On languages accepted by P/T systems composed by *join*

Pierluigi Frisco
School of Math. and Comp. 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

Abstract

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.

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

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

i) (P, T, F) *is a* net*:*

1. P *and* T *are sets with* $P \cap T = \varnothing$*;*
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) $W : F \to \mathbb{N}_1$ *is a* weight function*;*

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

iv) $C_{in} : P \to \mathbb{N}$ *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 $N(C_{in}) = (P, T, F, W, P_{in}, p_{init}, p_{fin})$ 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 $N(C_{in}) = (P, T, F, W, P_{in}, p_{init}, p_{fin})$ 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:

$C_{fin}(p_{fin}) > 0$;

there is at least one path from C_{in} to C_{fin};

no other configuration D in the paths from C_{in} to C_{fin} is such that $D(p_{fin}) > 0$.

The *set of vectors accepted* by N is denoted by $\mathsf{N}^k(N)$ and it is composed by the vectors $(C_{in}(p_{in,1}), \ldots, C_{in}(p_{in,k}))$ 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 8.

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

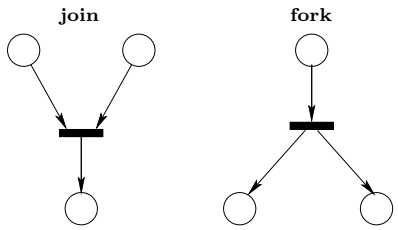

Figure 1: Building blocks: *join* and *fork*.

Also from [2] we take:

Definition 2 *Let $x, y \in \{join, fork\}$ be building blocks and let $\bar{t}_x$ and $\hat{t}_y$ be the transitions present in x and y respectively.*

We say that y comes after x *(or* x is followed by y*, or* x comes before y *or* x and y are in sequence*) if $\bar{t}_x^{\bullet} \cap {}^{\bullet}\hat{t}_y \neq \varnothing$ and ${}^{\bullet}\bar{t}_x \cap {}^{\bullet}\hat{t}_y = \varnothing$. We say that* x and y are in parallel *if ${}^{\bullet}\bar{t}_x \cap {}^{\bullet}\hat{t}_y \neq \varnothing$ and $\bar{t}_x^{\bullet} \cap {}^{\bullet}\hat{t}_y = \varnothing$.*

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 $N = (P, T, F, W, P_{in}, p_{init}, p_{fin})$ is such a P/T systems, then for each $t \in T$, ${}^\bullet t \in P_{in} \times P \setminus P_{in}$ and $t^\bullet \in P \setminus P_{in}$. 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 2.a is part of the net underlying an accepting P/T system N with P as set of places, $P_{in} \subset P$ as set input places and T as set of transitions. The net depicted in Figure 2.b belongs to a J P/T system N_J with $P \cup \{w'_1, w'_2\}$ as set of places, P_{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.

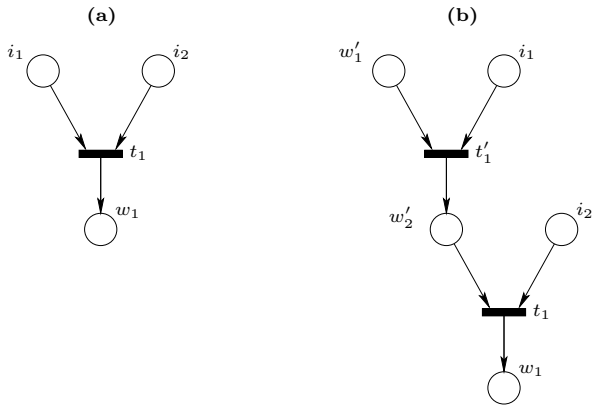

Figure 2: **(a)** a net of an accepting P/T system and **(b)** a net of a J P/T system.

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 Σ be an alphabet, then:*

ε (the empty string*) is a J expression;*

for each $v \in \Sigma$, v *is a J expression;*

if α *and* β *are J expressions, then* $(\alpha \cup \beta)$ *is a J expression (*union*, in this case* α *and* β *are called union-terms);*

if α *and* β *are J expressions such that* $\alpha, \beta \neq \varepsilon$ *but they can contain* ε *(e.g.,* $\alpha = a \cup \varepsilon$*), then* $(\alpha\beta)$ *(*concatenation*), and* (α^+) *(*positive closure*) are J expressions;*

if β_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 10), then* $\beta_1^{n_1} \dots \beta_k^{n_k}$ *(*exponentiation *in this case* β_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* $n_1 = n_3 = n_7 = p$, $n_2 = n_6 = q$, $p, q \in \mathbb{N}_1$ *(*p *and* q *are integer variables),* $n_4 = n_8 = 5$ *and* $n_5 = 3$ *(*n_4, n_5 *and* n_8 *are fixed positive integers). In this case we would have* $\beta = \beta_1^p \beta_2^q \beta_3^p \beta_4^5 \beta_5^3 \beta_6^q \beta_7^p \beta_8^5$*. It is important to note that some of the* β_j*s can be* ε*.*

The language defined by a J expression α *is a* J language *and it is indicated with* $\mathsf{L}(\alpha)$*. For instance,* $\mathsf{L}(a \cup \varepsilon) = \{a, \varepsilon\}$ *and* $\mathsf{L}(a^p b^3 a^p) = \{a^p b^3 a^p | p \geq 1\}$.

If α *is a J expression over the alphabet* Σ*, then the* length *of* α *is defined as the number of symbols of* $\Sigma \cup \{\varepsilon\}$ *present in* α*. 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 is 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 $\alpha = \varepsilon \cup (ab^+ \cup b)^+ \cup a^p(bc)^q c^3 a^p b^2 (cd)^q$.

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

The previous remark is clearly true: for each β_k exponentiation-term in β having n_k as fixed positive exponent, β' can be obtained concatenating n_k times β_k. So, for instance, if $\beta = a^p(bc)^q c^3 a^p b^2 (cd)^q$, then $\beta' = a^p(bc)^q ccca^p bb(cd)^q$.

If Σ is a set, then $|\Sigma|$ denotes the *cardinality* of Σ, that is the number of elements in Σ. The following follows from Definition 3:

Lemma 1 *Let* β *be an exponentiation-term. Then:*

if $\varepsilon \in \mathsf{L}(\beta)$*, then* $\mathsf{L}(\beta) = \{\varepsilon\}$*;*

if $|\mathsf{L}(\beta)| > 1$*, then* $\varepsilon \notin \mathsf{L}(\beta)$.

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).

We prefer to give a unique long proof (instead of splitting it in smaller proofs) because the several parts of the proof (concatenation, related to union, positive closure and exponentiation) are interconnected.

