Molecular and Cellular Computing

Lecture series at Universidad Politécnica de Madrid

Martyn Amos
Department of Computing and Mathematics
Manchester Metropolitan University
United Kingdom

http://www.martynamos.com

Day 2: From *in vitro* to *in vivo*
1. Models, Lab Work and the Transition

Manchester Metropolitan University

Introduction

- After Adleman's original experiment (and Lipton's follow-up paper) there quickly followed a large number of theoretical papers
- We have already covered the parallel filtering model
- In this lecture we summarise some of the more significant theoretical models of DNA computation

Manchester Metropolitan University

Boolean circuits

- Followed on from Ogihara and Ray
- We want to simulate circuits of NAND gates
- Useful function - provides complete basis
- Advances in silicon (eg. cMOS) often start with NAND - interesting historical precedent
- Outputs 0 iff both inputs=1, 1 otherwise
- Negated AND function

M. Ogihara and A. Ray, Simulating Boolean circuits on a DNA computer. First Annual Conference on Computational Molecular Biology, pages 326 - 331, ACM Press, 1997. The final version: *Algorithmica* **25**:239 - 250, 1999.

Manchester Metropolitan University

Representing Gates in DNA

- We represent each gate by a single strand
- 3 components - 2 inputs, and output=1
- Output section present=gate outputs 1
- If a gate takes its inputs from gates *x* and *y*, its *first* input section is the complement of *x*'s output section, and its *second* input section is the complement of *y*'s output section

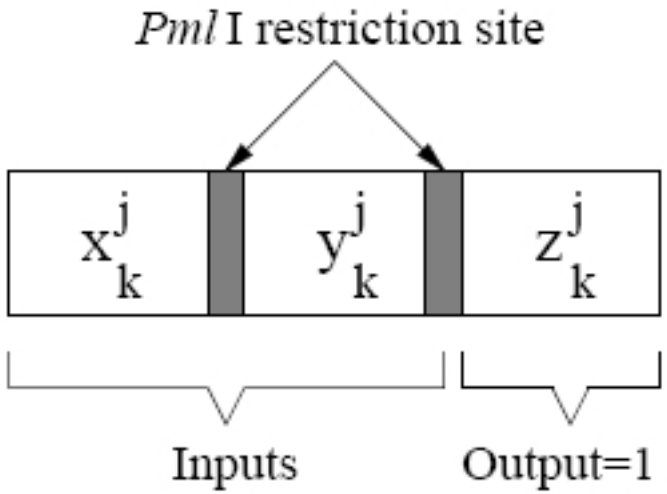
Manchester Metropolitan University

Restriction enzymes

- Destroy double-stranded DNA
- Recognise a specific *restriction site* (eg. GATC)
- RS embedded between two input sections
- Only made d.s. If both input sections are covered
- Gates only *destroyed* if both inputs=1

Manchester Metropolitan University

- Consider gate j at circuit level k
- The two input sequences are represented by x_k^j and y_k^j
- The output sequence is represented by z_k^j

