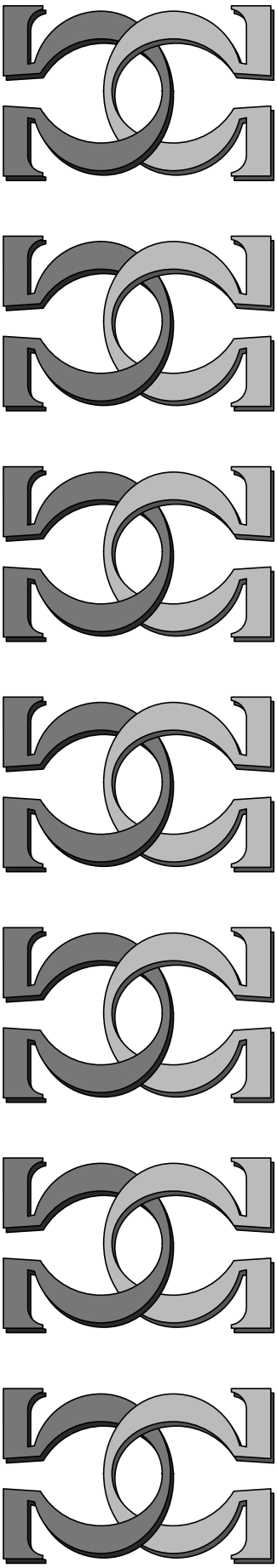

CDMTCS
Research
Report
Series

Implementing Bead–Sort with P systems

Joshua J. Arulanandham
Department of Computer Science
University of Auckland
Auckland, New Zealand

CDMTCS-186
May 2002

Centre for Discrete Mathematics and
Theoretical Computer Science

Implementing Bead–Sort with P systems

Joshua J. Arulanandham

Abstract

In this paper, we implement *Bead–Sort*, a natural sorting algorithm we introduced in [1], with the new, biochemically inspired *P systems*. We make use of a special type of P system — a *tissue P system* that computes by means of communication (using *symport/antiport* rules) only.

1 *Bead–Sort* Algorithm

Bead–Sort is a natural sorting algorithm for positive integers. See [1] where we introduced the new sorting algorithm *Bead–Sort* along with a proof of correctness, analyzed its complexity and discussed different possible implementations in detail. Here, we implement Bead–Sort using the new, biochemically inspired *P systems*. See [2] (where Păun first introduced P systems) and [3] for a detailed discussion on P systems. A brief description of *Bead–Sort* algorithm follows.

We represent positive integers by a set of *beads* (like those used in an *Abacus*) as illustrated below in Figure 1.

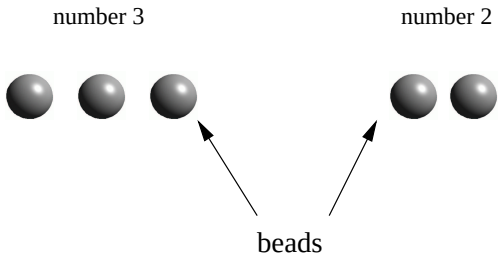

Figure 1

Beads slide through *rods* as shown in Figure 2.