Given a J expression α, with $\|\alpha\|_+$ we indicate the number of positive closures present in α while with $\|\alpha\|_{exp}$ we indicate the number of maximal exponentiations present in α. The elements of $\Sigma \cup \{\varepsilon\}$ present in a J expression α over Σ are considered numbered from left to right and are addressed with $\alpha_i, 1 \leq i \leq \|\alpha\|$. If an element in $\Sigma \cup \{\varepsilon\}$ has number k, then we say that that element has *position* k in α. If we consider the J expression $\alpha = \varepsilon \cup (ab^+ \cup b)^+ \cup a^p(bc)^q c^3 a^p b^2 (cd)^q$, then $\alpha_1 = \varepsilon$, $\alpha_2 = a$, $\alpha_3 = b$ and so on. The positive closures and the exponentiations present in a J expression are considered numbered from left to right and are addressed with $+_i$, $1 \leq i \leq \|\alpha\|_+$ and exp_j, $1 \leq j \leq \|\alpha\|_{exp}$, respectively. In the previous example $+_1 = b, +_2 = (ab^+ \cup b)$, and $exp_1 = a^p(bc)^q c^3 a^p b^2 (cd)^q$ and $\mathsf{L}(\alpha) = \{\varepsilon, ab, abb, \ldots, abab, abbabb, b, bb, \ldots, abcc^3 ab^2 cd, aabcc^3 aab^2 cd, abcbcc^3 sb^2 cdcd, \ldots\}$.

Lemma 2 *Every J language is accepted by a J P/T system.*

Proof. Given a J language $\mathsf{L}(\alpha)$ defined by the J expression α over an alphabet Σ we define a non-deterministic P/T system that can accept a vector $(C_{in}(p_{in,1}), \ldots, C_{in}(p_{in,k}))$ if and only if such vector encodes a word defined by α. Let w be a word over Σ, if $w \notin \mathsf{L}(\alpha)$, then the P/T system halts with no token in its final place; if $w \in \mathsf{L}(\alpha)$, then the P/T system can halt with a token in its final place.

Let $\Sigma = \{v_1, \ldots, v_n\}$ be an alphabet and let α be a J expression over Σ defining the J language $\mathsf{L}(\alpha)$.

The P/T system has input places named $p_{\varepsilon,j}$, $p_{v_i,j}$, $1 \leq i \leq n$, $1 \leq j \leq \tau+1$. It has work places named $s_1, \ldots, s_\tau, s_{\tau+1}$ (where s_1 is the initial place and $s_{\tau+1}$ the final place), s_z, $s_1^+, \ldots, s_{\|\alpha\|_+}^+$.

The transitions are introduced in the following:

concatenation: t_l, $1 \leq l \leq \tau$, are transitions with ${}^\bullet t_l = \{s_l, p_{\alpha_l,l}\}$ and $t_l^\bullet = \{s_{l+1}\}$ (the following items *ii* and *v* indicate exceptions to these transitions).

Informally: these transitions allow the P/T system to check if all the symbols present in α are also present in the input word. The exceptions to these check are when unions or exponentiations are present in α;

union: for every $\beta_1, \ldots, \beta_r$ J expressions such that $\beta_1 \cup \ldots \cup \beta_r$ is present in α, $\beta_1 = \beta_{1,1} \ldots \beta_{1,\|\beta_1\|} = \alpha_{q_{1,1}} \ldots \alpha_{q_{1,\|\beta_1\|}}; \ldots; \beta_r = \beta_{r,1} \ldots \beta_{r,\|\beta_r\|} = \alpha_{q_{r,1}} \ldots \alpha_{q_{r,\|\beta_r\|}}$, there are transitions:

i) $t_j^{(1)},\ 2 \leq j \leq r$, with ${}^\bullet t_j^{(1)} = \{s_{q_{1,1}}, p_{\alpha_{q_{j,1}}, q_{j,1}}\}$ and:

if $|\beta_j| = 1$ and $\beta_j = \beta_j^+,\ k^{th}$ positive closure of α $(1 \leq k \leq \|\alpha\|_+)$, then ${t_j^{(1)}}^\bullet = \{s_{q_{j,1}}\}$.
Informally: if a union-term is a positive closure of only one symbol, then it is possible to check it again;

otherwise, if $|\beta_j| = 1$ and $\beta_j = \beta_j^n$ exponentiation-term of the k^{th} exponentiation of α $(1 \leq k \leq \|\alpha\|_{exp})$ to the power of n, then ${t_j^{(1)}}^\bullet = \{s_g\}$, g being the position of the first symbol of the next exponentiation-term to the power n in the same exponentiation.
Informally: if a union term is the first exponentiation-term having only one symbol, then it is possible to check the first symbol of the next exponentiation-term to the power n of the same exponentiation;

otherwise, if $|\beta_j| = 1$, then ${t_j^{(1)}}^\bullet = \{s_{q_{r,\|\beta_r\|+1}}\}$.
Informally: if the union-term is only one symbol, then it is possible to check what after the union;

otherwise ${t_j^{(1)}}^\bullet = \{s_{q_{j,2}}\}$.
Informally: if the union-term is more than one symbol, then it is possible to check the second symbol of the same union-term;

Informally: all these transitions allow the P/T system to jump to check the remaining symbols of a union-term when the first symbol of this union-term is present;

ii) $t_j^{(2)}, 1 \leq j \leq r$, with ${}^\bullet t_j^{(2)} = \{s_{q_{j,\|\beta_j\|}}, p_{\alpha_{q_{j,\|\beta_j\|}}, q_{j,\|\beta_j\|}}\}$, ${t_j^{(2)}}^\bullet = \{s_{q_{r,\|\beta_r\|+1}}\}$ and $t_{q_{j,\|\beta_j\|}}$ from *concatenation* is not present.

Informally: when the last symbol of an element of a union-term has been checked, then the P/T system goes to check the symbol present after the last symbol of the last union-term (and not the first symbol of the next union-term). If, for instance, $\alpha = (ab \cup cd)e$ and the b was there, then the P/T system can check the e (and not the c);

positive closure: for every J expression β such that β^+ is present in α, β^+ being the $j^{th}, 1 \leq j \leq \|\alpha\|_+$, positive closure present in α and $\beta = \beta_1 \ldots \beta_r = \alpha_q \ldots \alpha_{q+r}$, there are transitions:

iii) $t_{q+r}^{(3)}$ with ${}^\bullet t_{q+r}^{(3)} = \{s_{q+r}, p_{\alpha_{q+r}, q+r}\}$ and ${t_{q+r}^{(3)}}^\bullet = \{s_j^+\}$.
Informally: when the last symbol of β is checked the P/T system can put a token into s_j^+ the work place associated with that specific positive closure. The transitions given under *concatenation* allow the P/T system to check the symbol coming after β_r;

iv) t_j^+ with ${}^\bullet t_j^+ = \{s_j^+, p_{\alpha_q,q}\}$ and ${t_j^+}^\bullet = \{s_{q+1}\}$ if $r > 1$ or ${t_j^+}^\bullet = \{s_j^+\}$ if $r = 1$.