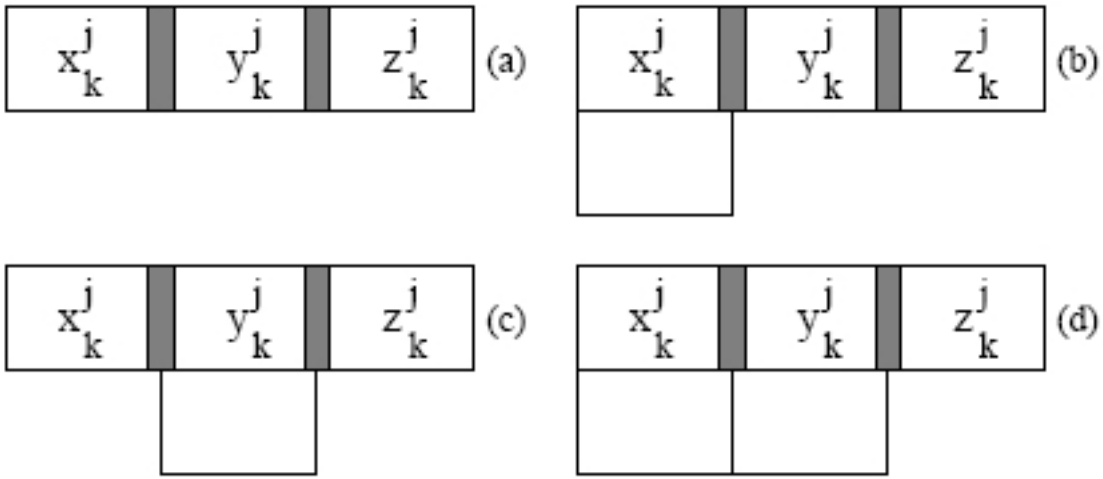


Manchester Metropolitan University

Circuit Construction

- We have a tube for each circuit level
- Thus, we “pour” the output of gates at one level into the gates at the next level
- Output signals “stick” to next level gates
- We want to destroy gates with both input sections covered, since they have both inputs equal to 1
- Achieve this with *restriction enzymes*

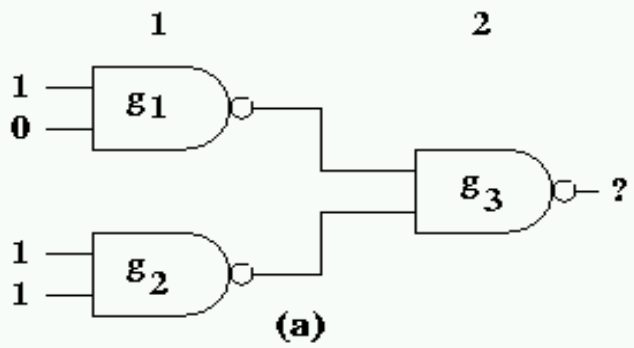
Manchester Metropolitan University



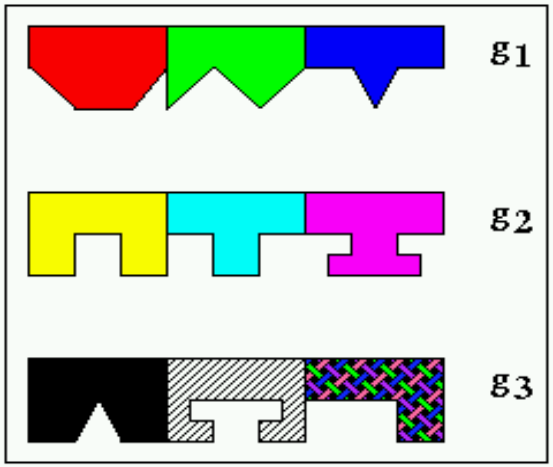
Manchester Metropolitan University

- We have a tube of DNA for each circuit level
- Each tube contains strands representing gates at that level
- We also create an input tube, containing short strands encoding input values of 1
- These stick to gate strands at level 1
- We then add the restriction enzyme to destroy gate strands with both input strands stuck to them
- Sort on length using gel electrophoresis, and retain only those intact strands
- Add pre-prepared primers and restrict again to "snip off" output sections of retained gate strands
- Sort on length again, this time retaining output sections
- Pour these into the tube representing the next level, and repeat until we reach the final level
- If there are intact strands representing the output gate, the circuit has value 1, otherwise it has value 0