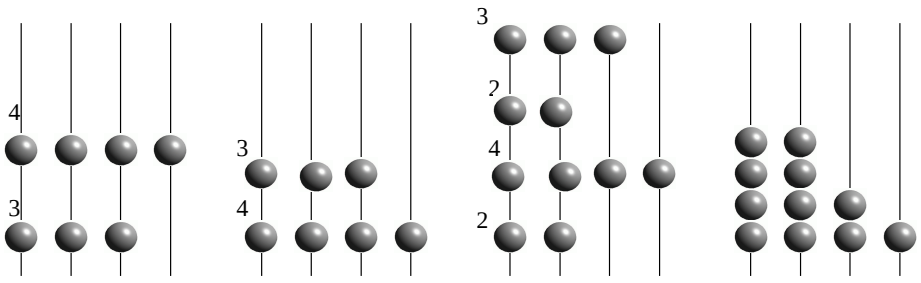

Figure 2

Fig. 2 (a) shows the numbers 4 and 3 (represented by beads) attached to rods; beads displayed in Fig. 2 (a) appear to be suspended in the air, just before they start sliding down. Fig. 2 (b) shows the state of the *frame* (a *frame* is a structure with the rods and beads) after the beads are 'allowed' to slide down. The row of beads representing number 3 has 'emerged' on top of the number 4 (the 'extra' bead in number 4 has dropped down one 'level'). Fig. 2 (c) shows numbers of different sizes, suspended one over the other (in a random order). We allow beads (representing numbers 3, 2, 4 and 2) to slide down to obtain the same set of numbers, but in a sorted order again (see Fig. 2 (d)). In this process, the smaller numbers emerge above the larger ones and this creates a natural *comparison* (an online animation of the above process can be seen at [4]).

Rods (vertical lines) are counted always from left to right and *levels* are counted from bottom to top as shown in Fig. 3. A *frame* is a structure consisting of *rods* and *beads*.

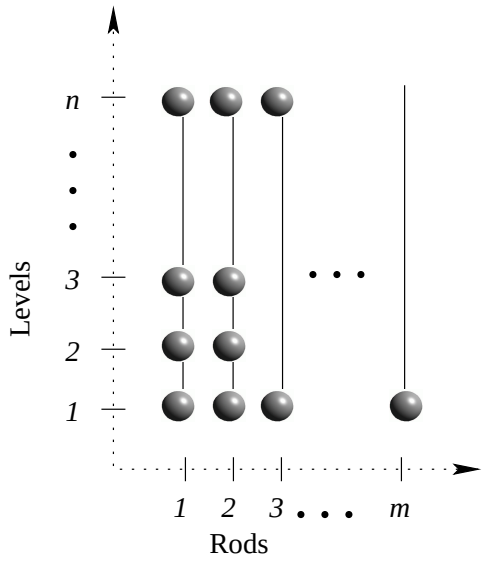

Figure 3

Consider a set A of n positive integers to be sorted and assume the biggest number in A is m. Then, the frame should have at least m rods and n levels. The Bead–Sort algorithm is the following:

The Bead–Sort Algorithm

`For all` $a \in A$ `drop` a `beads (one bead per rod) along the rods, starting from the` 1^{st} `rod to the` a^{th} `rod. Finally, the beads, seen level by level, from the` n^{th} `level to the first level, represent` A `in ascending order.`

The algorithm's run–time complexity ranges from $O(1)$ to $O(S)$ (S is the sum of the input integers) depending on the user's perspective.

2 Objects = *Beads*, Membranes = *Rods*

A *tissue P system with symport/antiport* is used to implement Bead-Sort. (See Figure 4.) Beads are represented by objects x placed within the membranes; a rod is represented

by a 'group' of membranes that can communicate with one another by means of symport/antiport rules. Note that the object '$-$' represents "absence of bead". Thus, the objects x and '$-$' together will reflect bead–positions in the initial state of the frame.

