On Languages Accepted by
P/T Systems Composed of joins

Pierluigi Frisco
School of Mathematical and Computer Sciences – Heriot-Watt University
EH14 4AS Edinburgh – UK
pier@macs.hw.ac.uk

Oscar H. Ibarra
Department of Computer Science – University of California
Santa Barbara – CA 93106 – USA
ibarra@cs.ucsb.edu

Recently, some studies linked the computational power of abstract computing systems based on multiset rewriting to models of Petri nets and the computation power of these nets to their topology. In turn, the computational power of these abstract computing devices can be understood by just looking at their topology, that is, information flow.

Here we continue this line of research introducing \( J \) languages and proving that they can be accepted by place/transition systems whose underlying net is composed only of joins. Moreover, we investigate how \( J \) languages relate to other families of formal languages. In particular, we show that every \( J \) language can be accepted by a \( \log n \) space-bounded non-deterministic Turing machine with a one-way read-only input. We also show that every \( J \) language has a semilinear Parikh map and that \( J \) languages and context-free languages (CFLs) are incomparable. For example, the CFL, \( \{ x^{#x^R} \mid x \in \{0,1\}^+ \} \), is not a \( J \) language, but there are non-CFLs that are \( J \) languages.

1 Introduction

In [1] a study on models of Petri nets linking their topological structure to the families of languages they can accept/generate was started. In particular this study concentrated on Petri nets whose topological structure (that is, their underlying net) was composed only of specific building blocks (motifs), that is, little nets connected to each other.

The following question was raised and partially answered in [1]: What is the computational power of networks composed of specific building blocks? The answer to this question was pursued in [2, 3]. As shown in [1,2,3] such research can help the study of the computational power of systems based on multiset rewriting. Given \( S_1 \), a formal system based on multiset rewriting, the study of its computational power is normally done by proving that it can be simulated by another formal system, say \( S_2 \), of known computational power. If \( S_2 \) can also simulate \( S_1 \), then we can say that the two systems have equivalent computational power. There is a new way to analyse the computational power of \( S_1 \) [1]. This new way depends on how the system stores and manipulates information and it deduces the computational power of \( S_1 \). The way information is stored and manipulated by systems based on multiset rewriting can be easily represented with Petri nets. From here then the link between the computational power of formal system based on multiset rewriting and the topological structure of Petri nets.

As indicated in [1], we have not been able to find in the Petri net literature work that has been done along the lines of what we propose.
In the present paper we continue to answer the above question introducing $J$ languages and proving that they can be accepted by place/transition systems (a model of Petri nets) whose underlying net is composed only of joins (a kind of building block). We study how $J$ languages relate to other families of formal languages and show how these relationships allow us to derive the computational power of a model of P systems.

Because of page limit restrictions, several proofs have been omitted.

2 Basic definitions

We assume the reader to have familiarity with basic concepts of formal language theory [6], and in particular with the topic of place/transition systems [11,10]. In this section we recall particular aspects relevant to our presentation.

We denote by $\mathbb{N}_1 = \{1, 2, \ldots \}$ the set of natural numbers while $\mathbb{N} = \mathbb{N}_1 \cup \{0\}$.

Definition 1. A place/transition system (P/T system) is a tuple $N = (P, T, F, W, K, C_{in})$ where:

i) $(P, T, F)$ is a net:
   1. $P$ and $T$ are sets with $P \cap T = \emptyset$;
   2. $F \subseteq (P \times T) \cup (T \times P)$;
   3. for every $t \in T$ there exist $p, q \in P$ such that $(p, t), (t, q) \in F$;

ii) $W : F \rightarrow \mathbb{N}_1$ is a weight function;

iii) $K : P \rightarrow \mathbb{N}_1 \cup \{+\infty\}$ is a capacity function;

iv) $C_{in} : P \rightarrow \mathbb{N}$ is the initial configuration (or initial marking).

We consider P/T systems in which the weight function returns always 1 and the capacity function returns always $+\infty$. We introduced these functions in the previous definition for consistency with the (for us) standard definition of P/T systems and for consistency with the definition in [1, 2, 3]. We follow the very well established notations (places are represented by empty circles, transitions by full rectangle’s, tokens by bullets, etc.), concepts and terminology (configuration, input set, output set, sequential configuration graph, etc.) relative to P/T systems [11,10].

In this paper we consider P/T systems as accepting computing devices. The definition of accepting P/T systems includes the indication of a set $P_{in} \subset P$ of input places, one initial place $p_{init} \in P \setminus P_{in}$ and one final place $p_{fin} \in P \setminus P_{in}$. The places in $P \setminus P_{in}$ are called work places.

