings composed of symbols from Q_o, possibly ending with a symbol from $Q_o^\sharp$. We prove that $\forall i \in \{0, \ldots, n\}$, if $\gamma_{n+1-i} = \phi(\alpha\sharp)$ with $\alpha \in \widetilde{Q}$, $|\alpha| \leq 2^i$, and $\overline{f}^i(\alpha\sharp^\ell) \in F \cap Q$ where $\ell = 2^i - |\alpha|$, then $\gamma_{n-i} = \phi(\beta\sharp)$ with $\beta \in \widetilde{Q}$, $|\beta| \leq 2^{i+1}$, and $\overline{f}^i(\alpha\sharp^\ell) = \overline{f}^{i+1}(\beta\sharp^{\ell'})$, where $\ell' = 2^{i+1} - |\beta|$.

Assume $\gamma_{n-i} = \phi(\alpha\sharp)$. Due to the maximally-parallel semantics of P systems, $\gamma_{n-i-1} = \phi(\beta\sharp)$. In fact, $\phi(\alpha\sharp)$ may only have been obtained by either (i) $|\alpha|$ applications of rule $xy \to z$ yielding each symbol in α, and one application of $\sharp \to \sharp$; or (ii) $|\alpha| - 1$ applications of rule $xy \to z$ yielding each *but one* symbol in α, and one application of $x\sharp \to z\sharp$. Moreover, note that $|\beta|_{Q_o^\sharp} \leq |\alpha|_{Q_o^\sharp} \leq 1$. Therefore, in both cases, $\alpha\sharp^\ell = \overline{f}(\beta\sharp^{\ell'})$, and hence $\overline{f}^i(\alpha\sharp^\ell) = \overline{f}^{i+1}(\beta\sharp^{\ell'})$.

Note that, at the beginning, $\gamma_0 = \mu \cup \{\!|\sharp|\!\}$, hence $\gamma_1 = \{\!|g(x) \mid x \in \mu|\!\} \cup \{\!|\sharp|\!\}$. It follows that, since $\gamma_{n+1} = \{\!|x\sharp|\!\}$ with $x \in F \cap Q$, then $\overline{f}^n(w\sharp^\ell) \in F \cap Q$, for some string $w \in \widetilde{Q}$ such that $\phi(w\sharp) = \gamma_1$. Therefore $w = \overline{g}(w')$, for some w' such that $\phi(w') = \mu$, from which we conclude $\overline{f}^n(\overline{g}(w')\sharp^\ell) = \overline{f}^n(\overline{g}(w'\sharp^\ell)) \in F \cap Q$.

Finally, note that Item 1 implies $L(R) \subseteq Ps(\Pi_R)$, while *Items 2* and *3* imply $Ps(\Pi_R) \subseteq L(R)$ (note that the initial configuration of Π_R does not contain $\mathbf{T}$, hence it needs at least one step to reach a final accepting configuration).

5 Conclusions

In this paper we have related systolic automata and P systems. We have extended systolic automata to accept multisets of symbols, by introducing Multiset Binary Systolic Tree Automata (MBSTAs). In particular, we have shown how an equivalent variant of MBSTAs (called Regular MBSTAs) can be easily translated into cooperative P systems.

References

1. Andrei, O., Ciobanu, G., Lucanu, D.: A rewriting logic framework for operational semantics of membrane systems. Theoretical Computer Science **373**(3), 163–181 (2007)