Informally: when a token is present in s_j^+ the P/T system checks once more the presence of the elements of the positive closure.

exponentiation: we assume exponentiations to have 1 as the only fixed positive integer (see Remark 1). During this explanation we will consider the exponentiation $\beta = (ab)^p b^q ccca^p dd(cb)^q$ where the exponentiation-terms are $\beta_1 = ab, \beta_2 = b, \beta_3 = c, \beta_4 = c$ and so on; the integer variables are $n_1 = p$ and $n_2 = q$. Moreover, for exponentiation-terms with more than one symbol we refer to their symbols. For instance, in the given exponentiation we have $\beta_{1,1} = a, \beta_{1,2} = b, \beta_{9,1} = c$ and $\beta_{9,2} = b$. We recall that, according to Definition 3 the exponentiation-terms do not contain the operators union and positive closure. There are transitions:

v) $t_{q,k}^{(4)}$, $1 \leq q \leq \|\alpha\|_{exp}$, $1 \leq k \leq \|\alpha\|$, where k is the position in α of the last symbol v of an exponentiation-term β_t to a certain power n_t (an integer variable) in β. ${}^\bullet t_{q,k}^{(4)} = \{s, p\}$ where $s = s_k$ and $p = p_{v,k}$ and ${t_{q,k}^{(4)}}^\bullet = \{s'\}$ where $s' = s_{k'}$ and k' is the position of the first symbol of $\beta_{t'}$ the next exponentialtion-term to the power of n_t in β.

Informally: these transitions allow the P/T system to check the presence of the next exponentiation-term having a certain integer variable as exponent after checking the presence of an exponentiation-term having the same integer variable as exponent. In the current example the presence of $\beta_{1,2} = b$ let the P/T system check the presence of $\beta_6 = a$ and the presence of $\beta_2 = b$ let the P/T system check the presence of $\beta_{9,1} = c$.

vi) $t_{q,k}^{(5)}$, $1 \leq q \leq \|\alpha\|_{exp}$, $1 \leq k \leq \|\alpha\|$, where k is the position of the last symbol v of an exponentiation-term β_t to a certain power $n_t \neq 1$ (an integer variable) in β. ${}^\bullet t_{q,k}^{(5)} = \{s, p\}$ where $s = s_k$ and $p = p_{v,k}$ and ${t_{q,k}^{(5)}}^\bullet = \{s'\}$ where $s = s_{k'}$ and k' is the position of the first symbol of $\beta_{t'}$, the first exponentiation-term to the power $n_{t'}$. It can be that $n_{t'} = n_t$.

Informally: when the last symbol of an exponentiation-term to the power n_t has been checked, then the P/T system can check the presence of the first symbol of the first exponentiation-term to the power $n_{t'}$. In the current example this means that when $\beta_6 = a$ or $\beta_{9,2} = b$ have been checked, then the P/T system can check the presence of either $\beta_{1,1} = 1$, $\beta_2 = b$ or $\beta_3 = c$.

vii) $t_{q,k}^{(6)}$, $1 \leq q \leq \|\alpha\|_{exp}$, $1 \leq k \leq \|\alpha\|$, where k is the position of the last symbol v of an exponentiation-term β_t to the power 1 in β. ${}^\bullet t_{q,k}^{(6)} = \{s, p\}$ where $s = s_k$ and $p = p_{v,k}$ and ${t_{q,k}^{(6)}}^\bullet = \{s'\}$ where $s' = s_{k'}$ and k' is the position of the first symbol after the exponentiation β.

Informally: when the last symbol of the last exponentiation-term to the power of 1 has been checked, then the P/T system can check the presence of the first symbol after the exponentiation. IN the current example this means that when $\beta_8 = d$ has been checked, then the P/T system can check what comes after $\beta_{9,2}$.

extra: $t_{v_i,j}$ with ${}^\bullet t_{v_i,j} = \{s_{\tau+1}, p_{v_i,j}\}, t^\bullet_{v_i,j} = \{s_z\}$ for each $1 \leq i \leq n, 1 \leq j \leq \tau + 1$.

Informally: if when $s_{\tau+1}$ (final place) has a token any of the input places contain at least one token, then the P/T system removes the token from the final place.

All these transitions belong to *join*. An example of the construction of such a P/T system is given in Section 4.1.

The P/T system is such that the firing of every transition removes one token from an input place and one token from a work place and put one token in a work place. This means that the number of tokens in the work places is constant (invariant) and equal to 1 for all the computation of the P/T system.

The initial configuration of the P/T system has always one token in s_1. Additionally, several tokens can be present in the input places.

The P/T system is such that if $C_{in}(p_{w_1,j_1}) = k_1 \dots C_{in}(p_{w_t,j_t}) = k_t,\ w_i \in \Sigma \cup \{\varepsilon\},\ 1 \leq j \leq \tau,\ 1 \leq i \leq t,\ t \in \mathbb{N}_1$, then the P/T system halts with a token in $s_{\tau+1}$ if and only if $w_1^{k_1} \dots w_t^{k_t}$ belongs to $\mathsf{L}(\alpha)$.

The P/T system works in the following way: most of the firing of transitions check if the tokens present in the input places can be matched with the symbols in the J expression α defining the language $\mathsf{L}(\alpha)$. If the firing occurs, then such matching is present, if not the P/T system will never have a token in its final place $s_{\tau+1}$.

The possible checking firing sequences of the P/T system are:

concatenation: for each $\beta = \beta_1 \dots \beta_{\|\beta\|},\ \beta_1, \dots, \beta_{\|\beta\|} \in \Sigma \cup \{\varepsilon\}$, J expression present in α, the firing sequence checking if $\beta_1, \dots, \beta_{\|\beta\|}$ are consecutive symbols in the input word is possible;

union: for each $\beta_1 \cup \dots \cup \beta_r$ J expression present in $\alpha,\ \beta_1, \dots, \beta_r$, being J expressions, the firing sequences checking if any of the $\beta_i, 1 \leq i \leq r$, is present in the input word is possible;

positive closure: for each β^+ J expression present in α, the firing sequence checking if any consecutive repetition of β is present in the input word is possible;

exponentiation: for each $\beta = \beta_1^{n_1} \dots \beta_k^{n_k}$ J expression present in α such that $\beta_j,\ 1 \leq j \leq k,\ k \in \mathbb{N}_1$, are J expressions not containing the operators union and positive closure (as requested by the definition of exponentiation in Definition 3), the firing checking that all the β_j exponentiation-terms are present n_j times is possible.

It can happen that after any of these checking firing sequences the P/T system reaches a configuration having a token in $s_{\tau+1}$. When this happens another (last) firing can take place only if any of the input places has a token.

