CMC 19 – International Conference on Membrane Computing
Dresden, 4 – 7 September 2018

Time and Space Complexity of P Systems ~ And Why They Matter ~

Alberto Leporati

Università degli Studi di Milano - Bicocca
Dip. di Informatica, Sistemistica e Comunicazione (DISCo)
Viale Sarca 336/14 – Milano - Italy

E-mail: alberto.leporati@unimib.it

What are P systems?

- A distributed, parallel model of computation
- Really, a framework that inspired many models of computation:
  - cell-like P systems
  - tissue-like P systems
  - neural-like P systems
  - numerical P systems
  - …
- Synchronous or asynchronous
- Most variants are Turing complete, even with one cell

What can P systems be used for?

- Anything, since they are Turing complete
- But, in particular, anything that requires a distributed, parallel (synchronous or asynchronous) model of computation
- For example, simulation of physical / natural systems
- As Marian Gheorghe says:

  *P systems are an **attractive alternative** to mathematical models, e.g. **ordinary differential equations***

How do we (usually) study P systems?

- Computability issues:
  - is this variant of P systems Turing-complete?
  - what are the computational ingredients needed to reach completeness / universality?

    (frontiers of computability)

- (Computational) complexity issues:
  - is this variant able to solve {NP, PSPACE, LOGSPACE, …}–complete decision problems?
  - and their counting versions?
  - and their optimization versions?

Why is computational complexity useful?

- Because it tells us what we can compute, but especially what we cannot compute
- Example: assume you want to design a Boolean circuit (with AND, OR, NOT gates) that computes the PARITY function:

$$\text{PARITY}(x_1, \ldots, x_n) = x_1 \oplus x_2 \oplus \ldots \oplus x_n$$

  - you want that the circuit has polynomial size (= it is constructible) and constant depth (= answers in constant time)
  - unfortunately, this is not possible [Furst, Saxe, Sipser 1985]: no (uniform family of) polynomial size constant-depth circuit for PARITY exists

Why is computational complexity useful?

- In a sense,

  results about computational complexity, usually referred to as efficiency results,

  become

  results about the computing power of our models

- What changes is the universe of problems / languages / functions considered: P, NP, PSPACE, etc. instead of RE

What can P systems be used for?

- For example, simulation of physical / natural systems
- However, if the computational model is Turing complete:
  - most of dynamic properties are undecidable (see also Rice's theorem)
  - it can simulate anything (it is universal), but perhaps in an indirect way (e.g., it is difficult to program), which is not very useful
  - not even interesting from a theoretical point of view

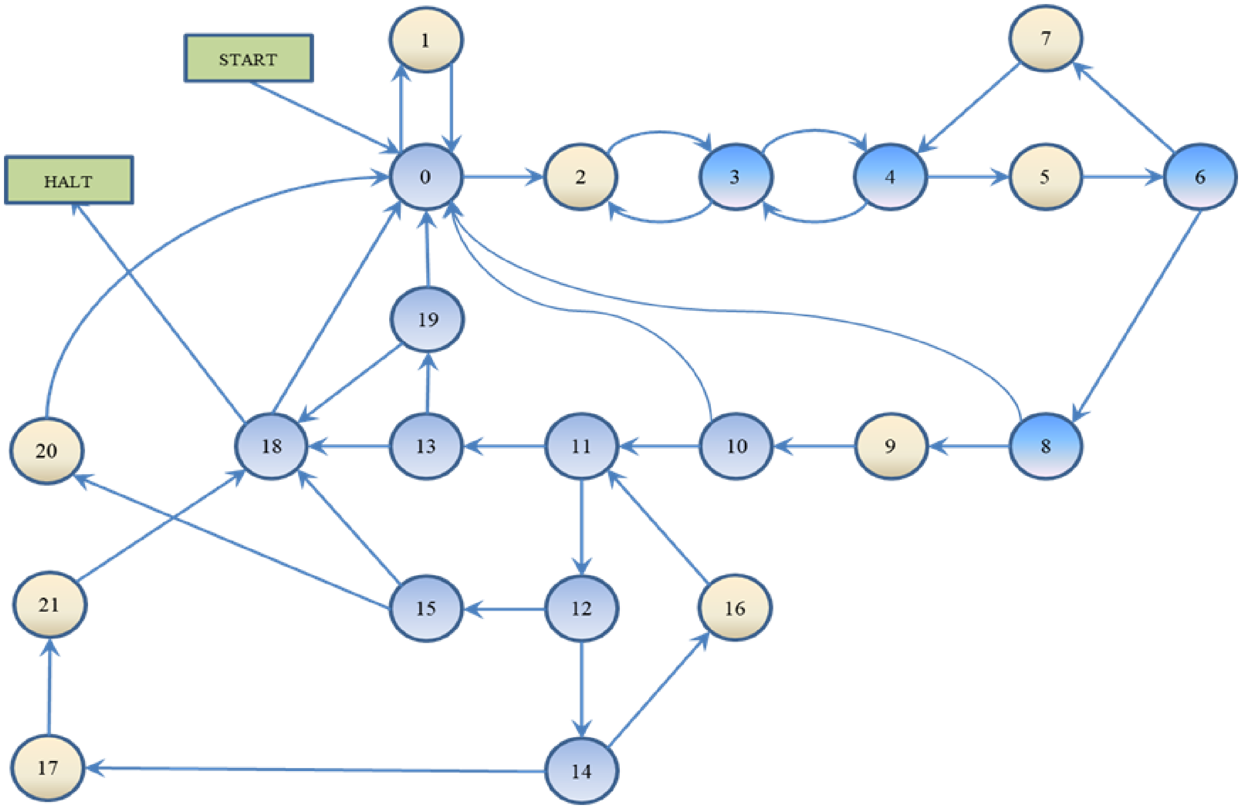
What can P systems be used for?