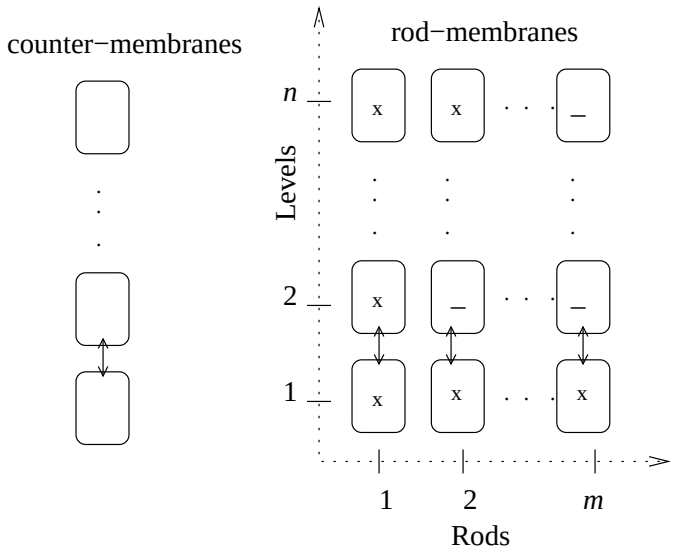

Figure 4

Morover, the "flow" of objects between the group of membranes representing a rod (using communication rules) will reflect the actual flow of beads in the physical system. The 'counter membranes' (see Figure 4) along with the object p will serve the purpose of generating 'clock pulses'; they are useful especially to synchronize, while ejecting the output in the desired sequence. We distinguish these from the other set of membranes — the 'rod membranes'.

More formally, we have a *tissue P system with symport/antiport* of degree $m \times n + n$ ($m \times n$ membranes to represent m rods with n levels; extra n membranes to serve as counters). Note that m rods with n levels can sort n positive integers, the biggest among them being m.

In the following sections, we discuss the various symport/antiport rules used to simulate Bead–Sort. A simple P system that can sort 3 integers (biggest among them is 3) is used for illustrative purpose. To explain the necessity of each rule, we initially start with a system having only the 'basic rule', then gradually add more and more complex rules till we arrive at the complete system. We formally define the system only at the final stage.

3 Making bead–objects "fall down"

As outlined in the previous section, one can set–up the initial state of the frame using a tissue P system. Now, the objects x (representing the beads) should be made to fall down/flow like the real beads in the physical system. This is achieved through a simple antiport rule. See Figure 5 which demonstrates the rule with a simple tissue P system representing 3 rods with 3 levels; we start with the unsorted set $\{2, 1, 3\}$. The antiport rule initiates an exchange of objects x and '$-$'; this "simulates" the action of beads falling down. Figure 5(c) shows the sorted state of the frame; note that the

antiport rule can no longer be applied. Also, one can see that not more than $(n-1)$ steps will be consumed to reach the sorted state using the application of this rule.

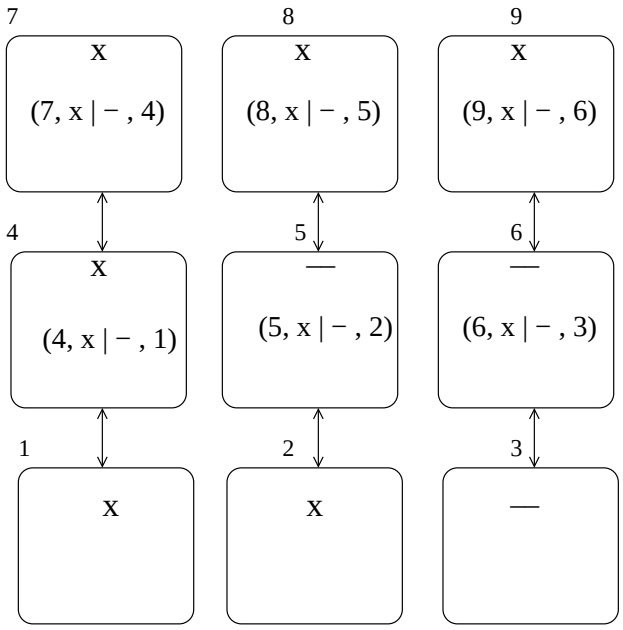

Figure 5(a)

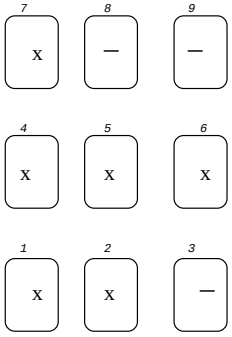

Figure 5(b)

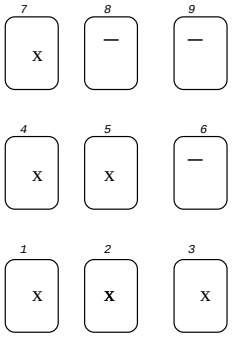

Figure 5(c)

Observe that the multiplicity of the object x in the membranes comprising the same *level* taken together (e.g. membranes 1,2 and 3 in Figure 5 comprise *level–1*) denotes an integer. And, these integers are now in the sorted order, viewed level–wise, as seen in Figure 5(c).