An accepting P/T system $N$ with input $C_{in}$ is denoted by $N(C_{in}) = (P, T, F, W, K, P_{in}, p_{init}, p_{fin})$, where $C_{in} : (P_{in} \cup \{p_{init}\}) \rightarrow \mathbb{N}$, $C_{in}(p_{init}) = 1$, is the initial configuration of the input places. So, in the initial configuration some input places can have tokens and the work place $p_{init}$ has one token. All the remaining places are empty in the initial configuration. A configuration $C_{fin} \in \mathbb{C}_N$, the set of all reachable configurations of $N$, is said to be final (or dead state) if no firing is possible from $C_{fin}$. We say that a P/T system $N(C_{in}) = (P, T, F, W, K, P_{in}, p_{init}, p_{fin})$ with $P_{in} = \{p_{in,1}, \ldots, p_{in,k}\}$, $k \in \mathbb{N}_1$, accepts the vector $(C_{in}(p_{in,1}), \ldots, C_{in}(p_{in,k}))$ if in the sequential configuration graph of $N(C_{in})$ there is a final configuration $C_{fin}$ such that:
\( C_{\text{fin}}(p_{\text{fin}}) > 0; \)

- there is at least one path from \( C_{\text{in}} \) to \( C_{\text{fin}}; \)
- no other configuration \( D \) in the paths from \( C_{\text{in}} \) to \( C_{\text{fin}} \) is such that \( D(p_{\text{fin}}) > 0. \)

The set of vectors accepted by \( N \) is denoted by \( N^k(N) \) and it is composed by the vectors

\[
(C_{\text{in}}(p_{\text{in},1}), \ldots, C_{\text{in}}(p_{\text{in},k}))
\]

accepted by \( N \). The just given definition of (vector) acceptance for P/T systems is new in Petri nets. Normally, the language generated by Petri nets is given by the concatenation of the labels in firing sequences. We discuss this point in Section 7.

As in [2] we call the nets join and fork building blocks, see Figure 1, where the places in each building block are distinct.

Also from [2] we take:

**Definition 2.** Let \( x, y \in \{ \text{join}, \text{fork} \} \) be building blocks and let \( \tilde{t}_x \) and \( \tilde{t}_y \) be the transitions present in \( x \) and \( y \) respectively.

We say that \( y \) comes after \( x \) (or \( x \) is followed by \( y \) or \( x \) and \( y \) are in sequence) if \( \tilde{t}_x \cap \tilde{t}_y \neq \emptyset \) and \( \tilde{t}_x \cap \tilde{t}_y = \emptyset \). We say that \( x \) and \( y \) are in parallel if \( \tilde{t}_x \cap \tilde{t}_y \neq \emptyset \) and \( \tilde{t}_x \cap \tilde{t}_y = \emptyset \).

We say that a net is composed of building blocks (it is composed of \( x \)) if it can be defined by building blocks (it is defined by \( x \)) sharing places but not transitions. So, for instance, to say that a net is composed of joins means that the only building blocks present in the net are join.

In this paper we consider accepting P/T systems (in which the weight functions returns always 1 and the capacity function returns always +\( \infty \)) whose underlying net is composed of joins. Moreover, if \( N = (P, T, F, W, K, P_{\text{init}}, p_{\text{fin}}) \) is such a P/T systems, then for each \( t \in T, \tilde{t} \in (P_{\text{in}} \times P \setminus P_{\text{in}}) \) and \( t^* \in P \setminus P_{\text{in}} \). Informally, this means that for each transition \( t \in T \) the input set is given by an input place and a work place, while the output set is a work place. We call these systems J P/T systems.

It should be clear that J P/T systems are a normal form of accepting P/T systems: for each accepting P/T system there is a J P/T systems accepting the same language. Such J P/T systems has, eventually, more places and transitions than the original P/T system. For instance, let us assume that the net depicted in Figure 2a is part of the net underlying an accepting P/T system \( N \) with \( P \) as set of places, \( P_{\text{in}} \subset P \) as set input places and \( T \) as set of transitions. The net depicted in Figure 2b belongs to a J P/T system \( N_J \) with \( P \cup \{w_1', w_2'\} \) as set of places, \( P_{\text{in}} \) as set of input places and \( T \cup \{t'_1\} \) as set of transitions. The two nets in Figure 2 can be regarded as similar in the sets of vectors they accept.
On Languages Accepted by P/T Systems Composed of joins

3 J languages and P/T systems

In this section we prove the main result of the present paper. In order to do this, we need to introduce a new family of formal languages.