- The behavior of small universal systems can be complicated to understand. Example: Korec's small universal register machine:

0 : (DEC(1), 1, 2)	1 : (INC (7), 0)
2 : (INC(6), 3)	3 : (DEC (5), 2, 4)
4 : (DEC (6), 5, 3)	5 : (INC (5), 6)
6 : (DEC (7), 7, 8)	7 : (INC (1), 4)
8 : (DEC (6), 9, 0)	9 : (INC (6), 10)
10 : (DEC (4), 0, 11)	11 : (DEC (5), 12, 13)
12 : (DEC (5), 14, 15)	13 : (DEC (2), 18, 19)
14 : (DEC (5), 16, 17)	15 : (DEC (3), 18, 20)
16 : (INC (4), 11)	17 : (INC (2), 21)
18 : (DEC (4), 0, 22)	19 : (DEC (0), 0, 18)
20 : (INC (0), 0)	21 : (INC (3), 18)

What can P systems be used for?

- Another view of Korec's small universal register machine:



What can P systems be used for?

- For applications / simulations, less powerful than Turing machines is better!
  - hopefully, dynamic properties are decidable (but what is their complexity?)
  - but, maybe, they cannot simulate interesting phenomena (power vs. expressivity)
  - in any case, they are more interesting from a formal language point of view

The complexity of dynamic behavior

- Even if a dynamic property is decidable, it may be not accessible in practice
  - example (discussed later): decide whether a neuron in a SN P system – whose rules use arbitrary regular expressions – ***will fire at the next computation step*** (may involve solving an NP-complete problem)
  - other example: reachability in some variants of Petri nets is NP-complete
- Even more, constant time complexity may be inaccessible!
  - example: brute-force attack to AES-128, to find the key, given a plaintext and the corresponding ciphertext

An example

SN P systems:

computational completeness

and time complexity

SN P systems: definition

An SN P system of degree $m \geq 1$ is a construct:

$$\Pi = (O, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, in)$$

where:

- $O = \{a\}$ is the singleton alphabet
- cells (neurons) $\sigma_i = (n_i, R_i)$ are placed in the nodes of the synapse graph *syn*, where:
  - $n_i \geq 0$: initial contents
  - R_i: finite set of rules
- $syn \in \{1,2,\ldots,m\} \times \{1,2,\ldots,m\}$: directed graph of synapses, without self-loops
- $in, out \in \{1,2,\ldots,m\}$: input and output neurons

SN P systems: rules, configurations

The rules can be:

- firing rules: $E/a^c \to a; d$ with $c \geq 1$, $d \geq 0$ integers
  (if $E = a^c$ we simply write $a^c \to a; d$)
- forgetting rules: $a^s \to \lambda$ with $s \geq 1$ and $a^s \notin L(E)$ for any firing rule $E/a^c \to a; d$ in the neuron
- initial configuration:
  - $n_1, n_2, \ldots, n_m$ spikes in the neurons
  - all neurons are open
- configuration (during computations): for each neuron:
  - number of spikes in the neuron
  - number of steps to wait until the neuron becomes open

SN P systems: computing functions

Computation of $f: \mathbb{N} \rightarrow \mathbb{N}$

- the system starts in the initial configuration
- input n enters in *in*, encoded as the time elapsed between two spikes (unary notation)
- computation proceeds sequentially in every neuron, and in parallel between the neurons
  - if ≥ 2 rules can be applied in a neuron, a nondeterministic choice is made
  - the system is deterministic if $L(E_i) \cap L(E_j) = \emptyset$ for all $i \neq j$
- if the system halts, then the output $f(n)$ is read as the time elapsed between the first two spikes emitted by the output neuron (to the environment)

SN P systems: some variants

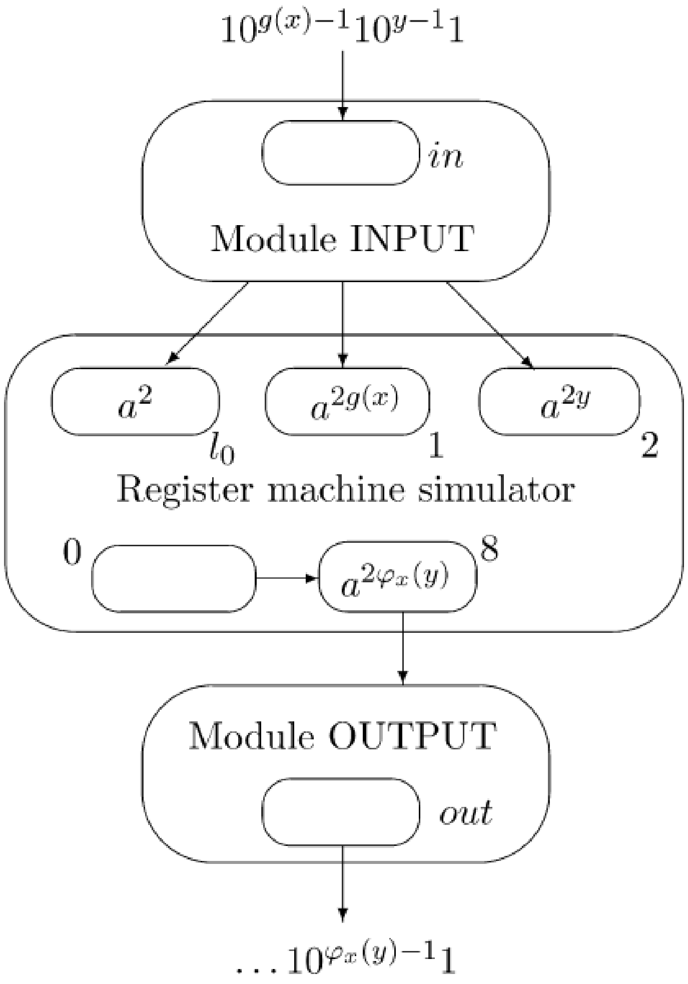
Many variants/possibilities:

