P Systems and Topology: Some Suggestions for Research

Pierluigi Frisco

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

Summary. Lately, some studies linked the computational power of abstract computing systems based on multiset rewriting to 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 just looking at their *topology*, that is, information flow.

This line of research is very promising for several aspects:

- its results are valid for a broad range of systems based on multiset rewriting;
- it allows to know the computational power of abstract computing devices without tedious proofs based on simulations;
- it links computational power to topology and, in this way, it opens a broad range of questions.

In this note we summarize the known result on this topic and we list a few suggestions for research together with the relevance of possible outcomes.

1 Introduction

Imagine that a computing device based on multiset rewriting is defined. Let us call this computing device S_1. This could be a P system with symport/antiport, or with catalysts, of with conformons, or it could be a definition of Diophantine equations, or some model of Brane calculi of whatever else you like.

What would you do in order to know the computing power of such a system?

Probably you would try to simulate with S_1 another computing system, say S_2 with known computational power. If this is possible, than you can say that S_1 can compute at least as much as S_2 can compute. Then you would probably try to simulate S_1 with S_2, if this is possible, then you know that the two systems have the same computational power. This is the standard way to know the computational power of a computing system.

There is another way to know the computational power of S_1. This new way is based on the fact that a computing system has a way to store information and a way to manipulate it. This other way looks at how such a system stores information

and how it manipulates it and it deducts (in between other things) the computing power of S_1.

From a point of view this is not a new idea: this is a very well establish concept in formal grammars. If somebody gives you a formal grammar S_1 and asks what it can compute, probably you would not try to simulate with S_1 another grammar S_2 of known computing power. Instead, you would simply look at the productions of S_1 and from this (because of the Chomsky hierarchy) you would be able to deduct what it can be generated by S_1.

The approach that we are going to consider in this paper is about this: it shows how to know the computing power of a formal system based on multiset rewriting without running simulations but simply looking at the kind of operations that the formal system can perform.

The suggestions for research present in Section 7 have been in part inspired by conversations taking place during the 7^{th} Brainstorming Week on Membrane Computing (February 2 - 6, 2009, Seville, Spain). This note is not self contained, it has been written having in a mind readers knowledgeable in P systems and with a strong interest in Petri nets. Citation indicate where the used but not defined concepts can be found.

2 About Simulations

If you are familiar with formal language theory, then you know that productions in a formal grammar are all of the same form: $\alpha \to \beta$ with α and β strings over a certain alphabet. There is not much confusion about what such a production does and about the language generated by a grammar. On the other hand, formal systems based on multiset rewriting do not have 'standard' way to operate and do not have 'standard' ways to get the result of their computation. For this reason, if we want to 'reduce' the way these systems operate to just one way (on which we can analyze the topology), then we have to use a definition of simulation.

Let S and S' be two formal systems with O and O' their respective sets of operations and $\mathbb{C} = \{c_1, c_2, \ldots\}$ and $\mathbb{C}' = \{c'_1, c'_2, \ldots\}$ their respective sets of configurations. We denote with $\overset{\sigma}{\Rightarrow}$ ($\overset{\sigma'}{\Rightarrow}$), σ multiset over O (σ' multiset over O'), the transition from one configuration to another in a computation of S (S') according to the application of the operations in σ (σ'). With $\overset{\sigma_1,\ldots,\sigma_n}{\Rightarrow^+}$ ($\overset{\sigma'_1,\ldots,\sigma'_n}{\Rightarrow^+}$) we denote non-empty sequences of transitions from one configuration to another in a computation of S (S') according to the application of the operations in $\sigma_1, \ldots, \sigma_n$ ($\sigma'_1, \ldots, \sigma'_n$) in sequence. So, for instance, if $c_1 \overset{\sigma_1}{\Rightarrow} c_2 \overset{\sigma_2}{\Rightarrow} c_3$, then we can write $c_1 \overset{\sigma_1,\sigma_2}{\Rightarrow^+} c_3$.