Definition 3. Let $\Sigma$ be an alphabet, then:

- $\varepsilon$ (the empty string) is a J expression;
- for each $v \in \Sigma$, $v$ is a J expression;
- if $\alpha$ and $\beta$ are J expressions, then $(\alpha \cup \beta)$ is a J expression (union, in this case $\alpha$ and $\beta$ are called union-terms);
- if $\alpha$ and $\beta$ are J expressions such that $\alpha, \beta \neq \varepsilon$ but they can contain $\varepsilon$ (e.g., $\alpha = a \cup \varepsilon$), then $(\alpha\beta)$ (concatenation), and $(\alpha^+)$ (positive closure) are J expressions;
- if $\beta_j, 1 \leq j \leq k, k \in \mathbb{N}_1$, are J expressions such that none of them contains the operator union and the operator positive closure (the reason for this is explained at page 115), then $\beta_1^{n_1} \beta_2^{n_2} \ldots \beta_k^{n_k}$ (exponentiation in this case $\beta_j$ are called exponentiation-terms) is a J expression where each $n_j \in \mathbb{N}_1$, called exponent, is either a fixed positive integer or an integer variable (representing all numbers in $\mathbb{N}_1$). We can specify that some of the exponents are equal. For example, if $k = 8$ it can be that $n_1 = n_3 = n_7 = p, n_2 = n_6 = q, p, q \in \mathbb{N}_1$ (in this case $p$ and $q$ are integer variables), $n_4 = n_8 = 5$ and $n_5 = 3$.

It is important to note that some of the $\beta_j$s can be $\varepsilon$.

The language defined by a J expression $\alpha$ is a J language and it is indicated with $L(\alpha)$. For instance, $L(\varepsilon \cup a) = \{a, \varepsilon\}$ and $L(a^p b^q a^p) = \{a^p b^q a^p | p \geq 1\}$.

If $\alpha$ is a J expression over the alphabet $\Sigma$, then the length of $\alpha$ is defined as the number of symbols of $\Sigma \cup \{\varepsilon\}$ present in $\alpha$. The length of a J expression is indicated with $|\alpha|$. The reason why we call these languages J is because this letter is the initial one in join, the building block composing the nets considered in this paper.

In writing J expressions we can omit many parentheses if we assume that positive closure and exponentiation have precedence over concatenation or union, and that concatenation has precedence over union. So, for instance, it is possible to write J expressions as $\alpha = \varepsilon \cup (ab^+ \cup b)^+ \cup a^p(bc)^q c^3 a^p b^2 (cd)^q$.

Figure 2: (a) a net of an accepting P/T system and (b) a net of a J P/T system.
Remark 4. If $\beta$ is an exponentiation with fixed positive integer exponents, we can construct another exponentiation $\beta'$ such that $L(\beta) = L(\beta)'$ and $\beta'$ has fixed positive integer constants that are all 1's.

The previous remark is clearly true: for each $\beta_k$ exponentiation-term in $\beta$ having $n_k$ as fixed positive exponent, $\beta'$ can be obtained concatenating $n_k$ times $\beta_k$. So, for instance, if $\beta = a^p(b^q)e^r a^p b^2(c + d)^q$, then $\beta' = a^p(b^q)e^r c c c e a^p b^2(c + d)^q$.

If $\Sigma$ is a set, then $|\Sigma|$ denotes the cardinality of $\Sigma$, that is the number of elements in $\Sigma$. The following follows from Definition 6.

Lemma 5. Let $\beta$ be an exponentiation-term. Then:

- if $\epsilon \in L(\beta)$, then $L(\beta) = \{\epsilon\}$;
- if $|L(\beta)| > 1$, then $\epsilon \not\in L(\beta)$.

The proof of the following lemma is rather long but not particularly difficult. The basic idea is to have a J P/T system in which input places are associated to the J expression defining the language accepted by the J P/T system, work places are associated with the possible union, concatenations, positive closure and exponentiations of the J language. The J P/T system repeatedly “consumes” (accepts) one token per time from the input places and passes one token from a work place to another. The J P/T system is non-deterministic (because it “guesses” to what part of the J expression a token can be matched).

Lemma 6. Every J language is accepted by a J P/T system.

Before presenting the next results we explain why exponentiation-terms cannot be union. There is no meaning in having (for instance) $\epsilon \in L(\beta)$, then $L(\beta) = \{\epsilon\}$; if $|L(\beta)| > 1$, then $\epsilon \not\in L(\beta)$.

Remark 4. If $\beta$ is an exponentiation with fixed positive integer exponents, we can construct another exponentiation $\beta'$ such that $L(\beta) = L(\beta)'$ and $\beta'$ has fixed positive integer constants that are all 1's.

The previous remark is clearly true: for each $\beta_k$ exponentiation-term in $\beta$ having $n_k$ as fixed positive exponent, $\beta'$ can be obtained concatenating $n_k$ times $\beta_k$. So, for instance, if $\beta = a^p(b^q)e^r a^p b^2(c + d)^q$, then $\beta' = a^p(b^q)e^r c c c e a^p b^2(c + d)^q$.