- different halting conditions
- different ways to encode input and output values
- different ways to compute f: $\mathbb{N}^{\alpha} \rightarrow \mathbb{N}^{\beta}$
  - directly (spike trains in input and output)
  - through bijections ($\mathbb{N}^{\alpha} \leftrightarrow \mathbb{N}$ and $\mathbb{N}^{\beta} \leftrightarrow \mathbb{N}$)
- generative case: we ignore the input
- accepting case: we ignore the output

- SN P systems are universal
- Several normal forms

SN P systems: completeness

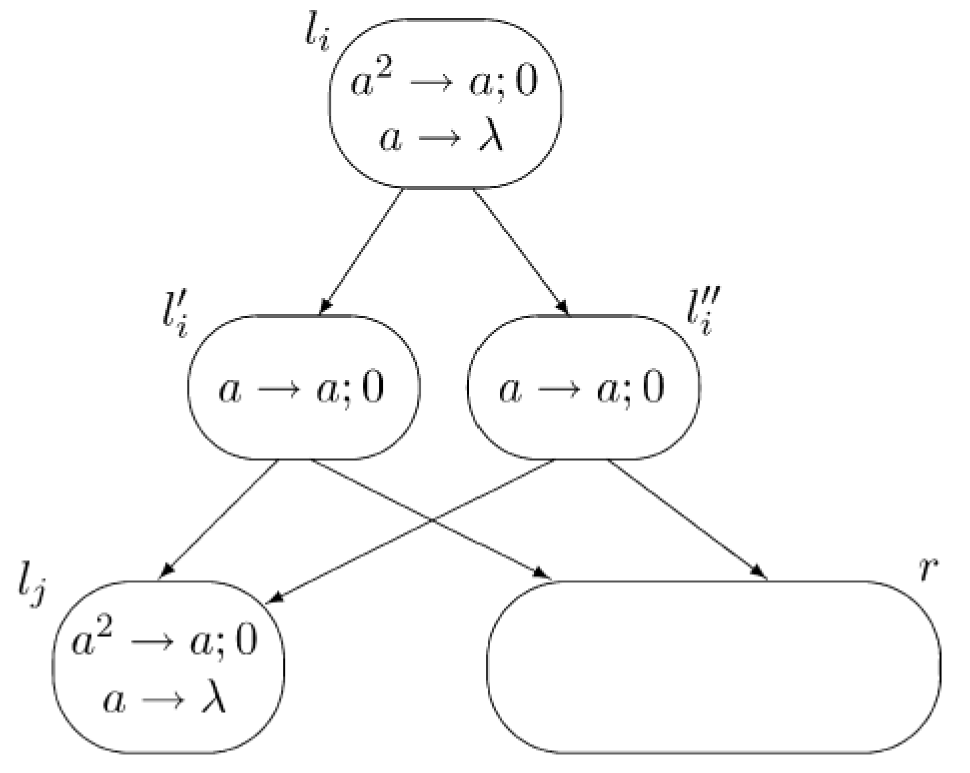
- Idea: simulating Korec's small universal register machine
- Input and output must be formatted in an appropriate way
- General design of the universal SN P system:



SN P systems: completeness

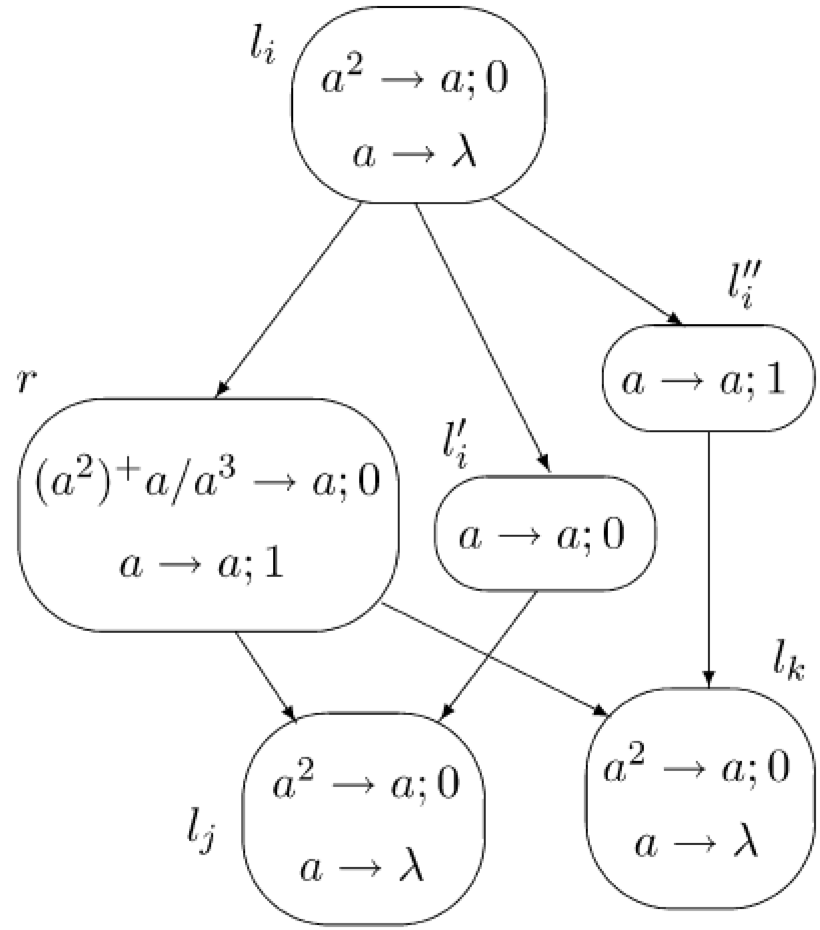
- module *ADD*, simulating *i*: INC(*r*), *j*