If this happens, then the token from $s_{\tau+1}$ is removed.

It is important to notice that in any configuration of a J P/T system only one (any one) work place contains just one token.

In order to complete this proof we have to prove that if the P/T system has an initial configuration such that $C_{in}(p_{w_1,j_1}) = k_1, \ldots, C_{in}(p_{w_t,j_t}) = k_t,\ w_i \in \Sigma \cup \{\varepsilon\},\ 1 \leq j \leq \tau,\ 1 \leq i \leq t,\ t \in \mathbb{N}_1$ and $w_1^{k_1} \ldots w_t^{k_t}$ does not belong to $\mathsf{L}(\alpha)$, then the P/T system will halt with no token in $s_{\tau+1}$.

We start noticing that if in the initial configuration of the P/T system there are tokens in $p_{v,j}$ and $p_{v',j},\ v, v' \in \Sigma \cup \{\varepsilon\},\ v \neq v',\ 1 \leq j \leq \tau + 1$, then the last configuration of the P/T system sees no token in $s_{\tau+1}$. This is because, considering how the P/T system has been defined, there are not two transitions t and t' different than *extra* such that $\{p_{v,j}\} \in {}^\bullet t$ and $\{p_{v',j}\} \in {}^\bullet t'$. So, the P/T system can halt in a configuration with a token in $s_{\tau+1}$ only if in its initial configuration there are no two places $p_{v,j}$ and $p_{v',j}$ with at least one token each.

By contradiction, let us assume that the P/T system has an initial configuration such that $C_{in}(p_{w_1,j_1}) = k_1, \ldots, C_{in}(p_{w_t,j_t}) = k_t,\ w_i \in \Sigma \cup \{\varepsilon\},\ 1 \leq j \leq \tau,\ 1 \leq i \leq t,\ t \in \mathbb{N}_1$, that $w_1^{k_1} \ldots w_t^{k_t}$ does not belong to $\mathsf{L}(\alpha)$ and that the P/T system halts with a token in $s_{\tau+1}$. This means that from the initial configuration there is a firing sequence (following the concatenations, unions, positive closures and exponentiations present in α) having as last configuration one in which $s_{\tau+1}$ has a token. But this implies that $w_1^{k_1} \ldots w_t^{k_t}$ belongs to α. A contradiction. □

Before presenting the next results we explain why exponentiation-terms have to be different than union and positive closure. Let $\beta = \beta_1^{n_1}\beta_2^{n_2}\beta_3^{n_1}\beta_4^{n_2}$ be an exponentiation with $\beta_1^{n_1}, \beta_2^{n_2}, \beta_3^{n_1}, \beta_4^{n_2}$ exponentiation terms. There is no meaning in having (for instance) $\beta_1 = \alpha^+$, where α is a J expression, as $\beta_1^{n_1} = \alpha^{+^{n_1}} = \alpha^+$. So, $\beta = \alpha^+\beta_2^{n_2}\beta_3^{n_1}\beta_4^{n_2}$ is the concatenation of α^+ to an exponentiation. A similar argument holds if an exponentiation term contains a positive closure, that is, for instance, $\beta = \alpha\gamma^+$ where α and γ are J expressions.

The reason why exponentiation-terms cannot be union depends on the fact that J P/T systems do not have memory. Let β be defined as in the above, let $n_2 > 1$ and let (for example) $\beta_2 = \alpha_1 \cup \alpha_2$, where α_1 and α_2 are J expressions. This means that $\beta = \beta_1^{n_1}(\alpha_1 \cup \alpha_2)^{n_2}\beta_3^{n_1}\beta_4^{n_2}$. Let us assume that in the initial configuration of the J P/T system accepting β there are some tokens in the input places associated to α_1 and to α_2. We know from Lemma 2 that the check of the presence of symbols in β_2 and β_4 is done in passages: first checking the occurrence of symbols in β_2, then the one in β_4, then (second passage) the one in β_2 again, and so on. It can be that (as the J P/T system does not have memory) in the first passage tokens related to α_1 are checked, while in the second passage tokens related to α_2 are checked. This would not be a desired

behaviour.

The fact that exponentiation-terms cannot be union is not a big limit as we can rewrite β as $\beta_1^{n_1}\alpha_1^{n_2}\beta_3^{n_1}\beta_4^{n_2} \cup \beta_1^{n_1}\alpha_2^{n_2}\beta_3^{n_1}\beta_4^{n_2}$.

Here a concept that we need in the following:

Definition 4 *Let N be a J P/T system. We say that N contains cycles if and only if some firing sequences of N are of the kind $\alpha\beta^n\gamma \in T^*$, where T is the set of transitions of N and $n > 1$. A* cycle *is a cyclic path in the net underlying N having β as sequential transitions in a firing sequence.*

We denote cycles with the sequence of pairs of places and transitions belonging to it. The length of a cycle *is the number of transitions present into it.*

In Figure 3 a cycle of length 2 is $(s_1^+, p_{a,2})t_1^+(s_3, p_{a,3})t_3^{(3)}$.

Here the converse of the previous lemma:

Lemma 3 *Every language accepted by a J P/T system is a J language.*

Proof. We only provide a sketch of the proof. 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 Σ. 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 have to 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 β_1 or β_2 is not a positive closure (but just a concatenation). Informally: for J P/T systems exponentiation is a shuffling of concatenations. □

From the previous two lemmas we have:

Theorem 1 *A language is a J language if and only if it is accepted by a J P/T system.*

4 Examples related to Lemma 2

In this section we describe the J P/T system having a finite number of places and transitions and whose underlying net is composed only by *join* accepting the J language defined by the J expression $\alpha = \varepsilon \cup (aa)^+ \cup a^{n_1}b^{n_2}a^{n_1}\varepsilon^{n_2}$ for $n_1, n_2 \in \mathbb{N}_1$. This is a rather simple J expression, anyhow we will see that the P/T system accepting $\mathsf{L}(\alpha)$ (partially depicted in Figure 3) is complex.

4.1 Creation of the P/T system

We consider the alphabet $\Sigma = \{a, b\}$ and the J expression $\alpha = \varepsilon \cup (aa)^+ \cup a^n bba^n$, so $\|\alpha\| = 7 = \tau,\ \|\alpha\|_+ = 1,\ \|\alpha\|_{exp} = 1$. The input places of the P/T system are: $p_{\varepsilon,j},\ p_{a,j}$ and $p_{b,j},\ 1 \leq j \leq 8 = \tau + 1$; the work places are: $s_1, \ldots, s_8$ (where s_1 is the initial place and s_8 is the final place), s_z, s_1^+ and s_1^{exp}.