Now, we discuss the rules for "reading" the output in the proper sequence, in the following section.

4 Reading the output

The output can be read in the proper sequence using only symport/antiport rules. But, the rules are a little more complex. The idea is to first "know" when the P system has reached the sorted state; then, the objects from the rod–membranes can be ejected into the environment, starting from *level–1* to *level–n*. The objects from membranes comprising (denoting) the same *level* will be ejected simultaneously as a 'single string'

to be externally interpreted as representing a distinct integer. Thus the ejected output will form the following language:

{*String of objects from level-1 membranes*, *String of objects from level-2 membranes*, ...}

Note that, externally, the multiplicity of x is "calculated" separately for each string in the language which are ejected at different points of time.

Now, let us include new rules that accomplish these actions related to reading the output. Figure 6 shows the inclusion of new symport rules in the rod–membranes. Observe the inclusion of new objects $c1$, $c2$, $c3$ in the rules. These rules would eject (symport) the x objects from the membranes into the environment along with the "prompting" objects $c1$, $c2$, $c3$, but only if they ($c1$, $c2$, $c3$) are present. It is clear that, one has to first send $c1$, $c2$, $c3$ into the rod–membranes in order to "prompt" the ejection of the output. In what follows, we discuss the rules that do the "prompting".

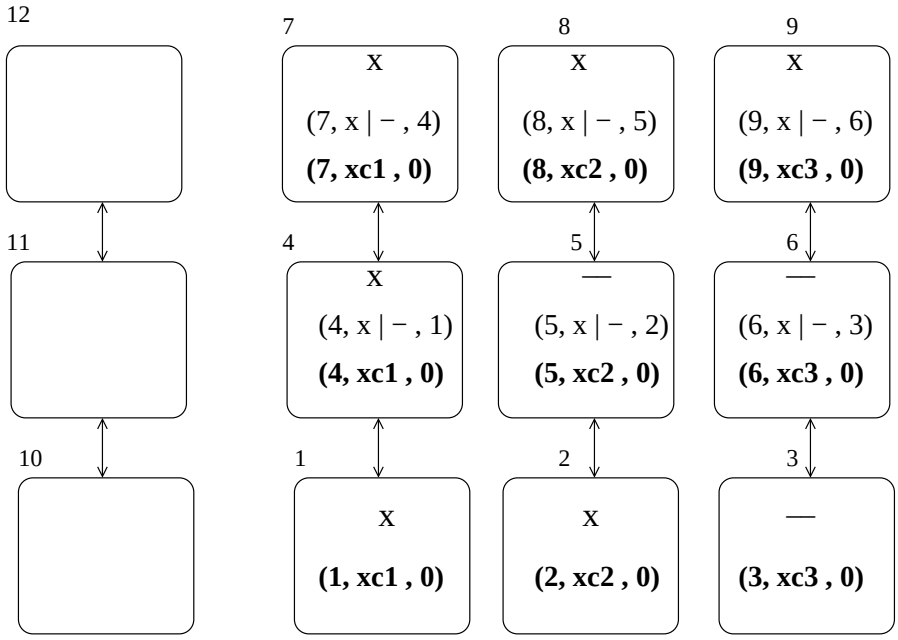

Figure 6

Remember, before prompting the ejection of output, the system has to first "know" (and ensure) that it has reached the sorted state. The new rules to be discussed help ensure this. Figure 7 shows new rules added to the counter–membranes. Note the presence of object p within membrane 12. These new symport rules would "move" p from one counter membrane to the other, driven by the global clock. After $n-1$ time units of the global clock (2 units in our example), p would have been symported to the counter–membrane denoting *level–1* (membrane 10). Recall from previous section, not more than $(n-1)$ steps will be consumed for the bead–objects to "fall down" and reach the sorted state.