- Note: if register *r* contains *n*, then the corresponding neuron contains 2*n* spikes



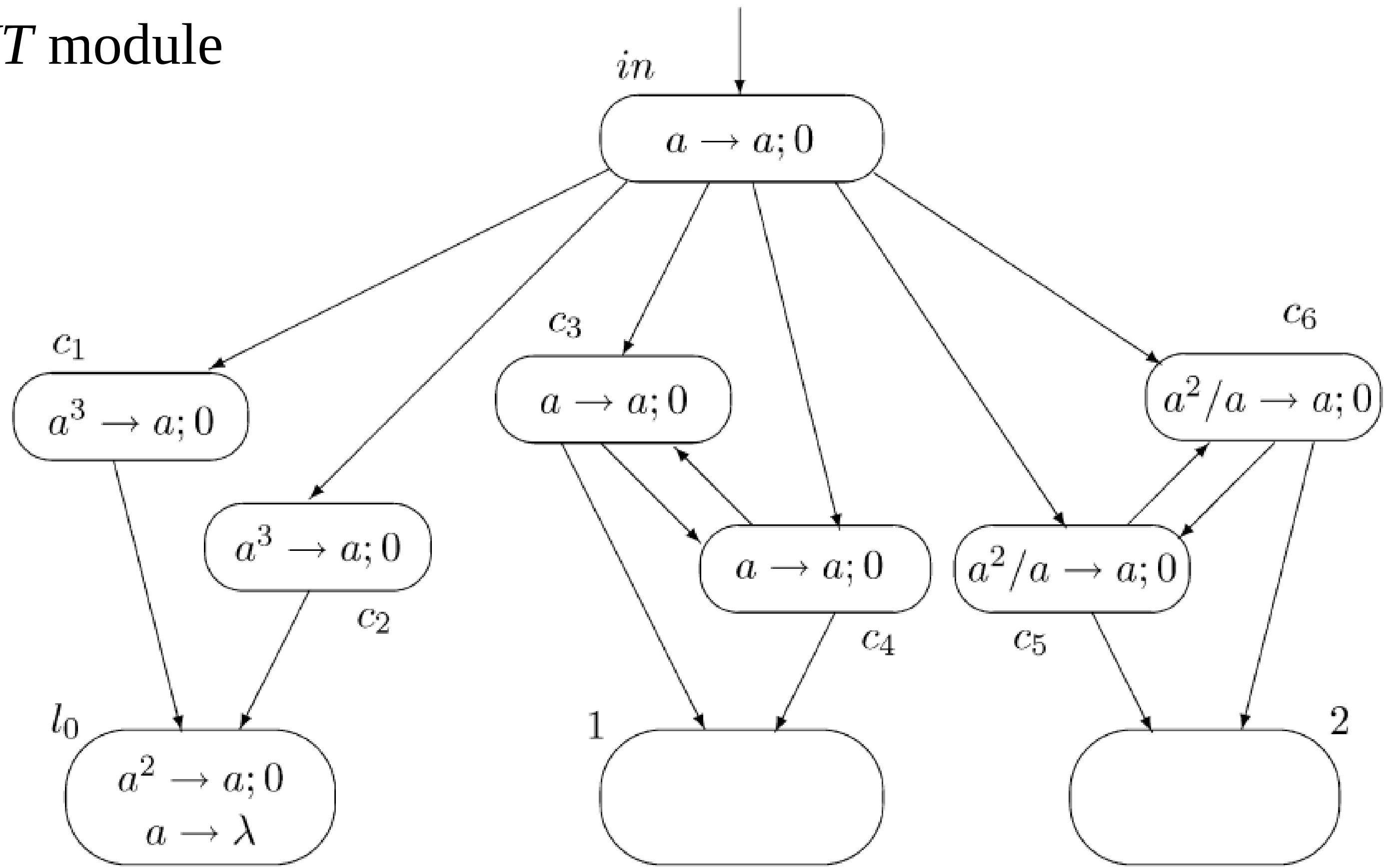
SN P systems: completeness

- module *SUB*,
  simulating *i:* DEC(*r*), *j*, *k*



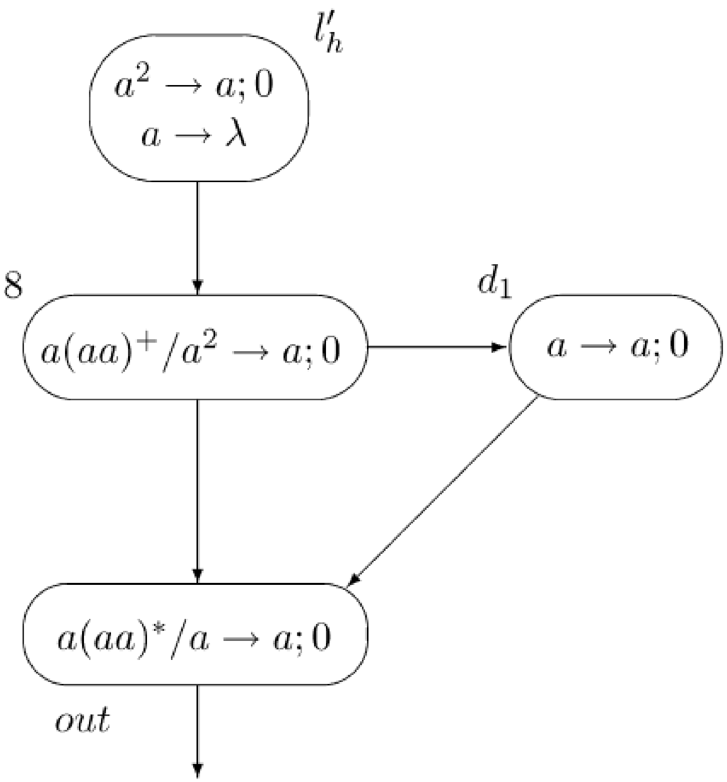
SN P systems: completeness

- the *INPUT* module



SN P systems: completeness

- the *OUTPUT* module



Size of SN P systems

Description size of Π: number of bits required to represent it:

- no bits for the alphabet
- $\leq m^2$ bits for the synapse graph
- *in* and *out*: $\lg m$ bits each
- for every neuron σ_i :
  - $n_i \leq N \longrightarrow \lg N$ bits
  - At most R rules; for each rule $E/a^c \rightarrow a; d$

    $type \in \{firing, accepting\} \longrightarrow 1$ bit

    regular expression $E \longrightarrow$ size $\leq S$ bits

    two numbers $\longrightarrow 2 \lg N$ bits
- Total size: $m^2 + 2 \lg m + m (\lg N + R (1 + S + 2 \lg N))$ bits

Deterministic SN P systems

- explicit simulation by DTMs given in [IJUC 2009]