It should be clear that, depending on the operational mode of S, the multiset σ can be a multiset of a specific kind. For instance, σ can be such that it returns at most 1.

Definition 1. *Let S and S' be two formal systems with O and O' their respective sets of operations, $\mathbb{C}$ and $\mathbb{C}'$ their respective sets of configurations and c_{init} and c'_{init} their respective initial configurations.*

We say that S simulates S' if there are two relations $\alpha \subseteq \mathbb{C} \times \mathbb{C}'$ and $\beta \subseteq O^n \times O'^m$, $n, m \in \mathbb{N}$, such that:

i) $(c_{init}, c'_{init}) \in \alpha$;

ii) *for all* $c_1, c_2 \in \mathbb{C}$, $c'_1 \in \mathbb{C}'$ *and* $\sigma \in O^n$: *if* $c_1 \overset{\sigma}{\Rightarrow}^+ c_2$ *and* $(c_1, c'_1) \in \alpha$, *then there is* $c'_2 \in \mathbb{C}'$ *such that* $c'_1 \overset{\sigma'}{\Rightarrow}^+ c'_2$ *with* $(c_2, c'_2) \in \alpha$ *and* $(\sigma, \sigma') \in \beta$;

iii) *for all* $c'_1, c'_2 \in \mathbb{C}'$, $c_1 \in \mathbb{C}$ *and* $\sigma' \in O^n$: *if* $c'_1 \overset{\sigma'}{\Rightarrow}^+ c'_2$ *and* $(c_1, c'_1) \in \alpha$, *then there is* $c_2 \in \mathbb{C}$ *such that* $c_1 \overset{\sigma}{\Rightarrow}^+ c_2$ *with* $(c_2, c'_2) \in \alpha$ *and* $(\sigma, \sigma') \in \beta$.

If S $\alpha\beta$ simulates S', then S is called the *simulating* system while S' is called the *simulated* system.

It is important to stress that in this paper the accepted or generated languages of simulations are related to the configurations, not to the labels associated to the operations (as in other definitions of simulation). Moreover, we have to point out that the just given definition of simulation differs substantially from the ones present in [11, 12] and from other similar definitions (specific to EN systems) as in [14].

3 Petri Nets

If we want to study how the topology of formal system based on multiset rewriting is related to their computational power, then we need one such system in which topology is clearly present. *Place/transition system* (*P/T systems*) are the ideal candidate. They are a type of Petri nets and, as such, are a (bipartite) graph. Similarly to grammars, one of the nice features of P/T systems is their simplicity: the operations that can be performed by them are very basic, no complexity hidden in the way they operate. In this way, the set of numbers that can be generated a P/T systems can be directly related to its topology, similarly to the way the language generated by a grammar can be directly related to its kind of productions.

Given an P/T system it is possible to 'run' it (that is, the way it fires) in different ways. This is different than other formal systems for which their way to run is embedded in their definition. For instance, in a grammar the productions are applied once per time (that is, given a sentential form at most one production is applied in order to pass to another sentential form) while in an L system productions are applied in parallel. So, if someone would give you a grammar, then you would also know the way it runs. This means that if together with a grammar you are also given a language, then it can either be that the grammar generates that language or not.

P/T systems (and Petri nets in general) do not have a unique way to run. If someone would give you a P/T system, then you could run it in different ways (each of these ways generates a *configuration graph* of a certain kind). This means that if together with the P/T system you are also given a set of numbers, then the P/T system can generate or not that set of numbers depending on the way it runs.

This separation between a P/T system and the way it runs is important to us as some of the results indicated in the following depend on a specific way to run while others are independent of the way to run.

For us it is important to say that we consider three types of Petri nets: *EN system*, *P/T systems* and *P/T′ systems* and we consider two ways they can run: SCG and $MSCG$. It is improper to refer to SCG and $MSCG$ as 'ways to run'. Here we use this improper language because we want to use a very simple terminology. The readers who are not familiar with these Petri nets concepts can find their definitions in [13, 5]. These concepts are important for understanding what in the following sections.