c1, c2, c3 .... (available in arbitrary numbers outside)

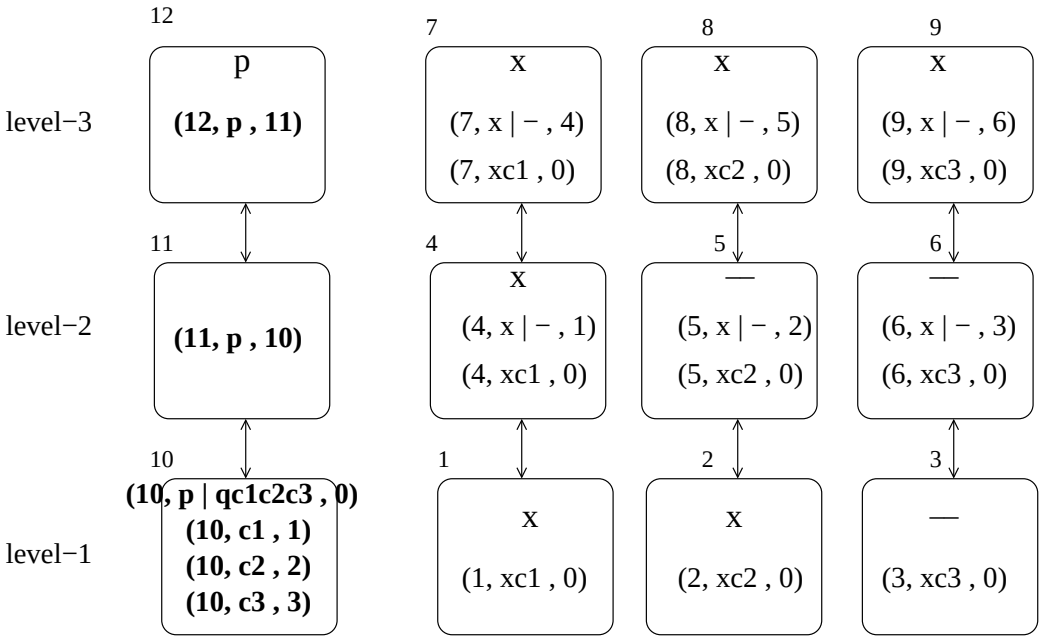

Figure 7

q, r ....
c1, c2, c3 .... (available in arbitrary numbers outside)