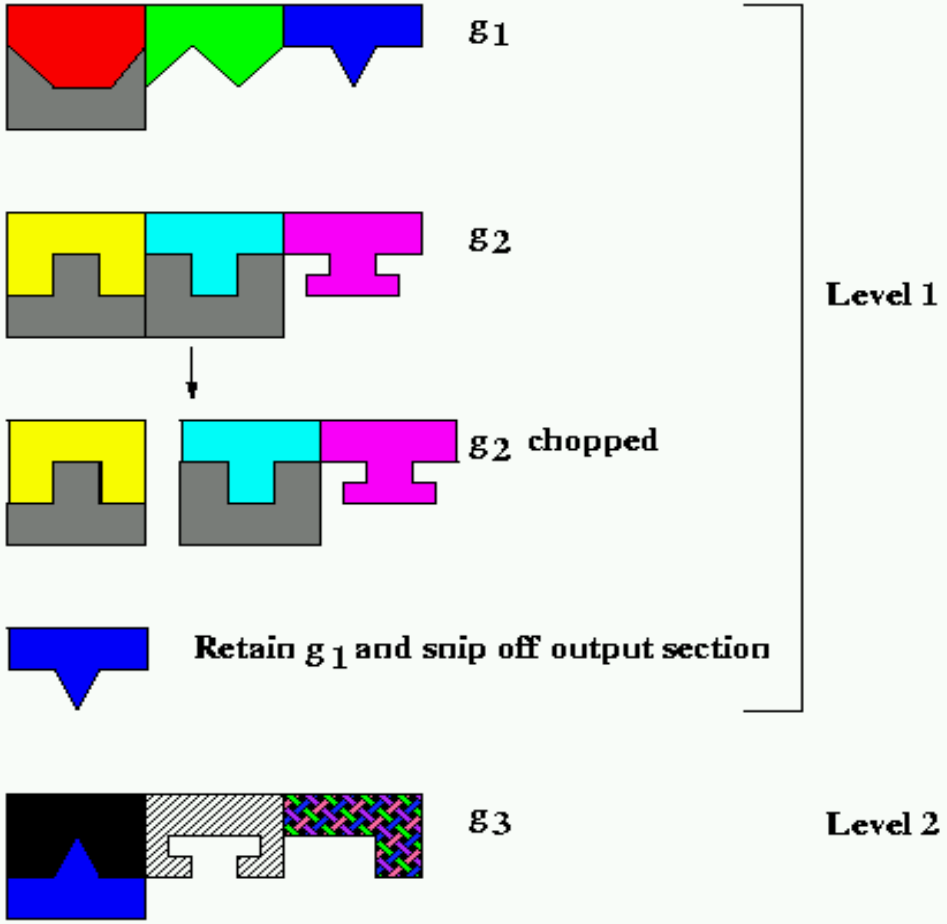
Example


1
2
1
0
g_1
g_3
?
1
1
g_2

(a)


g_1
g_2
g_3

(b)


g_1
g_2
Level 1
g_2 chopped
Retain g_1 and snip off output section
g_3
Level 2
g_3 retained, therefore circuit has output value = 1

(c)

Manchester Metropolitan University

Applications

- Several applications of *in vitro* computing proposed
- We survey some theoretical and experimental advances in the next lecture

Martyn Amos, Gheorge Paun, Grzegorz Rozenberg and Arto Salomaa, Topics in the Theory of DNA Computing, *Theoretical Computer Science* **287**, pp. 3-38, 2002

Manchester Metropolitan University

Sticker model

- Alternative filtering-style model due to Roweis *et al*.
- Operations performed on multisets of strings over the binary alphabet {0,1}
- *Memory* strands are *n* digits in length, and contain *k* non-overlapping substrings of length *m*
- Substrands are numbered contiguously, with no gaps
- Each substrand corresponds to a bit, and may be *on* or *off*

Leonard M. Adleman, Paul W. K. Rothemund, Sam Roweis and Erik Winfree, On Applying Molecular Computation To The Data Encryption Standard, Proc. DNA Computing II, 1996.

Manchester Metropolitan University

Sticker model operations

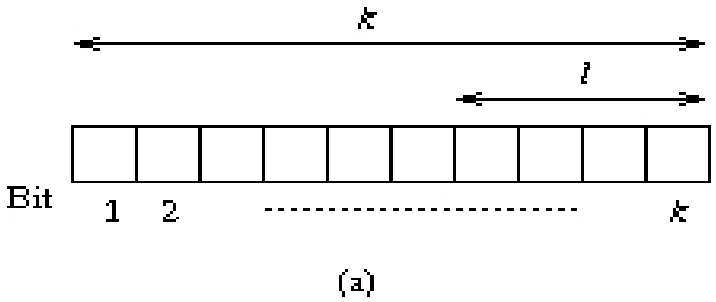
- *merge*. Create multiset union of two tubes
- *separate(N,i)*. Given a tube *N* and an integer *i*, create two new tubes +(*N*,*i*) and -(*N*,*i*), where +(*N*,*i*) contains all strings in *N* with bit *i* set to *on* (and -(*N*,*i*) to *off*)
- set(*N*,*i*). Produce a new tube from *N* in which the *i*th bit of every memory strand is set to *on*
- *clear(N,i).*Produce a new tube from N in which the *i*th bit of every memory strand is set to *on*