If $\Sigma$ is a set, then $|\Sigma|$ denotes the cardinality of $\Sigma$, that is the number of elements in $\Sigma$. The following follows from Definition 6.

Lemma 5. Let $\beta$ be an exponentiation-term. Then:

- if $\epsilon \in L(\beta)$, then $L(\beta) = \{\epsilon\}$;
- if $|L(\beta)| > 1$, then $\epsilon \not\in L(\beta)$.

The proof of the following lemma is rather long but not particularly difficult. The basic idea is to have a J P/T system in which input places are associated to the J expression defining the language accepted by the J P/T system, work places are associated with the possible union, concatenations, positive closure and exponentiations of the J language. The J P/T system repeatedly “consumes” (accepts) one token per time from the input places and passes one token from a work place to another. The J P/T system is non-deterministic (because it “guesses” to what part of the J expression a token can be matched).

Lemma 6. Every J language is accepted by a J P/T system.

Before presenting the next results we explain why exponentiation-terms cannot be union. There is no meaning in having (for instance) $\epsilon \in L(\beta)$, then $L(\beta) = \{\epsilon\}$; if $|L(\beta)| > 1$, then $\epsilon \not\in L(\beta)$.

Remark 4. If $\beta$ is an exponentiation with fixed positive integer exponents, we can construct another exponentiation $\beta'$ such that $L(\beta) = L(\beta)'$ and $\beta'$ has fixed positive integer constants that are all 1's.

The previous remark is clearly true: for each $\beta_k$ exponentiation-term in $\beta$ having $n_k$ as fixed positive exponent, $\beta'$ can be obtained concatenating $n_k$ times $\beta_k$. So, for instance, if $\beta = a^p(b^q)e^r a^p b^2(c + d)^q$, then $\beta' = a^p(b^q)e^r c c c e a^p b^2(c + d)^q$.

If $\Sigma$ is a set, then $|\Sigma|$ denotes the cardinality of $\Sigma$, that is the number of elements in $\Sigma$. The following follows from Definition 6.

Lemma 5. Let $\beta$ be an exponentiation-term. Then:

- if $\epsilon \in L(\beta)$, then $L(\beta) = \{\epsilon\}$;
- if $|L(\beta)| > 1$, then $\epsilon \not\in L(\beta)$.

The proof of the following lemma is rather long but not particularly difficult. The basic idea is to have a J P/T system in which input places are associated to the J expression defining the language accepted by the J P/T system, work places are associated with the possible union, concatenations, positive closure and exponentiations of the J language. The J P/T system repeatedly “consumes” (accepts) one token per time from the input places and passes one token from a work place to another. The J P/T system is non-deterministic (because it “guesses” to what part of the J expression a token can be matched).

Lemma 6. Every J language is accepted by a J P/T system.

Before presenting the next results we explain why exponentiation-terms cannot be union. There is no meaning in having (for instance) $\epsilon \in L(\beta)$, then $L(\beta) = \{\epsilon\}$; if $|L(\beta)| > 1$, then $\epsilon \not\in L(\beta)$.

Remark 4. If $\beta$ is an exponentiation with fixed positive integer exponents, we can construct another exponentiation $\beta'$ such that $L(\beta) = L(\beta)'$ and $\beta'$ has fixed positive integer constants that are all 1's.

The previous remark is clearly true: for each $\beta_k$ exponentiation-term in $\beta$ having $n_k$ as fixed positive exponent, $\beta'$ can be obtained concatenating $n_k$ times $\beta_k$. So, for instance, if $\beta = a^p(b^q)e^r a^p b^2(c + d)^q$, then $\beta' = a^p(b^q)e^r c c c e a^p b^2(c + d)^q$.

If $\Sigma$ is a set, then $|\Sigma|$ denotes the cardinality of $\Sigma$, that is the number of elements in $\Sigma$. The following follows from Definition 6.

Lemma 5. Let $\beta$ be an exponentiation-term. Then:

- if $\epsilon \in L(\beta)$, then $L(\beta) = \{\epsilon\}$;
- if $|L(\beta)| > 1$, then $\epsilon \not\in L(\beta)$.

The proof of the following lemma is rather long but not particularly difficult. The basic idea is to have a J P/T system in which input places are associated to the J expression defining the language accepted by the J P/T system, work places are associated with the possible union, concatenations, positive closure and exponentiations of the J language. The J P/T system repeatedly “consumes” (accepts) one token per time from the input places and passes one token from a work place to another. The J P/T system is non-deterministic (because it “guesses” to what part of the J expression a token can be matched).