The transitions are introduced in the following:

concatenation: t_1 is not present, $t_1^{(2)}$ is present instead;

t_2 is present with ${}^\bullet t_2 = \{s_2, p_{a,2}\}$ and $t_2^\bullet = \{s_3\}$;

t_3 is not present, $t_2^{(2)}$ is present instead;

t_4 is not present, $t_{1,1}^{(4)}$ is present instead;

t_5 is not present, $t_{1,1,2}^{(4)}$ is present instead;

t_6 is not present, $t_{1,1}^{(6)}$ is present instead;

t_7 is present with ${}^\bullet t_7 = \{s_7, p_{\varepsilon,7}\},\ t_7^\bullet = \{s_8\}$;

union: $t_2^{(1)}$ is present with ${}^\bullet t_2^{(1)} = \{s_1, p_{a,2}\}$ and ${t_2^{(1)}}^\bullet = \{s_3\}$;

$t_3^{(1)}$ is present with ${}^\bullet t_3^{(1)} = \{s_1, p_{a,4}\}$ and ${t_3^{(1)}}^\bullet = \{s_7\}$;

$t_1^{(2)}$ is present with ${}^\bullet t_1^{(2)} = \{s_1, p_{\varepsilon,1}\}$ and ${t_1^{(2)}}^\bullet = \{s_8\}$;

$t_2^{(2)}$ is present with ${}^\bullet t_2^{(2)} = \{s_3, p_{a,3}\}$ and ${t_2^{(2)}}^\bullet = \{s_8\}$;

$t_3^{(2)}$ is present with ${}^\bullet t_3^{(2)} = \{s_7, p_{a,7}\}$ and ${t_3^{(2)}}^\bullet = \{s_8\}$;

positive closure: $t_3^{(3)}$ is present with ${}^\bullet t_3^{(3)} = \{s_3, p_{a,3}\}$ and ${t_3^{(3)}}^\bullet = \{s_1^+\}$;

t_1^+ is present with ${}^\bullet t_1^+ = \{s_1^+, p_{a,2}\}$ and ${t_1^+}^\bullet = \{s_3\}$;

exponentiation: $t_{1,1,1}^{(4)}$ is present with ${}^\bullet t_{1,1,1}^{(4)} = \{s_4, p_{a,4}\}$ and $t_{1,1,1}^{(4)\bullet} = \{s_6\}$;

$t_{1,1,2}^{(4)}$ is present with ${}^\bullet t_{1,1,2}^{(4)} = \{s_5, p_{b,5}\}$ and $t_{1,1,2}^{(4)\bullet} = \{s_7\}$;

$t_{1,1}^{(5)}$ is present with ${}^\bullet t_{1,1}^{(5)} = \{s_6, p_{a,6}\}$ and $t_{1,1}^{(5)\bullet} = \{s_{1,1}^{exp}\}$;

$t_{1,2}^{(5)}$ is present with ${}^\bullet t_{1,2}^{(5)} = \{s_7, p_{\varepsilon,7}\}$ and $t_{1,2}^{(5)\bullet} = \{s_{1,2}^{exp}\}$;

$t_{1,1}^{exp}$ is present with ${}^\bullet t_{1,1}^{exp} = \{s_{1,1}^{exp}, p_{a,4}\}$ and $t_{1,1}^{exp\bullet} = \{s_6\}$;

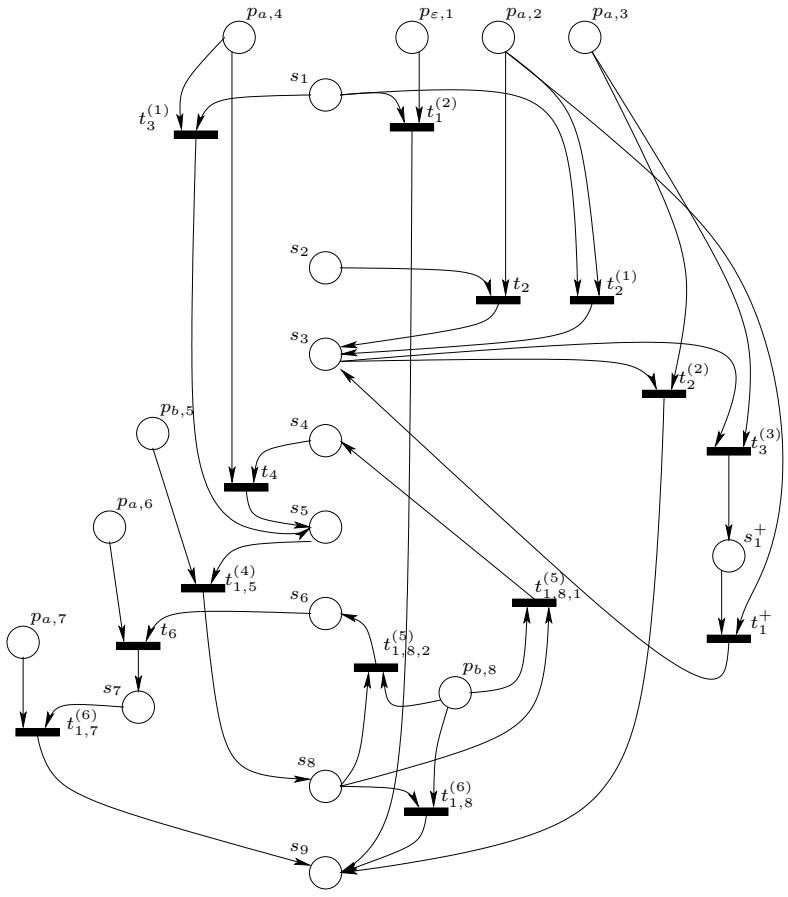

Figure 3: Partially depicted net underlying the P/T system defined in Section 4.1.

$t_{1,2}^{exp}$ is present with ${}^\bullet t_{1,2}^{exp} = \{s_{1,2}^{exp}, p_{b,5}\}$ and $t_{1,2}^{exp\bullet} = \{s_7\}$;

$t_{1,1}^{(6)}$ is present with ${}^\bullet t_{1,1}^{(6)} = \{s_{1,1}^{exp}, p_{a,6}\}$ and $t_{1,1}^{(6)\bullet} = \{s_5\}$.

extra: $t_{a,j}$ are present with ${}^\bullet t_{a,j} = \{s_8, p_{a,j}\}, t_{a,j}^\bullet = \{s_z\}$ and $t_{b,j}$ are present with ${}^\bullet t_{b,j} = \{s_8, p_{b,j}\},\ t_{b,j}^\bullet = \{s_z\},\ 1 \leq j \leq 8$.

The just described P/T system is partially depicted in Figure 3. All the work places and only the input places $p_{\varepsilon,1}, p_{a,2}, p_{a,3}, p_{a,4}, p_{b,5}, p_{a,6}$ and $p_{\varepsilon,7}$ are depicted; all transitions except $t_1^{(8)}$ and except the ones related to *extra* are depicted, too.