Manchester Metropolitan University

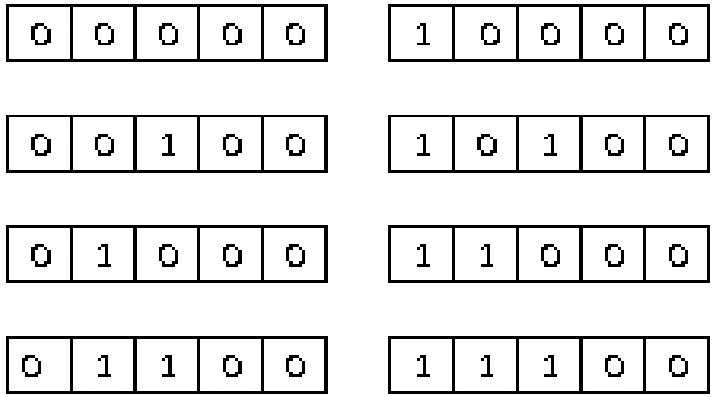
Sticker model computations

- Initial input to a computation consists of a *library* of memory strands
- A (*k*,*l*) library, where $1 \leq l \leq k$, consists of memory strands with *k* bits, the first *l* of which may be either *on* or *off*, and the last *k*-*l* of which are *off*
- In an initial tube the first *l* bits represent the input to the computation, and the remainder are used for working storage and output

Sticker model strand scheme



(a)



(b)