4 Building Blocks and Their Composition

Let us introduce the nets depicted in Fig. 1 and call them *building blocks*, *join* and *fork* in particular, as depicted in that figure. The places present in each building block are distinct.

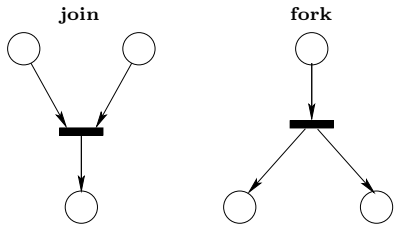

Fig. 1. Building blocks: *join* and *fork*. © With kind permission of Springer Science and Business Media [4].

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 $\bar{t}_x^{\bullet} \cap {}^{\bullet}\hat{t}_y \neq \emptyset$ and ${}^{\bullet}\bar{t}_x \cap {}^{\bullet}\hat{t}_y = \emptyset$. We say that* x and y are in parallel *if ${}^{\bullet}\bar{t}_x \cap {}^{\bullet}\hat{t}_y \neq \emptyset$ and $\bar{t}_x^{\bullet} \cap {}^{\bullet}\hat{t}_y = \emptyset$.*

We say that a net is composed by *building blocks (it is* composed by x*) if it can be defined by building blocks (it is defined by* x*) sharing places but not*

transitions. Consequently, we say that a Petri net is composed by *building blocks (it is* composed by x*) if its underlying net is composed by building blocks (it is* composed by x*).*

In Fig. 2 a *join* and a *fork* are depicted in parallel, while in Fig. 3.a and Fig. 3.b *join* and *fork* are depicted in sequence.

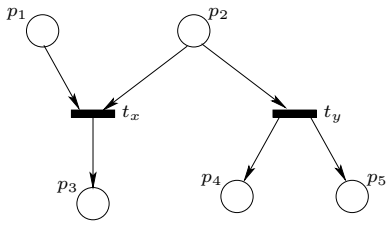

Fig. 2. A *join* and a *fork* in parallel.

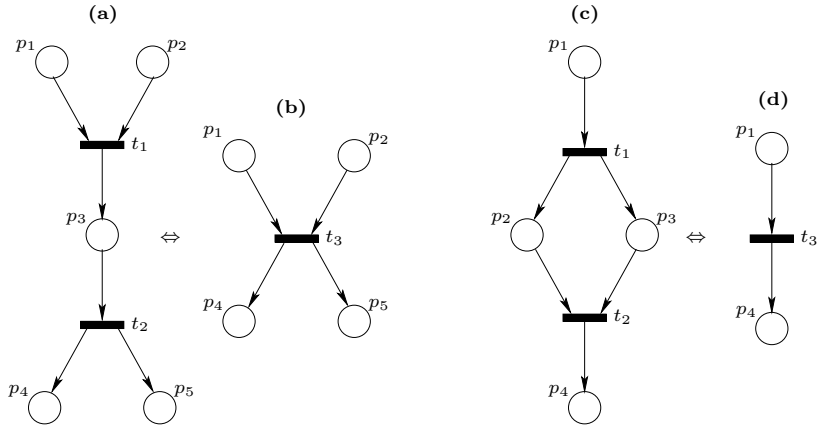

Fig. 3. (a) *join* and *fork* in sequence and **(b)** *fork* and *join* in sequence.