4.2 Initial configurations and firing sequences

Here we consider five initial configurations and we describe some of the firing sequences for each of these configurations. We recall that, as the P/T system is non-deterministic, for each initial configuration the system can have more than one firing sequence.
Let us consider the word $w_1 = a^4bba^4$ which belongs to $\mathsf{L}(\alpha)$ the J language defined by the J expression α. As indicated earlier only for words in $\mathsf{L}(\alpha)$ it is possible to have a proper encoding.

(First case) There is one token in s_1, two tokens in $p_{a,4}$, one token in $p_{b,5}$, two tokens in $p_{a,6}$ and one token in $p_{\varepsilon,7}$. One firing sequence associated to this initial configuration is $t_3^{(1)}t_{1,1}^{(5)}t_{1,1}^{exp}t_{1,1}^{(6)}t_{1,1,2}^{4}t_{1,2}^{(6)}$.

The firing of $t_{1,6}^{(6)}$ let a token to be put in s_8, final state of the P/T system, and no other firing occurs. In this way the word is accepted reflecting the fact that $a^2ba^2 \in \mathsf{L}(\alpha)$.

Another firing sequence associated to this initial configuration is $t_3^{(1)}t_{1,1}^{(6)}t_{1,1,2}^{(4)}t_{1,2}^{(6)}t_{a,4}$. The firing of $t_{a,4}$ let a token to be put in s_z. The word is not accepted.

(Second case) There is one token in s_1, two tokens in $p_{a,2}$, two tokens in $p_{a,3}$, one token in $p_{b,5}$, one token in $p_{b,6}$ and four tokens in $p_{a,7}$.

The firing sequence $t_2t_3^{(3)}t_1^{+}t_2^{(2)}t_{b,5}$ is associated to this encoding. The firing of $t_{b,5}$ let a token to be put in s_z. The input vector is not accepted.

(Second case) There is one token in s_1, four tokens in $p_{a,4}$ and one token in $p_{b,5}$.

The only possible firing sequence associated to this initial configuration is: $t_3^{(1)}$. The input vector is not accepted and there is no other firing sequence accepting the initial configuration. This reflects the fact that $a^4b \notin \mathsf{L}(\alpha)$.

(Third case) There is one token in s_1, one token in $p_{a,4}$, one token in $p_{b,5}$, two tokens in $p_{a,6}$ and one token in $p_{\varepsilon,7}$.

The firing sequences $t_3^{(1)}t_{1,1}^{(5)}$ and $t_1^{(3)}t_{1,1}^{(6)}t_{1,1,2}^{(4)}t_{1,2}^{(6)}t_{a,6}$ are associated to this initial configuration. In both cases the input vector is not accepted and there is no other firing sequence accepting the initial configuration. This reflects the fact that $aba^2 \notin \mathsf{L}(\alpha)$.

(Fourth case) There is one token in s_1, one token in $p_{a,2}$ and one token in $p_{a,3}$. One firing sequence associated to this initial configuration letting the P/T system not to accept the initial vector is $t_2t_3^{(3)}$, while firing sequences letting the P/T system to accept the initial vector are: $t_2t_2^{(2)}$ and $t_2^{(1)}t_2^{(2)}$. This reflects the fact that $aa \in \mathsf{L}(\alpha)$.

(Fifth case) There is one token in s_1, two tokens in $p_{a,2}$ and two tokens

in $p_{a,3}$. Two firing sequence associated to this initial configuration letting the P/T system not to accept the initial vector are: $t_2 t_2^{(2)} t_{a,2}$ (this last transition is not depicted in Figure 3), $t_2 t_3^{(3)} t_1^{+} t_3^{(3)}$, while one firing sequence letting the P/T system to accept the initial vector is: $t_2 t_3^{(3)} t_1^{+} t_2^{(2)}$. This reflects the fact that $a^2 a^2 \in \mathsf{L}(\alpha)$.

5 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

$$S = \{v \mid v = v_0 + i_1 v_1 + \cdots + i_t v_t,\ i_j \in N\}.$$

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 Σ^*, 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 2 *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.*

Theorem 3 *The Parikh map of every language denoted by a J expression is semilinear.*

The proof is an induction on the form of the J expression. Let α be a J expression and $\mathsf{L}(\alpha)$ be the language it denotes.

Basis. If $\alpha = \varepsilon$, then $\mathsf{L}(\alpha) = \{\varepsilon\}$, and $\psi(\mathsf{L}(\alpha)) = \{(0, ..., 0)\}$ is semilinear.

Induction hypothesis. If $\alpha = a_i$, where $a_i \in \Sigma$, then $\mathsf{L}(\alpha) = \{a_i\}$, and $\psi(\mathsf{L}(\alpha)) = \{(0, .., , 0, 1, 0, ..., 0)\}$ (where 1 is in the i-th coordinate) is semilinear.

Inductive step. Suppose α_1 and α_2 are J expressions denoting languages whose Parikh maps are semilinear sets generated by finite-state generators M_1 and M_2, respectively.

1. (*Union*). The language denoted by the J expression $\alpha_1 \cup \alpha_2$ has a semilinear Parikh map, since semilinear sets are closed under union. (Also, it can be generated by a finite-state generator M that non-deterministically simulates M_1 or M_2.)
2. (*Concatenation*). Construct a finite-state generator M which simulates M_1 and when M_1 halts, M then simulates M_2. It follows that M generates the semilinear set for the concatenation.
3. (*Positive closure*). Construct from M_1 a finite-state generator M which simulates halting computations of M_1 a non-deterministic number of times, $r > 0$. Then M will generate the semilinear set for α_1^+
4. (*Exponentiation*). The idea is similar to concatenation and positive closure. We illustrate the construction for an exponentiation having six exponentiation-terms and two integer variables. The construction easily generalizes for any number of exponentiation-terms and integer variables.

   Let $\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma_1, \gamma_2$ be J expressions not containing the operators union and positive closure (as requested by the definition of exponentiation in Definition 3). Let $A_1, A_2, B_1, B_2, C_1, C_2$ be the finite-state generators for the semilinear sets of these expressions.

   Consider the J expression (obtained by exponentiation) $E = \alpha_1^r\ \alpha_2^s\ \beta_1^r \beta_2^s \gamma_1^r \gamma_2^s$. We show that the Parikh map of the language denoted by E is semilinear. Assume that these J expressions are non-null (i.e, none is equal to ε, but they may contain the null string). Clearly, the Parikh map of the language denoted by E is the same as the Parikh map of the language denoted by $E' = \alpha_1^r \beta_1^r \gamma_1^r \alpha_2^s \beta_2^s \gamma_3^s$, which is the same as that of $E'' = (\alpha_1 \beta_1 \gamma_1)^r (\alpha_1 \beta_1 \gamma_1)^s$, although E' and E'' need not be J expressions. We show that the Parikh map of E'' is semilinear.