Complete (5,2) library

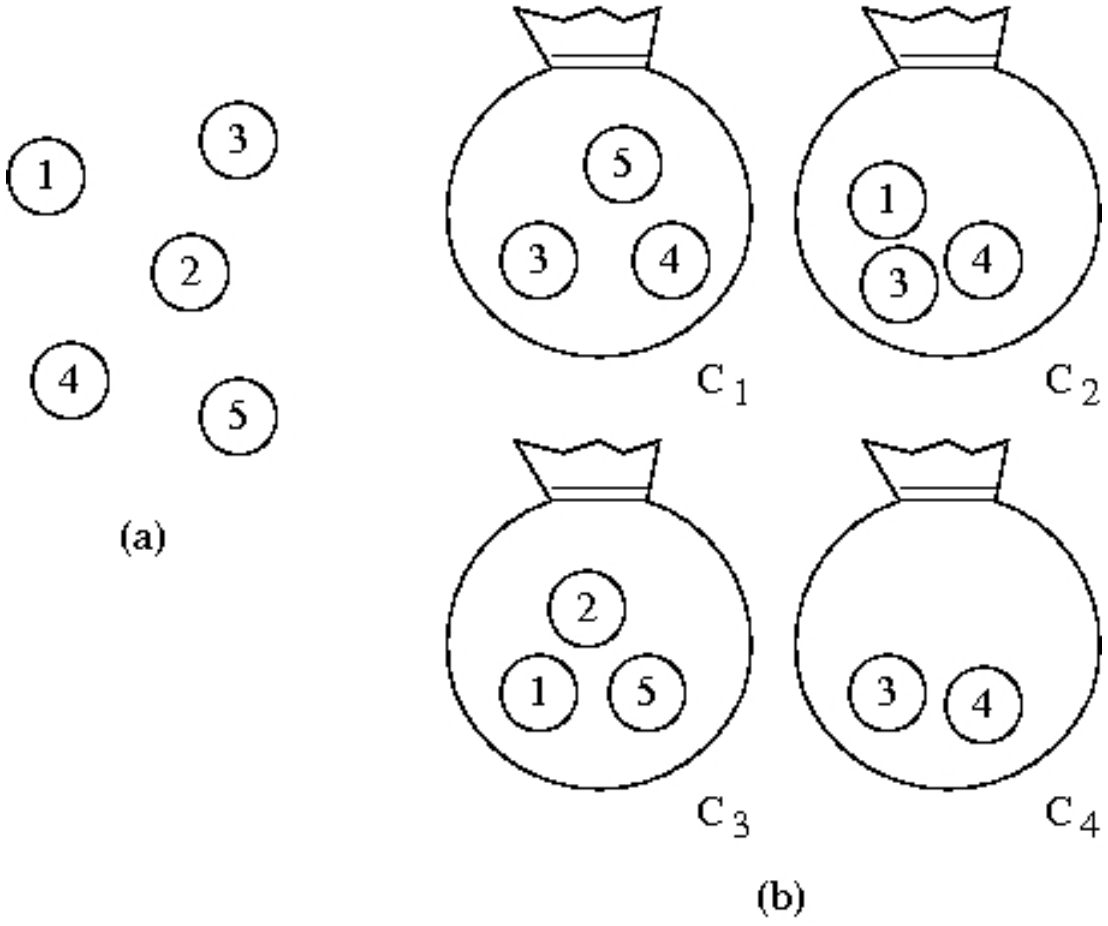
Manchester Metropolitan University

Example algorithm

- Sticker solution to the *Minimal Set Cover Problem*
- Given a set of different *objects* and a set of *bags* containing different subsets of the possible set of objects, what is the smallest number of bags a person must hold to ensure she has at least one of each object?

Manchester Metropolitan University

Example problem



Manchester Metropolitan University

Algorithm

- In previous example, p=5, q=4
- Memory strands have $k=p+q$ bits, and the initial tube N_0 contains a $(p+q,q)$ library
- Each memory strand represents a possible subset of bags from set C, with the first q bits encoding the presence/absence of bags
- Working bits represent the subset of *objects* encompassed by the bags "held"

Manchester Metropolitan University

Algorithm

- For example, a memory strand encoding 0101110110 is interpreted as follows:
- The first four bits encode the fact that bags C_2 and C_4 are held;

  0101110110
- C_2 U C_4 = {1,3,4}, represented by the last 5 bits

  0101110110

Manchester Metropolitan University

Algorithm

- When initial library created, working bits are all set to off, so we need to set them to appropriate values
- For a given memory strand, take the total subset of objects encoded by all of the bags it "holds" and use *set* to turn on the bits representing each object from that subset

Manchester Metropolitan University

Table 3.1. N_0 after setup

M	C_1	C_2	C_3	C_4	1	2	3	4	5	Subset
1	0	0	0	0	0	0	0	0	0	empty
2	0	0	0	1	0	0	1	1	5	(3,4,5)
3	0	0	1	0	1	1	0	0	1	(1,2,5)
4	0	0	1	1	1	1	1	1	1	(1,2,3,4,5)
5	0	1	0	0	1	0	1	1	0	(1,3,4)
6	0	1	0	1	1	0	1	1	0	(1,3,4)
7	0	1	1	0	1	1	1	1	1	(1,2,3,4,5)
8	0	1	1	1	1	1	1	1	1	(1,2,3,4,5)
9	1	0	0	0	0	0	1	1	1	(3,4,5)
10	1	0	0	1	0	0	1	1	1	(3,4,5)
11	1	0	1	0	1	1	1	1	1	(1,2,3,4,5)
12	1	0	1	1	1	1	1	1	1	(1,2,3,4,5)
13	1	1	0	0	1	0	1	1	1	(1,3,4,5)
14	1	1	0	1	1	0	1	1	1	(1,3,4,5)
15	1	1	1	0	1	1	1	1	1	(1,2,3,4,5)
16	1	1	1	1	1	1	1	1	1	(1,2,3,4,5)

Manchester Metropolitan University

Algorithm

- Now that we have this initial library, we first wish to retain only those that *cover* the full set of objects (that is, subsets that contain one of each object)
- We achieve this by discarding strands that *do not* have each of their working bits set to *on*
- This gives us the strands depicted on the next slide