5 Known Results

Table 1 lists the known results linking topology to accepted/generated vectors of numbers.

n.	*system*	*build. blocks*	*composition*	*n. places*	*way to run*	*acc./gen.*	*class*
1	P/T	join, fork	any	finite	SCG	acc.	= part. blind r.m.
2	P/T	join, fork	as Fig. 3.a	finite	$MSCG$	acc.	= restricted r.m.
3	P/T	join, fork	any	finite	$MSCG$	acc./gen.	$= \mathbb{N}\cdot\mathsf{RE}$
4	EN	join	any	finite	SCG	acc.	$= \mathbb{N}\cdot\mathsf{FIN}$
5	P/T$'$	join	any	infinite	SCG	acc./gen.	$= \mathbb{N}\cdot\mathsf{RE}$
6	P/T	join	any	finite	SCG	acc.	$\mathbb{N}\cdot\mathsf{REG} \subset J \subset \mathbb{N}\cdot\mathsf{CS}$

Table 1. Summary of known results

In Table 1:

n. refers to the row in the table;
system indicates the kind of Petri net;
build. blocks indicated the kinds of building blocks present in the system;
composition indicates how the building blocks are composed in the system;
n. places indicates the number of places present in the system;
way to run indicates the way the system is run;
acc./gen. indicates if known results refers to the accepting or generating model;
class indicates the class of numbers accepted/generated by the considered system.

Moreover:

part. blind r.m. means *partially blind register machines* [8];
restricted r.m. means *restricted register machines* [9];
$\mathbb{N}\cdot\mathsf{FIN}, \mathbb{N}\cdot\mathsf{REG}, \mathbb{N}\cdot\mathsf{CS}, \mathbb{N}\cdot\mathsf{RE}$ denote classes of numbers [15].

Because of Theorem 2 in [5] row 3 in Table 1 holds true also for other ways to run the P/T system included P/T systems having an infinite number of places and running according to *SCG*.

6 Links with P Systems with Catalysts

In the introduction we hinted to a novel approach to study the computational power of abstract computing devices. This novel approach is based on the simulation (according to Definition 1) of *join* and *fork* and their composition. Once this is known, then the results listed in Table 1 can be used to deduct the computing power of the abstract computing device under investigation.

This has been performed for *catalytic P systems*. Here we list the results obtained using this novel approach for this model of P system. The following results can be found in [5] together with the definitions of the used notation and terminology.

Lemma 1. *The building blocks* **join** *and* **fork** *can be simulated by generating P systems with catalysts of degree 1 and with 2 catalysts.*

Lemma 2. *Generating P systems with 2 catalysts and one compartment can simulate the 0-test P/T system.*

From these two lemmas and the results in Table 1 the following holds true:

Theorem 1. $\mathbb{N}_{02} \cdot aCP(1,2) = \mathbb{N}_{02} \cdot aCatP(1,3) = \mathbb{N}_2 \cdot \mathsf{RE}$;
$\mathbb{N} \cdot gCP(2,2) = \mathbb{N} \cdot gCatP(2,3) = \mathbb{N} \cdot aCP_{-c}(1,2) = \mathbb{N} \cdot aCatP_{-c}(1,3) = \mathbb{N} \cdot \mathsf{RE}$.

Theorem 2. *The families* $\mathbb{N} \cdot gCP(2,2)$, $\mathbb{N} \cdot gCatP(2,3)$, $\mathbb{N}_{02} \cdot aCP(1,2)$, $\mathbb{N}_{02} \cdot aCatP(1,3)$, $\mathbb{N} \cdot aCP_{-c}(1,2)$, $\mathbb{N} \cdot aCatP_{-c}(1,3)$, *when maximal parallelism is not present, are the ones generated also by partially blind register machines.*

Corollary 1. *Accepting catalytic-P systems with only rules of the kind* $cx \to c$, $c \in C$ *and* $x \in V \setminus C$ *can accept only finite languages.*

Corollary 2. *Restricted P systems with catalysts of degree 2 and two catalysts and restricted catalytic-P systems of degree 2 and three catalysts can simulate restricted register machines.*

Moreover:

Corollary 3. *The class of numbers 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 class of numbers 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.*

Where J is a class of numbers such that $\mathbb{N} \cdot \mathsf{REG} \subset J \subset \mathbb{N} \cdot \mathsf{CS}$ (that is, the one in row 6 of Table 1).