   Construct a finite-state generator M which operates as follows: M simulates a halting computation of A_1, then that of B_1, then that of C_1. It iterates this process a non-deterministic number of times, $r > 0$. We denote this process by $(A_1 B_1 C_1)^+$. Then M executes

$(A_2B_2C_2)^+$. Clearly, an n-tuple of numbers is generated in the counters of M if and only if the n-tuple is in the semilinear set for E''.

Suppose some of the J expressions are null, i.e, ε, e.g., suppose $\beta_1 = \varepsilon$. Then $E = \alpha_1^r\alpha_2^s\beta_2^s\gamma_1^r\gamma_2^s$. In this case, M executes $(A_1C_1)^+$ followed by $(A_2B_2C_2)^+$.

Considering Remark 1 and the idea of concatenation, the construction for exponentiations with also fixed positive integers easily follows. □
We will show a "converse" of Theorem 3.

Definition 5 *Let $S \subseteq N^n$ and $\Sigma = \{a_1, ..., a_n\}$. Define the language $L_S = \{a_1^{s_1}a_2^{s_2}...a_n^{s_n} \mid (s_1, ..., s_n) \in S\}$.*

Theorem 4 *If S is a semilinear set, then L_S is a J language.*

Proof. Let $S \subseteq N^n$ with generators $v_1, ..., v_m$. For notational convenience, we illustrate the construction only for $n = 3$ and $m = 2$. The technique works for any $n, m \geq 1$. Let $v_i = (k_{i1}, k_{i2}, k_{i3})$ for $1 \leq i \leq 3$. Define the following J expressions obtained by exponentiation:

1. $E_1 = (a^{k_{11}})^r(a^{k_{12}})^s(a^{k_{13}})^t(b^{k_{11}})^r(b^{k_{12}})^s(b^{k_{13}})^t$
2. E_2 which is derived from E_1 by deleting the segment with exponent r.
3. E_3 which is derived from E_1 by deleting the segment with exponent s.
4. E_4 which is derived from E_1 by deleting the segment with exponent t.
5. E_5 which is derived from E_1 by deleting the segments with exponents r and s.
6. E_6 which is derived from E_1 by deleting the segments with exponents r and t.
7. E_7 which is derived from E_1 by deleting the segments with exponents s and t.

Because exponentiation in J expression requires the exponents to be positive, we add expressions $E_2, ..., E_7$ to handle the cases when one or more exponents (but not all) are zero. The J expression corresponding to the linear set S is then
$E = E_1 \cup E_2 \cup E_3 \cup E_4 \cup E_5 \cup E_6 \cup E_7 \cup \varepsilon$
We added ε to take care of the case when $r = s = t = 0$. □

6 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.

Theorem 5 *Every J language can be accepted by an LCM.*

Proof. The proof is an induction on the form of the J expression. Clearly, if $\alpha = \varepsilon$ or $\alpha = a_i$, where $a_i \in \Sigma$, $\mathsf{L}(\alpha)$ can be accepted by an LCM (even without using the counters, i.e., they remain at zero value). Let α_1 and α_2 be J expressions whose associated languages L_1 and L_2 are accepted by LCMs M_1 and M_2, respectively. It is easy to construct LCMs to accept $L_1 \cup L_2$, $L_1 L_2$, and L_1^+. For example, $L_1 L_2$ can be accepted by an LCM M that processes the one-way input string by simulating M_1 on the first part of the input. When M_1 accepts, M simulates M_2.

We now look at exponentiation.

(*) First we observe that if J expressions α and β both contain ε, then the language denoted by the exponentiation expression $\alpha^n \beta^n$ is identical to the language denoted by $\alpha^+ \beta^+ = \alpha^n \beta^p$, where n and p are variables that can take on positive integer values, but they are independent (not related). We say that α and β are independent.

We illustrate the construction for an exponentiation having six exponentiation-terms and two integer variables. The construction easily generalizes for any number of exponentiation-terms and integer variables. Let $\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma_1, \gamma_2$ be J expressions not containing the operators union and positive closure (as requested by the definition of exponentiation in Definition 3). Let $A_1, A_2, B_1, B_2, C_1, C_2$ be the LCMs accepting the languages denoted by these expressions.

Consider the J expression (obtained by exponentiation) $E = \alpha_1^r \alpha_2^s \beta_1^r \beta_2^s \gamma_1^r \gamma_2^s$. We show that the language denoted by E can be accepted by an LCM. Assume that these J expressions are non-null (i.e, none is equal to ε, but they may contain the null string).

(**) First assume that at least one of the J expressions $\alpha_1, \beta_1, \gamma_1$ does not contain ε, and at least one of the expressions $\alpha_2, \beta_2, \gamma_2$ does not contain ε.

M will simulate the computations of $A_1, A_2, B_1, B_2, C_1, C_2$ in the following way. M uses four new counters, d_1, d_1', d_2, d_2' M reads the (one-way) in-

put and simulates accepting computations of A_1 $r \geq 1$ times, where r is non-deterministically chosen. Each time it simulates an accepting computation, it increments counters d_1 and d_1' by 1. After simulating r times, M simulates $s \geq 1$ accepting computations of A_2. Again s is chosen non-deterministically and recorded in counters d_2 and d_2'. Then M simulates r accepting computations of B_1, using counter d_1 (it decrements this counter by 1 after every accepting simulation). M continues by simulating s accepting computations of B_2, using counter d_2. Finally, M simulates r accepting computations of C_1 using counter d_1' followed by the simulation of s accepting computations of C_2 using counter d_2'. It is easily verified that $\mathsf{L}(E)$ is accepted by M. Clearly, by assumption (**), M's counters are linearly bounded.
Now suppose all the J expressions $\alpha_1, \beta_1, \gamma_1$ contain ε. Then by (*), the expressions are independent, Hence, counters d_1 and d_1' are no loger needed, as M can simulate the accepting computations of A_1, B_1, C_1 without comparing the numbers of accepting computations they make. Similarly, if all the J expressions $\alpha_2, \beta_2, \gamma_2$ contain ε, then counters d_2, d_2' are no longer needed.

Considering Remark 1 and the LCM accepting concatenations of J expressions, the construction for exponentiations with also fixed positive integers easily follows. For instance, given an exponentiation with both integer variables and fixed positive integers it is possible to create an LCM that first accepts the exponentiation-terms to the power of an integer variable and then the exponentiation-terms to the power of a fixed positive integer. □

Corollary 1 *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 2 *There are context-free languages that are not J languages.*

7 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 Σ be the set of terminal symbols. The non-terminal symbols are partitioned into two disjoint sets, $\mathcal{Q}$ and $\mathcal{R}$. There is a unique start non-terminal $S_0 \in \mathcal{Q}$ 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 $S \in \mathcal{Q}$ does not appear on the RHS of any basic rule, but can appear in a matrix rule 6 below.