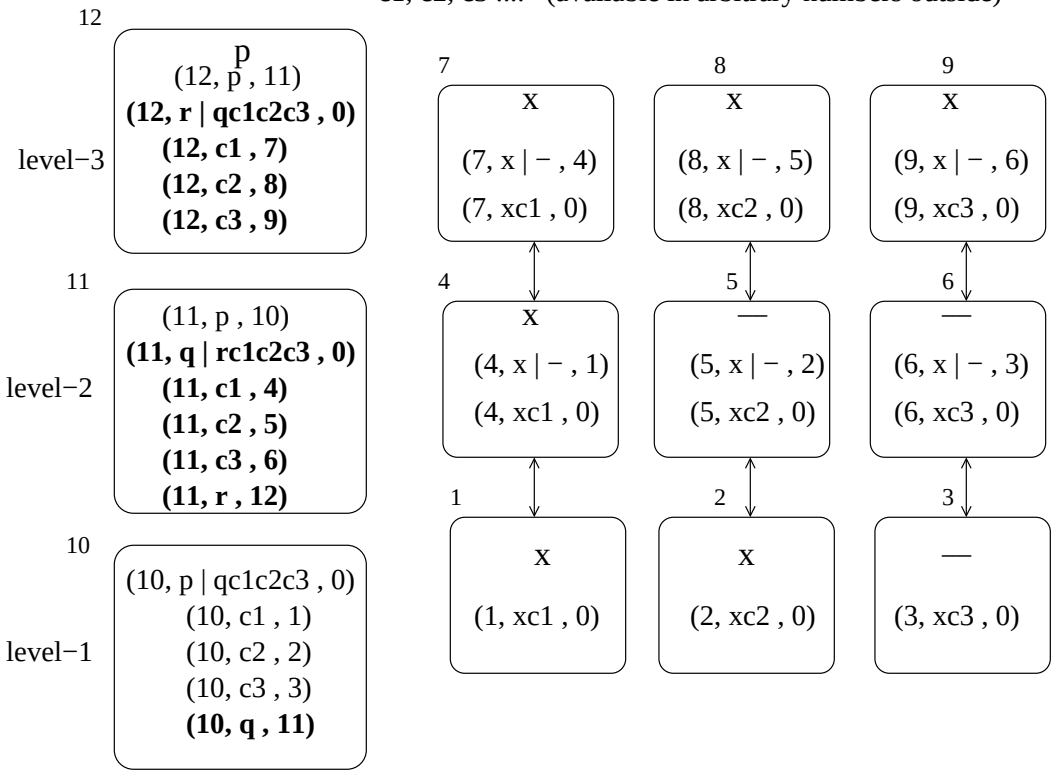

Figure 8

Thus, the presence of p within *level–1* counter–membrane (membrane 10) would ensure that the bead–objects have settled down in a sorted state.[1] Note that the other rules written in membrane 10 (*level–1* counter–membrane) would start prompting ejection of the output, after p is availabe in membrane 10. The rules get $c1$, $c2$, $c3$ (and another object q) from the environment and transfer them into the group of membranes representing *level–1* (1,2 and 3 in our case) thus prompting the ejection of x objects as output. (Assume the presence of arbitrary number of $c1$, $c2$, $c3$ in the environment. The need for object q will be discussed shortly.)

Still we need to add more rules which prompt the ejection of (output) strings from *level–2* upto *level–n*. The idea is to move two new objects q and r (alternately) through the counter–membranes, now from *level–2* upto *level–n* ("upward")[2]. The presence of objects q and r in the counter–membranes would trigger certain new rules that would subsequently prompt output from *level–2* upto *level–n* rod–membranes, one by one. (Note, in our case, $n = 3$.) These rules have been included in Figure 8. (Assume the presence of arbitrary number of q's and r's in the environment.)

5 A Sample Illustration

We first formally define a *tissue P system* of degree 12 (3 rods $\times$ 3 levels + 3 counter membranes) with *symport/antiport* rules which can sort a set of three positive integers, the maximum among them being three:

$\Pi = (V, T, \omega 1, \omega 2, ..., \omega 12, M_0, R)$
where:
(i) V (the alphabet) = $\{x, -, p, q, r, c1, c2, c3\}$
(ii) T (the output alphabet) $\subset V$
T = $\{x, c1, c2, c3\}$ (the multiplicity of $c1$, $c2$, $c3$ is to be ignored, finally)
(iii) $\omega 1$, ...,$\omega 12$ are strings representing the multisets of objects initially present in the regions of the systems. These are shown in Figure 9(i).
(iv) $M_0(x) = M_0(-) = M_0(p) = 0$;
$M_0(q) = M_0(r) = M_0(c1) = M_0(c2) = M_0(c3) = \infty$
(v) R is a finite set of (symport/antiport) rules.
They are enumerated in Figure 8.
Note that we do not use a separate 'output membrane', but prefer to "eject" the output into the environment.

Figure 9 illustrates sorting {2, 1, 3} with snap–shots of all the intermediate configurations, clearly showing the evolution of the system until output is "read".

[1] Note, we are forced to adopt this method because, there seems to be no better way to ensure whether the system has reached the final state or not;for instance, we can not deduce this from the 'states' of individual membranes.