2. Barbuti, R., Caravagna, G., Maggiolo-Schettini, A., Milazzo, P.: P systems with endosomes. International Journal of Computers, Communications and Control **4**(3), 214–223 (2009)
3. Barbuti, R., Maggiolo-Schettini, A., Milazzo, P., Pardini, G.: Simulation of spatial P system models. Theoretical Computer Science **529**, 11–45 (2014)
4. Barbuti, R., Maggiolo-Schettini, A., Milazzo, P., Pardini, G., Tesei, L.: Spatial P systems. Natural Computing **10**(1), 3–16 (2011)
5. Barbuti, R., Maggiolo-Schettini, A., Milazzo, P., Tini, S.: Compositional semantics and behavioral equivalences for P systems. Theoretical Computer Science **395**(1), 77–100 (2008)
6. Barbuti, R., Maggiolo-Schettini, A., Milazzo, P., Tini, S.: A P systems flat form preserving step-by-step behaviour. Fundamenta Informaticae **87**(1), 1–34 (2008)
7. Barbuti, R., Maggiolo-Schettini, A., Milazzo, P., Tini, S.: P systems with transport and diffusion membrane channels. Fundamenta Informaticae **93**(1–3), 17–31 (2009)
8. Barbuti, R., Maggiolo-Schettini, A., Milazzo, P., Tini, S.: Compositional semantics of spiking neural P systems. Journal of Logic and Algebraic Programming **79**(6), 304–316 (2010)
9. Barbuti, R., Maggiolo-Schettini, A., Milazzo, P., Tini, S.: Membrane systems working in generating and accepting modes: Expressiveness and encodings. In: Gheorghe, M., Hinze, T., Păun, G., Rozenberg, G., Salomaa, A. (eds.) CMC 2010. LNCS, vol. 6501, pp. 103–118. Springer, Heidelberg (2010)
10. Barbuti, R., Maggiolo-Schettini, A., Milazzo, P., Tini, S.: An overview on operational semantics in membrane computing. International Journal of Foundations of Computer Science **22**(1), 119–131 (2011)
11. Busi, N.: Using well-structured transition systems to decide divergence for catalytic P systems. Theoretical Computer Science **372**(2–3), 125–135 (2007)
12. Csuhaj-Varjú, E., Martín-Vide, C., Mitrana, V.: Multiset Automata. In: Calude, C.S., Păun, G., Rozenberg, G., Salomaa, A. (eds.) Multiset Processing. LNCS, vol. 2235, pp. 69–83. Springer, Heidelberg (2001)
13. Culik II, K., Gruska, J., Salomaa, A.: On a family of L languages resulting from systolic tree automata. Theoretical Computer Science **23**(3), 231–242 (1983)
14. Culik II, K., Gruska, J., Salomaa, A.: Systolic automata for VLSI on balanced trees. Acta Informatica **18**(4), 335–344 (1983)
15. Culik II, K., Gruska, J., Salomaa, A.: Systolic trellis automata I. International Journal of Computer Mathematics **15**(1–4), 195–212 (1984)
16. Culik II, K., Gruska, J., Salomaa, A.: Systolic trellis automata II. International Journal of Computer Mathematics **16**(1), 3–22 (1984)
17. Culik II, K., Salomaa, A., Wood, D.: Systolic tree acceptors. RAIRO-Theoretical Informatics and Applications-Informatique Théorique et Applications **18**(1), 53–69 (1984)
18. Fachini, E., Gruska, J., Maggiolo-Schettini, A., Sangiorgi, D.: Simulation of systolic tree automata on trellis automata. International Journal of Foundations of Computer Science **1**(2), 87–110 (1990)
19. Fachini, E., Maggiolo-Schettini, A., Resta, G., Sangiorgi, D.: Nonacceptability criteria and closure properties for the class of languages accepted by binary systolic tree automata. Theoretical Computer Science **83**(2), 249–260 (1991)
20. Fachini, E., Maggiolo-Schettini, A., Sangiorgi, D.: Comparisons among classes of Y-tree systolic automata. In: Rovan, B. (ed.) Mathematical Foundations of Computer Science 1990. LNCS, vol. 452, pp. 254–260. Springer, Berlin Heidelberg (1990)

21. Fachini, E., Maggiolo-Schettini, A., Sangiorgi, D.: Classes of systolic Y-tree automata and a comparison with systolic trellis automata. Acta Informatica **29**(6/7), 623–643 (1992)
22. Gruska, J., Monti, A., Napoli, M., Parente, D.: Succinctness of descriptions of SBTA-languages. Theoretical Computer Science **179**(1–2), 251–271 (1997)
23. Păun, G.: Membrane Computing: An Introduction. Springer, Heidelberg (2002)
24. Rozenberg, G., Salomaa, A.: The mathematical theory of L systems. Academic Press (1980)