2. $S \rightarrow S_1|S_2$, where S, S_1, S_2 are distinct non-terminals in $\mathcal{Q}$, and S does not appear on the RHS of any basic rule, but can appear in a matrix rule 6 below.

3. $S \rightarrow S_1S_2$, where S, S_1, S_k are distinct non-terminals in $\mathcal{Q}$, and S does does not appear on the RHS of any basic rule, but can appear in a matrix rule 6 below.

4. $S \rightarrow SS$, where $S \in \mathcal{Q}$ does not appear on the RHS of any basic rule (except in this rule), but can appear in a matrix rule 6 below.

5. $S \rightarrow (A_{11}A_{12}...A_{1m}, ..., A_{k1}A_{k2}...A_{km})$, where $m \geq 1$, $k \geq 1$, each A_{ij} is a non-terminal in $\mathcal{R}$ and $S \in \mathcal{Q}$ 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. $[A_1 \rightarrow S_1A_1, ..., A_k \rightarrow S_kA_k]$, where $k \geq 1$, each A_i a non-terminal in $\mathcal{R}$, and each $S_i \in \mathcal{Q}$ (subject to the restriction in rule 5 above).

   Restriction 1: We require that if $[A_1 \rightarrow S_1A_1, ..., A_k \rightarrow S_kA_k]$ and $[A_1 \rightarrow S'_1A_1, ..., A_k \rightarrow S'_kA_k]$ 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. $[A_1 \rightarrow w_1, ..., A_k \rightarrow w_k]$, where $k \geq 1$, each A_i a non-terminal in $\mathcal{R}$, each w_i in Σ^*

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 $(A_{11}A_{12}...A_{1m}, ..., A_{k1}A_{k2}...A_{km})$. 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 $\mathcal{Q}^+$, we "merge" the k components into a single string. Then the derivation continues until w is reached.

Theorem 6 *The languages generated by ERLSMGs are exactly the J languages, which allow union and positive closure in exponentiation.*

Proof. First we show that every J language can be generated by an ERLSMG. We do this by induction.

Basis. To generate $\{\varepsilon\}$, the grammar has one rule $S \rightarrow \varepsilon$.

Induction hypotesis. To generate $\{a\}$, the grammar has one rule $S \rightarrow a$.

Induction step **(Union and Concatenation)** Suppose S_1 and S_2 are start non-terminals for grammars G_1 and G_2 generating languages L_1 and L_2. Assume the non-terminals in the grammars are disjoint. Let S and S' be symbols not in the non-terminals of the two grammars. Then

The grammar for $L_1 \cup L_2$ consists of the rules in G_1 and G_2 and a new rule $S \rightarrow S_1 \mid S_2$, with S the new start symbol.

The grammar for $L_1 L_2$ consists of the rules in G_1 and G_2 and a new rule $S' \rightarrow S_1 S_2$, with S' is the new start symbol.

(Positive closure) Suppose S is the start non-terminal for grammar G generating language L. Let S' be a new start non-terminal. Then the grammar for L^+ consists of the rules in G and two new rules $S' \rightarrow S'S'$ and $S' \rightarrow S$.

(Exponentiation) Here, we use rule 5 and the right-linear simple matrix rules. We illustrate by an example. Let S_1, S_2, S_3, S_4 be the start non-terminals for grammars G_1, G_2, G_3, G_4 generating languages L_1, L_2, L_3, L_4. Suppose we want to generate the language $L = \{x_{11}...x_{1r}x_{21}...x_{2s}x_{31}...x_{3r}x_{41}...x_{4s} | r, s \geq 1, x_{1i} \in L_1, x_{3i} \in L_3, 1 \leq i \leq r, x_{2j} \in L_2, x_{4j} \in L_4, 1 \leq j \leq s\}$. Assume that the non-terminals of the G_i's are distinct. Let S, A_1, A_2, B_1, B_2 be new non-terminals, with S the new start non-terminal. The rules of the grammar for L are all the rules in the G_i's plus the following rules:

$S \rightarrow (A_1B_1, A_2B_2)$

$[A_1 \rightarrow S_1A_1, A_2 \rightarrow S_3A_2]$

$[B_1 \rightarrow S_2B_1, B_2 \rightarrow S_4B_2]$

$[A_1 \rightarrow \varepsilon, A_2 \rightarrow \varepsilon]$

$[B_1 \rightarrow \varepsilon, B_2 \rightarrow \varepsilon]$

Clearly, from S, it is possible to derive $(S_1^r S_2^s, S_3^r S_4^s)$ for $r, s \geq 1$, which when merged becomes $S_1^r S_2^s S_3^r S_4^s$.

The converse, i.e., that every ESRLMG language is a J language can also be shown. We omit the proof. □

Corollary 3 *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.*

8 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 *joins* and *fork* can facilitate the study of the computational power of models of *membrane systems* (also known as *P systems*) [9] based on multiset rewriting. These results use a definition of *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 *fork*, *join* and their composition, then the results on the computational power of P/T systems whose underlying net is composed of *joins* and *fork* are also valid to S.

In [3, 2] it is shown that P systems with catalysts can simulate a *fork* using rules of the kind $a \to b_1 b_2$, while the simulation of a *join* does not require the use of such rules. So, knowing from [3, 2] how P systems with catalysts can simulate *join* and Theorem 1, we can say that the family of languages generated by P systems with catalysts not using rules of the kind $a \to b_1 b_2$ is J.

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

Corollary 4

The family of languages accepted by P systems with catalysts of degree 2 and 2 catalysts not using rules of the kind $a \to b_1 b_2$ is J;

the family of languages accepted by purely catalytic P systems of degree 2 and 3 catalysts not using rules of the kind $a \to b_1 b_2$ is J.

We end this paper with an open problem.

In the rule of form 6, we had a restriction that if $[A_1 \to S_1 A_1, ..., A_k \to S_k A_k]$ and $[A_1 \to S'_1 A_1, ..., A_k \to S'_k A_k]$ 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. P systems, Petri nets, and Program machines. In R. Freund, G. Lojka, M. Oswald, and G. Păun, editors, *Membrane Computing. 6th International Workshop, WMC 2005, Vienna, Austria, July 18-21, 2005, Revised Selected and Invited Papers*, volume 3850 of *Lecture Notes in Computer Science*, pages 209–223. Springer-Verlag, Berlin, Heidelberg, New York, 2006.

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

[3] P. Frisco. *Computing with Cells. Advances in Membrane Computing.* Oxford University Press, 2009. to appear.

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

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

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

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

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

[9] G. Păun. Computing with membranes. *Journal of Computer and System Science*, 1(61):108–143, 2000.

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

[11] W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science.* Springer-Verlag, Berlin, Heidelberg, New York, 1998.