Manchester Metropolitan University

Table **3.2.** N_0 after culling

M	C_1	C_2	C_3	C_4	1	2	3	4	5	Subset
4	0	0	1	1	1	1	1	1	1	(1,2,3,4,5)
7	0	1	1	0	1	1	1	1	1	(1,2,3,4,5)
8	0	1	1	1	1	1	1	1	1	(1,2,3,4,5)
11	1	0	1	0	1	1	1	1	1	(1,2,3,4,5)
12	1	0	1	1	1	1	1	1	1	(1,2,3,4,5)
15	1	1	1	0	1	1	1	1	1	(1,2,3,4,5)
16	1	1	1	1	1	1	1	1	1	(1,2,3,4,5)

Manchester Metropolitan University

Algorithm

- Once we have established a set of coverings, we then need to find one that uses the *smallest* number of bags
- We sort the initial tube N_0 into a number of tubes N_0, N_1, ..., N_q, where tube *Ni* contains memory strands encoding coverings using *i* bags

Manchester Metropolitan University

Sorting procedure

- This may be visualized as a row of tubes, stretching from left to right, with N_0 at the left, and N_q at the right
- We loop, maintaining a counter *i* from 1 to 4, each time "dragging" right one tube any strands containing bag *i*
- Imagine it as an inverted form of GE, where "heavier" strands (those containing more bags) go furthest