  - t steps of any deterministic SN P system Π can be simulated in *poly*(t, *description size*) steps of a DTM
- crucial assumption: regular expressions are of very restricted form, for example:
  - a^i, with $i \leq 3$
  - $a(aa)^+$

  (the membership problem must be polynomial also in the succint version)
- hence, to solve NP-complete problems we either need:
  - nondeterminism (trivial), or complicated reg. expr., or
  - some way to trade-off space for time (division, budding,…)

The Subset Sum problem

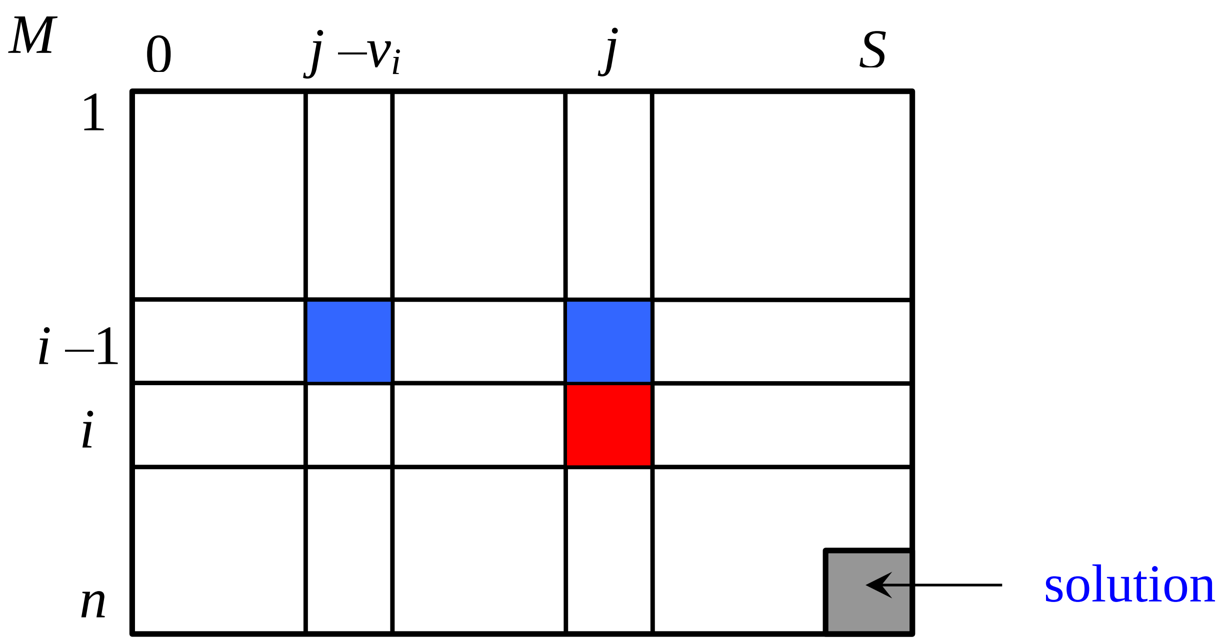
- instance:
  - a (multi)set $V = \{v_1, v_2, \ldots, v_n\}$ of positive integer values
  - a positive integer value S
- question: is there a sub(multi)set $A \subseteq V$ such that

$$\sum_{a \in A} a = S \ ?$$

- numerical, pseudo-polynomial NP-complete problem
- instance size:
  - let $K = max\{v_1, v_2, \ldots, v_n, S\}$
  - each number requires lg K bits to be represented
  - total size: $\Theta(n \lg K)$