[2] We can not use p again as it has already been used by earlier rules; p would activate the same set of actions as before, which is undesirable.

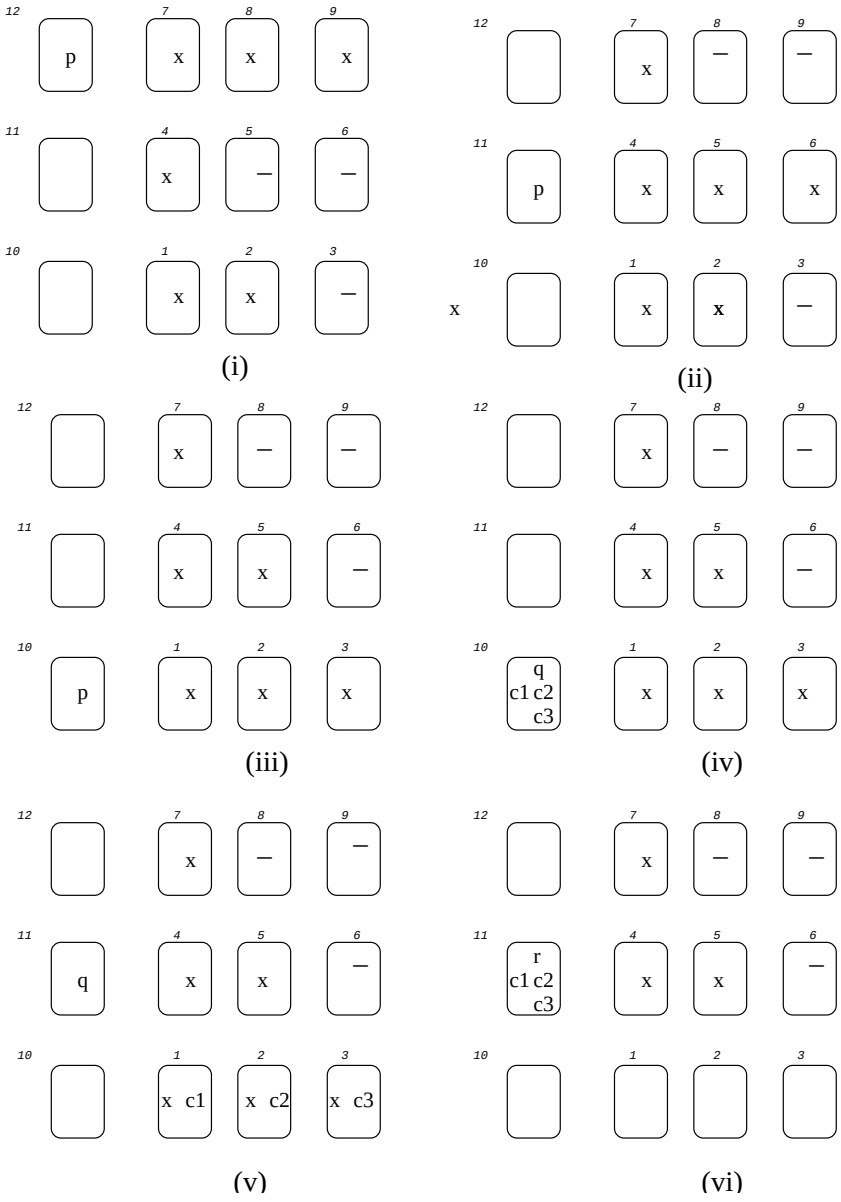
12 7 8 9
p x x x
11 4 5 6
x –
10 1 2 3
x x –
(i)
12 7 8 9
x – –
11 4 5 6
p x x x
x
10 1 2 3
x x –
(ii)
12 7 8 9
x – –
11 4 5 6
x x –
10 1 2 3
p x x x
(iii)
12 7 8 9
x – –
11 4 5 6
x x –
10 1 2 3
q c1 c2 c3 x x x
(iv)
12 7 8 9
x – –
11 4 5 6
q x x –
10 1 2 3
x c1 x c2 x c3
(v)
12 7 8 9
x – –
11 4 5 6
r c1 c2 c3 x x –
10 1 2 3
(vi)