Some of the results in this section (as, for instance, Theorem 1) have previously been obtained using 'classical' (that is, simulating other devices of known computing power) methodologies [2, 10].

7 Suggestions for Research

The results in Section 6 show how far reaching the simulation of join and fork can be in respect to the direct simulation of a specific computational model. We say 'can be' and not 'is' because it is not always the case that some features considered for building blocks can be naturally translated in features of the simulating system (P systems with catalysts, in this case). Corollary 2 is an example of the difficulties in this 'translation'. How to translate in natural terms for P systems with catalysts the fact that *join* and *fork* are composed as in Fig. 3.a?

The definition of restricted P systems with catalysts considered for Corollary 2 is a very simple way to perform such a translation, but still it did not allow to say that the computational power of such P systems is equivalent to the one of restricted register machines. For this reason we formulate:

Suggestion for research 1 *Define a model of P systems with catalysts having computational power equivalent to the one of restricted register machines.*

Line 6 in Table 1 refers to the class of numbers accepted by P/T systems composed only by *join* and having a finite number of places when they run according to *SCG*. This class of number, called J in [7], is well defined: it is proved that this kind of P/T systems can only accept J. What it is not well known is how J relates to other classes of numbers.

Suggestion for research 2 *As indicated in Table 1, it is known that* $\mathbb{N}\cdot\mathsf{REG} \subset J \subset \mathbb{N}\cdot\mathsf{CS}$. *Restrict this (rather broad) interval.*

A solution to this suggestion does not necessarily mean to have a result of the kind: $A \subset J \subset B$ where A and B are classes of numbers. The indication of what part of $\mathbb{N}\cdot\mathsf{CS}$ is in J and what not, would be already of interest.

In the Introduction we said: "it shows how to know the computing power of a formal system based on multiset rewriting without running simulations but simply looking at the kind of operations that the formal system can perform". What do we mean with 'looking'?

In this case 'looking' means the way Definition 1 is implemented, that is what it is considered as configuration of the simulated system. Given a configuration of a formal system it is possible to 'look' at it in different ways. For instance, we could ignore some elements in the configuration, we could group different elements in the configuration, etc. This concept of 'looking' (observing) has been formalized (see, for instance, [1]). As a consequence of this, given a formal system, we could implement Definition 1 in different ways, leading to different models of Petri nets and, possibly, to different accepted/generated languages.

Suggestion for research 3 *Study how different implementations of Definition 1 in a given formal system change the class of numbers accepted/generated by it. Is it possible to link these results to the topology as presented in this paper?*

The previous suggestion for research can go beyond the study of a few formal systems 'observed' in different ways. It can include more general studies as the classification of systems that can be 'observed' in only one way, or in an unbounded number of ways, or that can accept/generate the same language independently of the way they are observed, etc. For instance, we could obtain results of the kind: the 'observation' of the systems in row 4 of Table 1 can only accept/generate finite classes of numbers, etc.

If we look at rows 2 and 3 in Table 1 you notice that the difference on the accepted class of numbers is due to the allowed composition. Here one could imagine that if the composition would be something in between what present in these two rows, then the accepted language would also be 'in between'. We try to clarify this point. In row 2 we deal with systems in which *join* and *fork* can only be in sequence as in Fig. 3.a, while in row 3 we deal with systems in which the two building blocks can be composed in any way. What language can be accepted by

P/T systems having a finite number of *join* and *fork* in which only a few are in sequence as in Fig. 3.a?

This line of research could be called "language generated by pseudo-random Petri nets" where 'random' refers to the fact that the Petri net is created composing building blocks in a random way, while 'pseudo' refers to the fact that we impose some limitations to this randomness as, for instance, the fact that a few *join* and *fork* have to be in a specific sequence.

Suggestion for research 4 *Study languages generated by pseudo-random Petri nets.*

This line of research requires the creation of some computer programs (or the use of already available computer programs). Of course, a similar line of research can be pursued for P systems and any other kind of computing device that can be build in a pseudo-random way.