Dynamic programming solution

- boolean matrix $M[1..n, 0..S]$
- $M[i, j] = 1 \Leftrightarrow$ there exists $B \subseteq \{v_1, \ldots, v_i\}$ such that $\sum_{b \in B} b = j$
- space and time complexity: $\Theta(nS) = \Theta(nK)$



Dynamic programming solution

- pseudo-code for filling the matrix:

$\underline{\textsc{Subset Sum}}(\{v_1, v_2, \ldots, v_n\}, S)$

$\texttt{for } j \leftarrow 0 \texttt{ to } S$
$\quad \texttt{do } M[1,j] \leftarrow 0$
$M[1,0] \leftarrow M[1,v_1] \leftarrow 1$
$\texttt{for } i \leftarrow 2 \texttt{ to } n$
$\quad \texttt{do for } j \leftarrow 0 \texttt{ to } S$
$\qquad\quad \texttt{do } M[i,j] \leftarrow M[i-1,j]$
$\qquad\qquad \texttt{if } j \geq v_i \texttt{ and } M[i-1, j-v_i] > M[i,j]$
$\qquad\qquad\quad \texttt{then } M[i,j] \leftarrow M[i-1, j-v_i]$
$\texttt{return } M[n,S]$

Unary regular languages

Remark: in SN P systems we work with unary languages

- every string is bijectively associated with its length
- a compact representation of E works on the lengths
- regular expressions on natural numbers
  - union and Kleene star are computed as usual
  - $L_1 \bullet L_2$: all numbers of L_1 are summed with those of L_2 in all possible ways

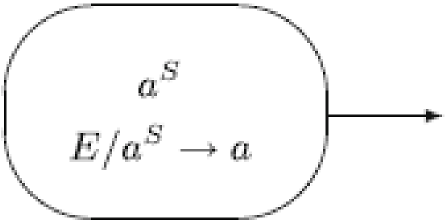
Example: $\{2,3\} \bullet \{2,5\} = \{4,5,7,8\}$

Solving Subset Sum with one rule !

- Let $(\{v_1, v_2, \ldots, v_n\}, S)$ be an instance of Subset Sum
- Consider the languages (in succint form):

$$L_i = \{0, v_i\}, \quad \text{for } i \in \{1, 2, \ldots, n\}$$

- Let $L = L_1 \bullet L_2 \bullet \ldots \bullet L_n$
- Membership problem: is $S \in L$?
  - the answer is **yes** if and only if the instance of Subset Sum is positive
- Deciding whether the rule $E/a^S \to a; d$ can be applied, when $L = L(E)$ and the neuron contains S spikes, can be difficult