Figure 9

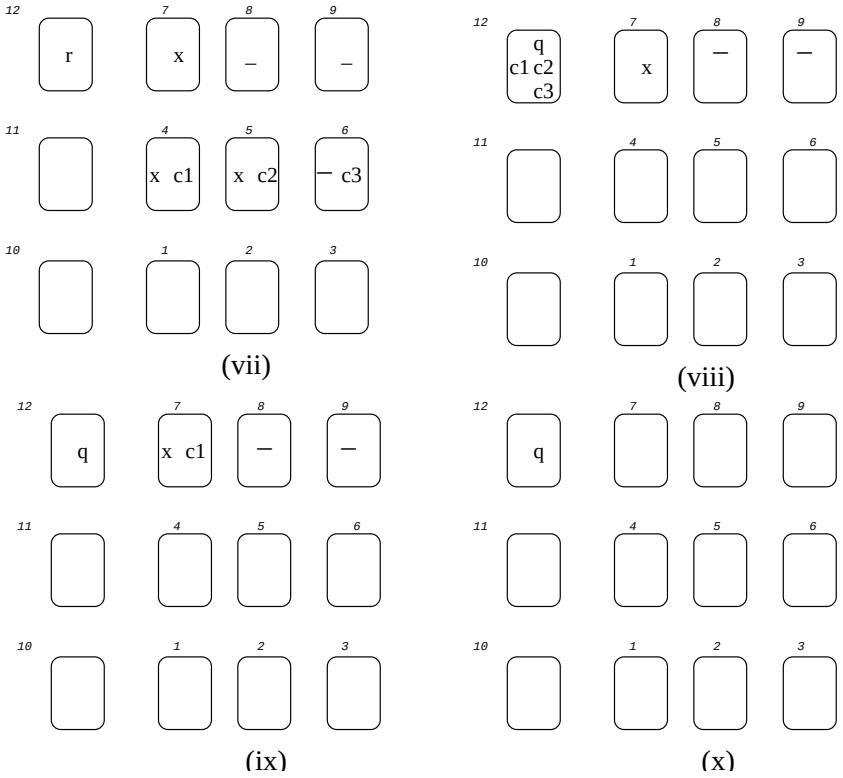

Figure 9 (*cont'd.*)

Note that output strings from membranes denoting levels 1, 2 and 3 would be ejected during stages(vi), (viii) and (x) (in Figure 9) respectively. One has to ignore the $c1$, $c2$, $c3$s that accompany x objects and take into account only the multiplicity of x in each output string.

6 Conclusion

Bead–Sort, a natural algorithm for sorting positive integers has been implemented with a *tissue P system* that uses only *symport/antiport* rules. The complexity of the algorithm is $O(n)$. As the system uses only communication rules (symport/antiport), the procedure to read the output has been a bit tedious. The P system built for sorting three numbers can be generalized for sorting any n positive integers by simply adding more membranes, without any change in the number of rules employed.

Acknowledgement

We thank Dr. G. Păun for comments and suggestions which improved the paper.

References

[1] J. J. Arulanandham, C. S. Calude, M. J. Dinneen. Bead–Sort: A natural sorting algorithm, *EATCS Bull.*, 76 (2002), 153–162.

[2] Gh. Păun. Computing with membranes, *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143.

[3] C. S. Calude, Gh. Păun. *Computing with Cells and Atoms: An Introduction to Quantum, DNA, and Membrane Computing*, Taylor and Francis, New York, 2000.

[4] J. J. Arulanandham. *The Bead–Sort.* Animation, `www.geocities.com/natural_algorithms/josh/beadsort.ppt`.