In Section 6 we indicated how the results in Table 1 have been used on P systems with catalysts. We tried to use similar results on other models of P systems (symport/antiport, conformons, etc., see [6]) and all went as we expected. This means that using Definition 1 and the results in Table 1 we obtained results similar, in terms of descriptional complexity, to the ones already known. The exception to this were *spiking P systems*. We did not succeed in defining a simulation (as in Definition 1) that let us re-obtain the known results on this model of P systems with the same descriptional complexity. The results we got needed more features (more compartments, the presence of forgetting rules, etc.) not present in direct proofs.

This fact is particularly intriguing because it could suggest some limits in the approach considered in this note. For this reason we propose:

Suggestion for research 5 *Use the approach considered in this paper on spiking P systems, analyzing advantages and limitations of it.*

Only row 2 in Table 1 considers P/T system in which the underlying net has one kind of limitations in its composition. What about other limitations in the arrangement?

Suggestion for research 6 *Study further the computational power of P/T systems whose relative arrangement of pairs of* fork *and* join *is limited.*

In particular one could focus on the limitations needed to generate semilinear sets.

The relevance of the following suggestion should be straightforward:

Suggestion for research 7 *Are* join *and* fork *the only building blocks that lead to the result in Table 1? Are there other building blocks leading to the same or different results?*

The overall aim of the approach considered in this paper is:

Suggestion for research 8 *Create a full hierarchies of accepting and generating computational processes in terms of sets of building blocks, compositions of elements in these sets and the functions W and K (present in the definition of P/T systems in [13, 5]).*

References

1. M. Cavaliere: Computing by observing: A brief survey. In A. Beckmann, C. Dimitracopoulos, B. Löwe, editors, *Logic and Theory of Algorithms, 4th Conference on Computability in Europe, CiE 2008, Athens, Greece, June 15-20, 2008, Proceedings*, volume 5028 of *Lecture Notes in Computer Science*. Springer, Berlin, 2008.
2. R. Freund, L. Kari, M. Oswald, P. Sosík: Computationally universal P systems without priorities: two catalysts are sufficient. *Theoretical Computer Science*, 330, 2 (2005), 251–266.
3. R. Freund, G. Lojka, M. Oswald, G. Păun, eds.: *Membrane Computing. 6^{th} International Workshop, WMC 2005, Vienna, Austria, July 18-21, 2005, Revised Selected and Invited Papers*, volume 3850 of *Lecture Notes in Computer Science*. Springer, Berlin, 2006.
4. P. Frisco: P systems, Petri nets, and Program machines. In Freund et al. [3], 209–223.
5. 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`.
6. P. Frisco: *Computing with Cells. Advances in Membrane Computing*. Oxford University Press, 2009. to appear.
7. P. Frisco: On languages accepted by P/T systems composed by *join*. Technical report, Heriot-Watt University, 2009. HW-MACS-TR-0064 `http://www.macs.hw.ac.uk:8080/techreps/index.html`.
8. S.A. Greibach: Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7 (1978), 311–324.
9. O.H. Ibarra: On membrane hierarchy in P systems. *Theoretical Computer Science*, 334 (2005), 115–129.
10. O.H. Ibarra, H.-C. Yen: Deterministic catalytic systems are not universal. *Theoretical Computer Science*, 363, 2 (2006), 149–161.
11. R. Milner: *Communication and Concurrency*. Prentice-Hall, Englewood, N.J., 1989.
12. R. Milner: *Communicating and Mobile Systems: the π-calculus*. Cambridge University press, 1999.
13. W. Reisig, G. Rozenberg, eds.: *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer, Berlin, 1998.
14. G. Rozenberg, J. Engelfriet: *Elementary net systems*, pages 12–121. Volume 1491 of Reisig and Rozenberg [13], 1998.
15. G. Rozenberg, A. Salomaa: *Handbook of Formal Languages*. Springer, Berlin, 1996.