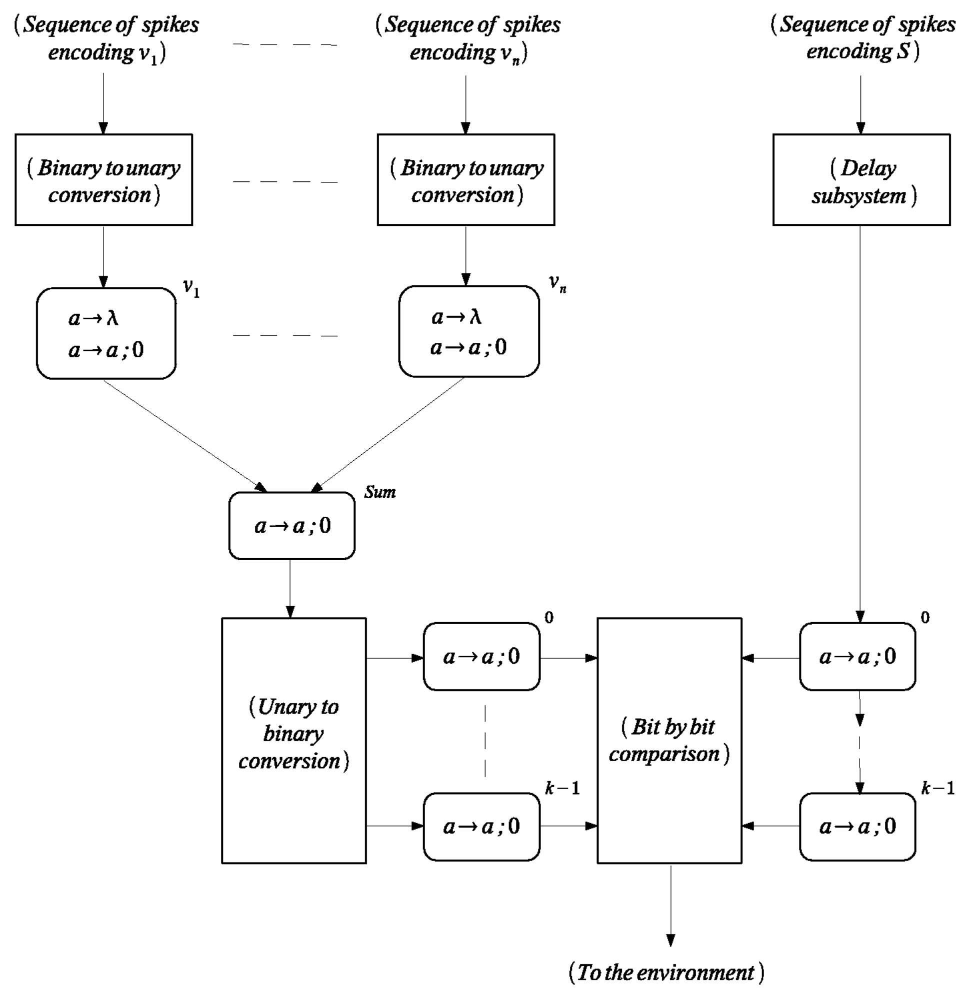


Solving Subset Sum with SN P systems

- In the same paper, we also provided
  - a semi-uniform solution of Subset Sum by (extended) SN P systems
  - a uniform version
- But with extended rules, and the numbers are provided simultaneously as inputs in binary form
  - they are converted from binary to unary
- The output is observed after a given number of steps
- Nondeterminism is kept at minimum

- In practice, a (nondeterministic) circuit made of neurons!

Solving Subset Sum (uniform solution)

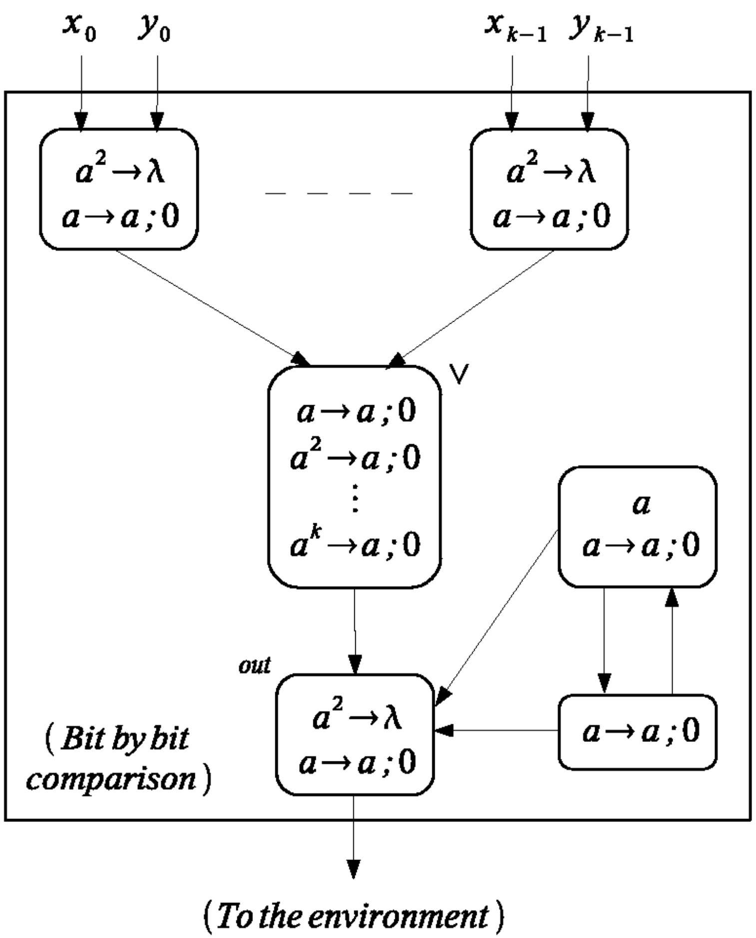


- the output has to be observed exactly after 3k+6 steps
- length of delay subsystem: 3k+2 steps

The Comparison subsystem

- emits one spike if and only if the two numbers given in input (expressed in binary form) are equal
- the subsystem computes the following boolean function:

$$\textsc{Compare}(x_0, \ldots, x_{k-1}, y_0, \ldots, y_{k-1})$$
$$= \bigwedge_{i=0}^{k-1} (\neg(x_i \oplus y_i)) = \neg \left(\bigvee_{i=0}^{k-1} (x_i \oplus y_i) \right)$$



Space complexity results

- Space complexity = #objects + #membranes
- The model: P systems with active membranes
- PSPACE = decision problems solved by polynomial space P systems
- The same for EXPSPACE and higher classes

But, curiously,

- PSPACE = decision problems solved by logarithmic (or even constant) space P systems

---

- Another possible approach: IP = PSPACE (dP systems?)

Space complexity results

- A toolbox for designing P systems with active membranes
  - polynomial number of charges
  - several other simulations
- Use of oracles introduces a hierarchy based on the nesting depth of the membrane structure
  - linear (and, now, $\frac{n}{\log n}$) depth: PSPACE
  - depth 1: $\mathrm{P}^{\#\mathrm{P}}$
  - in the middle: *hic sunt leones*!

The factorization problem

- Given $n = pq$, where p and q are prime numbers, it is difficult to compute p (or q)
- Let $m = \log_2 n$, then trying to divide by all numbers between 2 and $\sqrt{n}$ takes an exponential time:

$$O(\sqrt{n}) = O(\sqrt{2^m}) = O(2^{m/2})$$

- Nobody knows whether a polynomial time algorithm exists

- We have seen a brute-force parallel attack, by P systems
- Is there a « better » parallel algorithm?