Lemma 6. Every J language is accepted by a J P/T system.

Before presenting the next results we explain why exponentiation-terms cannot be union. There is no meaning in having (for instance) $\epsilon \in L(\beta)$, then $L(\beta) = \{\epsilon\}$; if $|L(\beta)| > 1$, then $\epsilon \not\in L(\beta)$.

Remark 4. If $\beta$ is an exponentiation with fixed positive integer exponents, we can construct another exponentiation $\beta'$ such that $L(\beta) = L(\beta)'$ and $\beta'$ has fixed positive integer constants that are all 1's.

The previous remark is clearly true: for each $\beta_k$ exponentiation-term in $\beta$ having $n_k$ as fixed positive exponent, $\beta'$ can be obtained concatenating $n_k$ times $\beta_k$. So, for instance, if $\beta = a^p(b^q)e^r a^p b^2(c + d)^q$, then $\beta' = a^p(b^q)e^r c c c e a^p b^2(c + d)^q$.

If $\Sigma$ is a set, then $|\Sigma|$ denotes the cardinality of $\Sigma$, that is the number of elements in $\Sigma$. The following follows from Definition 6.

Lemma 5. Let $\beta$ be an exponentiation-term. Then:

- if $\epsilon \in L(\beta)$, then $L(\beta) = \{\epsilon\}$;
- if $|L(\beta)| > 1$, then $\epsilon \not\in L(\beta)$.

The proof of the following lemma is rather long but not particularly difficult. The basic idea is to have a J P/T system in which input places are associated to the J expression defining the language accepted by the J P/T system, work places are associated with the possible union, concatenations, positive closure and exponentiations of the J language. The J P/T system repeatedly “consumes” (accepts) one token per time from the input places and passes one token from a work place to another. The J P/T system is non-deterministic (because it “guesses” to what part of the J expression a token can be matched).

Lemma 6. Every J language is accepted by a J P/T system.
Here the converse of the previous lemma:

**Lemma 8.** Every language accepted by a J P/T system is a J language.

**Proof:** We only provide a sketch of the proof a detailed proof would be tedious. It is very important to recall that:

- the underlying topological structure of J P/T systems is composed by *join* and that for each transition the input set is given by an input place and a work place;
- the initial configuration sees tokens in input places and in only one work place (the initial place).

Let $N$ be a J P/T system and let its input places be associated to symbols in an alphabet $\Sigma$. If $N$ contains no cycle, then $N$ accepts concatenations of symbols and unions of symbols and their concatenation. If instead $N$ contains cycles, then this means that concatenations of symbols can be repeatedly checked. This means that $N$ can accept the positive closure of symbols, concatenations and their union.

Now we prove that $N$ can accept exponentiations. Let us assume that $N$ accepts $\beta_1^+ \beta_2^+$ with $\beta_1 = \beta_{1,1} \beta_{1,2} \ldots \beta_{1,k_1}$, $\beta_2 = \beta_{2,1} \beta_{2,2} \ldots \beta_{2,k_2}$, $\beta_{1,i}, \beta_{2,j} \in \Sigma^+$, $1 \leq i \leq k_1$, $1 \leq j \leq k_2$. In order to simplify the proof we assume that $k_1 = k_2$. With slight modifications the result holds also if $k_1 \neq k_2$.

It is possible to define another J P/T system $N'$ accepting $\beta_{1,1}^{n_1} \beta_{1,2}^{n_2} \ldots \beta_{1,k_1}^{n_{k_1}} \beta_{2,1}^{n_1} \beta_{2,2}^{n_2} \ldots \beta_{2,k_1}^{n_{k_1}}$. The system $N'$ is very similar to $N$. It is made such that when the last symbol of $\beta_{1,1}$ is checked, then the first symbols of $\beta_{2,1}$ is checked. When the last symbol of $\beta_{2,1}$ is checked, then the system can either check the first symbol of $\beta_{1,1}$ or the first symbol of $\beta_{1,2}$ and so on. The same result holds if either $\beta_1$ or $\beta_2$ is not a positive closure (but just a concatenation). Informally: for J P/T systems exponentiation is a shuffling of concatenations.

Now we prove that nothing else can be accepted by J P/T systems. By contradiction, let us assume that there is a set of vectors accepted by a J P/T system having $P_{in}$ as set of initial places such that it cannot be represented by a J expression over $P_{in}$. Clearly, the set of vectors has to have an infinite number of elements. If not, then a J expression given by the union of the concatenations of the different elements in each of the finite number of vectors would represent this set.

As the number of places and transitions is finite, then the number of cycles in the J P/T system is finite, too. Depending on the number and the length of the cycles present in the J P/T system, there is a finite set of accepted initial configurations (called *border configuration*) such that for each of them there are vectors (called *added vector*) such that the (vector) sum of one border configuration to any multiple of any of its added vector leads to an accepted initial configuration. Informally, the acceptance of any border configuration needs some cycles to be traversed. Given a border configuration, its added vectors allow these cycles to be traversed other times. But then, there is a J expression that can represent the set of vectors accepted by the J P/T system. This J expression is given by the union of J expressions representing border configurations where each place is concatenated with the respective place in the added vectors to the power of an integer variable. A contradiction.

For instance, let $P_{in} = \{p_1, p_2\}$, (4,6) be a border configuration, and let (2,0) and (1,3) be added vectors for the border configuration. The J expression is then: $p_1^4(p_1p_1)^{k_1} p_2^2 \cup p_1^3 p_2^3 (p_2p_2p_2)^{k_3}$ where $k_1, k_2, k_3 \in \mathbb{N}_1$ are integer variables. □

From the previous two lemmas we have:

**Theorem 9.** A language is a J language if and only if it is accepted by a J P/T system.
4 Semilinearity of J languages

In this section, we show that the Parikh map of every J languages is semilinear. We also prove a “converse” (this is made more precise later) of this result.

Let \( N \) be the set of non-negative integers and \( n \) be a positive integer. A subset \( S \) of \( N^n \) is a linear set if there exist vectors \( v_0, v_1, \ldots, v_t \) in \( N^n \) such that
\[
S = \{ v \mid v = v_0 + i_1v_1 + \cdots + i_tv_t, \ i_j \in N \}.
\]
The vectors \( v_0 \) (referred to as the constant vector) and \( v_1, v_2, \ldots, v_t \) (referred to as the periods) are called the generators of the linear set \( S \). The set \( S \subseteq N^n \) is semilinear if it is a finite union of linear sets. The empty set is a trivial (semi)linear set, where the set of generators is empty. Every finite subset of \( N^n \) is semilinear – it is a finite union of linear sets whose generators are constant vectors. It is also clear that the semilinear sets are closed under (finite) union.

Let \( \Sigma = \{a_1, a_2, \ldots, a_n\} \) be an alphabet. For each word \( w \) in \( \Sigma^* \), define the Parikh map of \( w \) to be
\[
\psi(w) = (|w|_{a_1}, |w|_{a_2}, \ldots, |w|_{a_n}).
\]
where \( |w|_{a_i} \) denotes the number of occurrences of symbol \( a_i \) in \( w \). For a language \( L \subseteq \Sigma^* \), the Parikh map of \( L \) is \( \psi(L) = \{ \psi(w) \mid w \in L \} \). The language \( L \) is semilinear if \( \psi(L) \) is a semilinear set.

There is a simple automata characterisation of semilinear sets. Let \( M \) be a non-deterministic finite automaton without an input tape, but with \( n \) counters (for some \( n \geq 1 \)). The computation of \( M \) starts with all the counters zero and the automaton in the start state. An atomic move of \( M \) consists of incrementing at most one counter by 1 and changing the state (decrements are not allowed). An \( n \)-tuple \( v = (i_1, \ldots, i_n) \in N^n \) is generated by \( M \) if \( M \), when started from its initial configuration, halts with \( v \) as the contents of the counters. The set of all \( n \)-tuples generated by \( M \) is denoted by \( G(M) \). We call this automaton a finite-state generator.

The following result was shown in [5]:

**Theorem 10.** Let \( n \geq 1 \). A subset \( S \subseteq N^n \) is semilinear if and only if it can be generated by a finite-state generator with \( n \) counters.

Using Theorem 10, we can then prove the following result.

**Theorem 11.** The Parikh map of every language denoted by a J expression is semilinear.

For the “converse” of Theorem 11, we need the following definition.

**Definition 12.** Let \( S \subseteq N^n \) and \( \Sigma = \{a_1, \ldots, a_n\} \). Define the language
\[
L_S = \{ a_1^{s_1}a_2^{s_2} \cdots a_n^{s_n} \mid (s_1, \ldots, s_n) \in S \}.
\]

**Theorem 13.** If \( S \) is a semilinear set, then \( L_S \) is a J language.

5 Complexity of J Languages

Here, we briefly discuss the (TM) space complexity of J languages. We will show that every J language can be accepted by a non-deterministic Turing machine (NTM) with a one-way read-only input and a \( \log n \) space-bounded read-write work-tape. Actually, what we show is that the language can be accepted
by a one-way non-deterministic finite automaton augmented with a finite number of counters. In each computing step each counter can be incremented/decremented by 1 and tested for zero. The counters start with zero value, and we assume (without loss of generality) that the machine accepts when in the final state and when all counters store zero. During the computation, the (non-negative) integer value in each counter never exceeds the length of the one-way read-only input. We call this machine a linear-space multicounter machine, or simply, LCM. Clearly, an LCM can be simulated by a one-way \( \log n \) space-bounded NTM, since the values in the counters can be stored and managed on a \( \log n \) read-write work-tape.