The factorization problem

- Example of instance:

RSA-768	=	1230186684530117755130494958384962720772853569595 9533479219732245215172640050726365751874520219 9786469389956474942774063845925192557326303453731 54826850791702612214291346167042921431160222124 04792747377940806653514195974598 5 6902143413

The factorization problem

- Example of instance:

RSA-768 = 12301866845301177551304949583849627207728535695
95334792197322452151726400507263657518745202199
78646938995647494277406384592519255732630345373
15482685079170261221429134616704292143116022212
40479274737794080665351419597459856902143413

= 33478071698956898786044169848212690817704794983
71376856891243138898288379387800228761471165253
1743087737814467999489

$\times$ 36746043666799590428244633799627952632279158164
34308764267603228381573966651127923337341714339
6810270092798736308917

The factorization problem

- Consider $\phi(n) = (p-1)(q-1)$ (Euler's totient function)
- In general, $\phi(n) = |x \in \mathbb{N} : 1 < x \leq n \text{ and } \mathrm{GCD}(x,n) = 1|$
- If we know the factorization of n then computing $\phi(n)$ is easy, otherwise it is difficult
  - we would break the cryptosystem RSA
  - we could factorize n:

$$\phi(n) = (p-1)(q-1) = pq - (p+q) + 1$$

from where:

$$\begin{cases} pq = n \\ p + q = n - \phi(n) + 1 \end{cases}$$

p and q are the solutions of $x^2 - (p+q)x + pq = 0$

The factorization problem

- So, computing $\phi(n)$ has the same difficulty as factorizing n
- Question: do we know a parallel algorithm to compute

$$\phi(n) = |x \in \mathbb{N} : 1 < x \leq n \ \text{ and } \ \mathrm{GCD}(x,n) = 1| \ ?$$

- Answer: no, and the bad news are that GCD seems to be not parallelizable!

Computational power vs Expressivity

- Usually, Turing-completeness is proved by simulating Turing machines or register machines
- Example: simulation of register machines by SN P systems
  - assume simulation of a deterministic register machine, computing functions $\mathbb{N} \to \mathbb{N}$
- This means that SN P systems can be used to compute any computable function
- Exercise: design a SN P system that, given $n \in \mathbb{N}$, computes and outputs n^2
  - is the design process simple?
  - is it handy?

A "high-level" programming language for building P systems

- Idea:
  - we may first write a program for a register machine, and then build the SN P system by composing ADD and SUB modules
    - this substitution can be performed *automatically*
    - it works for many universal models of P systems
- The difficulty in writing the program may depend upon the function to be computed, hence we could:
  - write a program for a "high-level" programming language, which is then easily compiled to an equivalent program for a register machine
  - build the P system by composing ADD and SUB modules

A “high-level” programming language for building P systems

- Proposal: make both translations automatically, that is:

- The first compiler would be fixed, the others would depend upon the model of P systems considered
- The output could be given in P-Lingua
- To start with, the high-level language should be very easy
  - a possible candidate: the WHILE language
  - of course, the WHILE language is Turing-complete

A “high-level” programming language for building P systems

- The WHILE language:
  - Variables x_j, for $j \in \mathbb{N}$, each containing a non-negative integer value
  - Assignment commands:

    $x_k := 0 \qquad x_k := x_j + 1 \qquad x_k := x_j \dot{-} 1 \qquad$ (truncated decrement)
  - While commands:

    while $x_k \neq 0$ do C

    where C is an arbitrary command
  - Compound commands:

    begin $C_1;\ C_2;\ \ldots\ C_m;$ end $\qquad (m > 0)$

    where $C_1;\ C_2;\ \ldots\ C_m$ are arbitrary commands
  - A program is a compound command

A "high-level" programming language for building P systems

- The WHILE language can be extended through macros of the kind

$$x_i = Op(x_j, x_k)$$

  For example, *Op* can be *Sum*, *Product*, *TruncatedSum*, *IntegerDivision*, *Mod*, *CantorPairingFunction*, …

- Other natural extensions/alternatives:
  - using a more sophisticated/expressive language
    - programs would be easier to write, but on the other hand
    - the compiler would be harder to write
  - compiling to more sophisticated/expressive low-level languages (RAM machines, appropriate assembly languages, …)

A “high-level” programming language for building P systems

- However, in this way P systems are used in the sequential way
  - what about a concurrent programming language?
    - Inspired from Occam?
  - what about distributed (and concurrent), possibly asynchronous, languages?
    - SCOOP? Message-passing, it allows the creation of « contracts »

[C. Corrodi, A. Heußner, C.M. Poskitt: A Semantics Comparison Workbench for a Concurrent, Asynchronous, Distributed Programming Language. arxiv:1710.03928, October 2017]

P systems and Cryptography

A topic deeply related with computational complexity: Cryptography

- Some (provocative?) ideas:
  - use of P systems to implement cryptographic operations (encryption, PRNGs, …)
  - even more: cryptographic protocols
  - even more: DApps (Decentralized Apps): see Ethereum
  - even more: computations on encrypted data. It has been done for Boolean circuits and for Turing machines

Killer applications for P systems

We also need parallel, distributed, interesting problems

- Is a problem parallelizable?
  - try to design a Boolean circuit; in which complexity class is the problem?
  - what kind of circuit? (Recall the PARITY example)

    For example, threshold circuits seem to be related with monodirectional P systems with active membranes
- However, a killer app would probably be a decentralized app (DApp)
  - unfortunately, I do not know any really interesting candidate problem / algorithm / protocol. Maybe some form of consensus protocol?

Last slide…

Thanks for
your attention !

Danke für Ihre Aufmerksamkeit!