The next two results can be shown.

**Theorem 14.** Every J language can be accepted by an LCM.

**Corollary 15.** Every J language can be accepted by a one-way \( \log n \) space-bounded NTM.

It is well-known and, actually easily shown, that \( L = \{ x\#x^R \mid x \in \{0,1\}^+ \} \) (\( R \) denotes reverse) cannot be accepted by a one-way \( \log n \) space-bounded NTM, hence, cannot be accepted by an LCM. (For an input \( x\#x^R \) of length \( 2n + 1 \), a one-way NTM with \( \log n \) space can only differentiate a linear number of strings of \( x \)'s before the symbol \#. But there are \( 2^n \) different \( x \)'s.)

**Corollary 16.** There are context-free languages that are not J languages.

### 6 A grammatical characterisation of J languages

In this section, we provide a grammatical characterisation of J languages. The grammar is an extension of the right-linear simple matrix grammar studied in [7].

Let \( \Sigma \) be the set of terminal symbols. The non-terminal symbols are partitioned into two disjoint sets, \( Q \) and \( R \). There is a unique start non-terminal \( S_0 \in Q \) from which all derivations start from. The rules are of two types:

**Basic Rules:**

1. \( S \to w \), where \( w \in \Sigma \cup \{ \varepsilon \} \) and \( S \in Q \) does not appear on the RHS of any basic rule, but can appear in a matrix rule 6 below.
2. \( S \to S_1S_2 \), where \( S, S_1, S_2 \) are distinct non-terminals in \( Q \), and \( S \) does not appear on the RHS of any basic rule, but can appear in a matrix rule 6 below.
3. \( S \to S_1S_2 \), where \( S, S_1, S_k \) are distinct non-terminals in \( Q \), and \( S \) does does not appear on the RHS of any basic rule, but can appear in a matrix rule 6 below.
4. \( S \to SS \), where \( S \in Q \) does not appear on the RHS of any basic rule (except in this rule), but can appear in a matrix rule 6 below.
5. \( S \to (A_{11}A_{12}\cdots A_{1m}, \ldots, A_{k1}A_{k2}\cdots A_{km}) \), where \( m \geq 1, k \geq 1 \), each \( A_{ij} \) is a non-terminal in \( R \) and \( S \in Q \) can appear on the RHS of basic rules 2, 3, 4, but cannot appear in a matrix rule 6 below.

**Right-Linear Simple Matrix Rules:**

6. \( [A_1 \to S_1A_1, \ldots, A_k \to S_kA_k] \), where \( k \geq 1 \), each \( A_i \) a non-terminal in \( R \), and each \( S_i \in Q \) (subject to the restriction in rule 5 above).

Restriction 1: We require that if \( [A_1 \to S_1A_1, \ldots, A_k \to S_kA_k] \) and \( [A_1 \to S_1'A_1, \ldots, A_k \to S_k'A_k] \) are both matrix rules, then \( S_i = S_i' \) for \( 1 \leq i \leq k \). Thus, the RHS is unique for the given \( A_i \)'s on the LHS.
7. [\(A_1 \rightarrow w_1, \ldots, A_k \rightarrow w_k\)], where \(k \geq 1\), each \(A_i\) a non-terminal in \(\mathcal{R}\), each \(w_i\) in \(\Sigma^*\).

The derivation of a string \(w \in \Sigma^*\) in the language starts from the non-terminal \(S_0\). If at some point during the derivation, an intermediate string is reached that contains a non-terminal \(S\) for which a rule of form 5 is applied, this \(S\) will be replaced by an \(n\)-tuple \((A_{11}A_{12} \cdots A_{1m}, \ldots, A_{k1}A_{k2} \cdots A_{km})\). Next, a rule of form 6 is applied in parallel, i.e., application of the rule rewrites the leftmost non-terminal of each of the \(k\)-coordinates. Application of rule 6 is done \(r \geq 0\) times, where \(r\) is chosen non-deterministically; after which rule 7 is applied. The process is repeated for the next leftmost non-terminal of each coordinate. At the end, when all \(k\) coordinates are non-null strings in \(Q^+\), we “merge” the \(k\) components into a single string. Then the derivation continues until \(w\) is reached.

We can prove the following result.

**Theorem 17.** The languages generated by ERLSMGs are exactly the \(J\) languages, which allow union and positive closure in exponentiation.

**Corollary 18.** The languages generated by ERLSMG’s in which the \(S\)’s on the left-hand-side of rules of forms 2 and 4 do not appear on the right-hand-sides of rules of form 6 are exactly the \(J\) languages.

### 7 Final remarks

In Section 2 we said that the way to accept languages (sets of vectors) considered by us differs from the standard one used in Petri nets (concatenations of the labels of firing sequences) [4,8]. The reason why we did not consider this standard way in the present paper is because we wanted here to focus only on the topology. (We are in the process of writing a paper discussing the relations between these two different ways of accepting languages).

In [3,2] it is shown how the results obtained from the computational power of P/T system whose underlying net is composed of \(\text{joins}\) and \(\text{fork}\) can facilitate the study of the computational power of models of \(\text{membrane systems}\) (also known as \(\text{P systems}\)) [9] based on multiset rewriting. These results use a definition of \(\text{equivalence}\) (also present in [3,2]). This is the “new way to analyse the computational power of a formal system” we mentioned in Section 1.

In a nutshell, the idea is the following: if a formal system \(S\) can simulate \(\text{fork}, \text{join}\) and their composition, then the results on the computational power of P/T systems whose underlying net is composed of \(\text{joins}\) and \(\text{fork}\) are also valid to \(S\).

In [3,2] it is shown that P systems with catalysts can simulate a \(\text{fork}\) using rules of the kind \(a \rightarrow b_1b_2\), while the simulation of a \(\text{join}\) does not require the use of such rules. So, knowing from [3,2] how P systems with catalysts can simulate \(\text{join}\) and Theorem 9, we can say that the family of languages generated by P systems with catalysts not using rules of the kind \(a \rightarrow b_1b_2\) is \(J\).

Using the definitions and results of P systems with catalysts in [3,2] we can be more precise and state:

**Corollary 19.**

- The family of languages accepted by P systems with catalysts of degree 2 and 2 catalysts not using rules of the kind \(a \rightarrow b_1b_2\) is \(J\);
- the family of languages accepted by purely catalytic P systems of degree 2 and 3 catalysts not using rules of the kind \(a \rightarrow b_1b_2\) is \(J\).
We end this paper with an open problem.
In the rule of form 6, we had a restriction that if

\[ A_1 \rightarrow S_1 A_1, \ldots, A_k \rightarrow S_k A_k \]  

and

\[ A_1 \rightarrow S'_1 A_1, \ldots, A_k \rightarrow S'_k A_k \]

are both matrix rules, then \( S_i = S'_i \) for \( 1 \leq i \leq k \). Suppose we remove this restriction. Is there an extension of the J P/T systems that can characterise these grammars?

References

[1] P. Frisco (2006): *P systems, Petri nets, and Program machines*. In: R. Freund, G. Lojka, M. Oswald & G. Păun, editors: *Membrane Computing. 6th International Workshop, WMC 2005, Vienna, Austria, July 18–21, 2005, Revised Selected and Invited Papers*, LNCS 3850. Springer-Verlag, Berlin, Heidelberg, New York, pp. 209–223.

[2] P. Frisco (2008): *A hierarchy of computational processes*. Technical Report HW-MACS-TR-0059, Heriot-Watt University. [http://www.macs.hw.ac.uk:8080/techreps/index.html](http://www.macs.hw.ac.uk:8080/techreps/index.html).

[3] P. Frisco (2009): *Computing with Cells. Advances in Membrane Computing*. Oxford University Press. To appear.

[4] M. Hack (1976): *Petri Net Language*. MIT-Cambridge, MA.

[5] T. Harju, O. H. Ibarra, J. Karhumaki & A. Salomaa (2002): *Some decision problems concerning semilinearity and commutation*. Journal of Computer and System Science 65, pp. 278–294.

[6] J. E. Hopcroft & D. Ullman (1979): *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley.

[7] O. H. Ibarra (1970): *Simple matrix languages*. Information and Control 17, pp. 359–394.

[8] M. Jantzen (1987): *Language theory of Petri nets*. In: *Advances in Petri nets 1986, part I on Petri nets: central models and their properties*. Springer-Verlag, Berlin, Heidelberg, New York, pp. 397–412.

[9] G. Păun (2000): *Computing with membranes*. Journal of Computer and System Science 1, pp. 108–143.

[10] W. Reisig (1985): *Petri Nets: An Introduction*, Monographs in Theoretical Computer Science 4. Springer-Verlag, Berlin, Heidelberg, New York.

[11] W. Reisig & G. Rozenberg, editors (1998): *Lectures on Petri Nets I: Basic Models*, LNCS 1491. Springer-Verlag, Berlin, Heidelberg, New York.