Manchester Metropolitan University

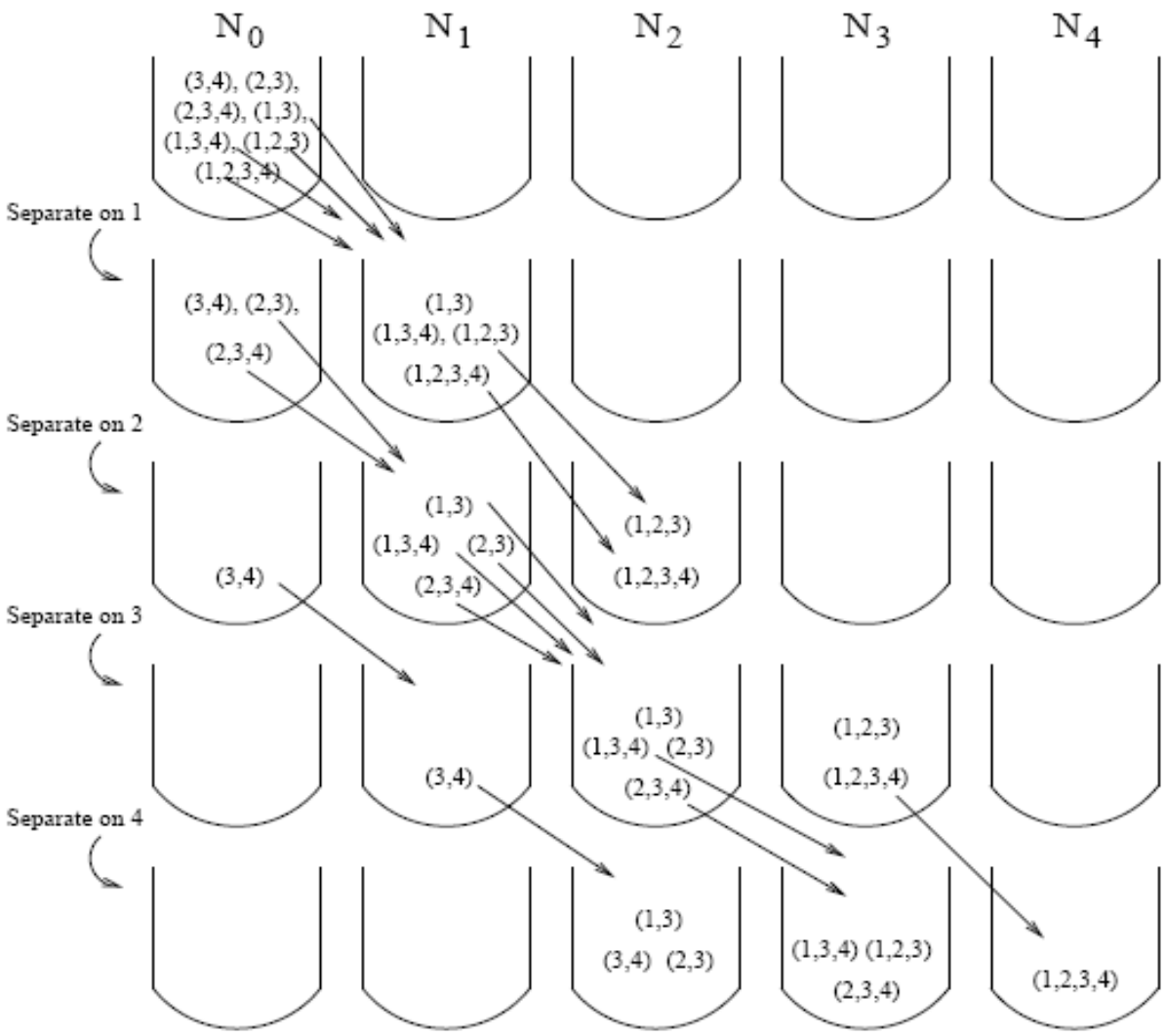


Fig. 3.4. Sorting procedure

Manchester Metropolitan University

Formal algorithm

(1) Initialize (p,q) library in tube N_0
(2) **for** $i = 1$ to q **do begin**
(3) $N_0 \leftarrow separatei(+(N_0, i), -(N_0, i))$
(4) **for** $j = 1$ to $\mid C_i \mid$
(5) $set(+(N_0, i), q + c_i^j)$
(6) **end for**
(7) $N_0 \leftarrow merge(+(N_0, i), -(N_0, i))$
(8) **end for**

This section sets the object identifying substrands. Note that c_i^j denotes the jth element of set C_i. We now separate out for further use only those memory complexes where each of the last p substrands is set to on.

(1) **for** $i = q + 1$ to $q + p$ **do begin**
(2) $N_0 \leftarrow +(N_0, i)$
(3) **end for**

Manchester Metropolitan University

Formal algorithm

We now sort the remaining strands according to how many bags they encode.

(1) **for** $i = 0$ to $q - 1$ **do begin**
(2) **for** $j - 1$ **down to** 0 **do begin**
(3) $separate(+(N_j, i+1), -(N_j, i+1))$
(4) $N_{j+1} \leftarrow merge(+(N_j, i+1), N_{j+1})$
(5) $N_j \leftarrow -(N_j, i+1)$
(6) **end for**
(7) **end for**

Line 3 separates each tube according to the value of i, and line 4 performs the right shift of selected strands. We then search for a final output:

(1) Read N_1
(2) **else if** empty **then** read N_2
(3) **else if** empty **then** read N_3
(4) ...

Manchester Metropolitan University

P systems

- We have already encountered the concept of performing computations by the manipulation of multisets of objects
- This is a well-established form of programming (eg. Petri nets)
- Abstract machines such as the PRAM or TM are sequential in nature
- *Concurrent* models allow multiple processes to execute in parallel

Manchester Metropolitan University

P systems

- The GAMMA (General Abstract Model for Multiset Manipulation) of Banatre and Le Matayer was developed by Berry and Boudol into the *Chemical Abstract Machine*
- Programs viewed almost as *chemical reactions*, with the set being the chemical solution, some reaction *condition* being a property to be satisfied by reacting elements, and some *action* being the resulting product of the reaction
- A computation ends when no further reactions can occur, and a stable state is reached
- Importantly, the CHAM also introduced the notion of a *membrane* construct

Manchester Metropolitan University

Membrane computing

- P systems, a variant of the membrane model, were introduced by Gheorge Paun
- They were inspired by features of biological membranes found in nature
- Membranes act as barriers or filters, separating the cell into distinct regions and controlling the passage of molecules between regions
- *Inspired by*, but not intended to *model*

Gheorge Paun, *Membrane Computing: An Introduction*. Springer, 2002

Manchester Metropolitan University

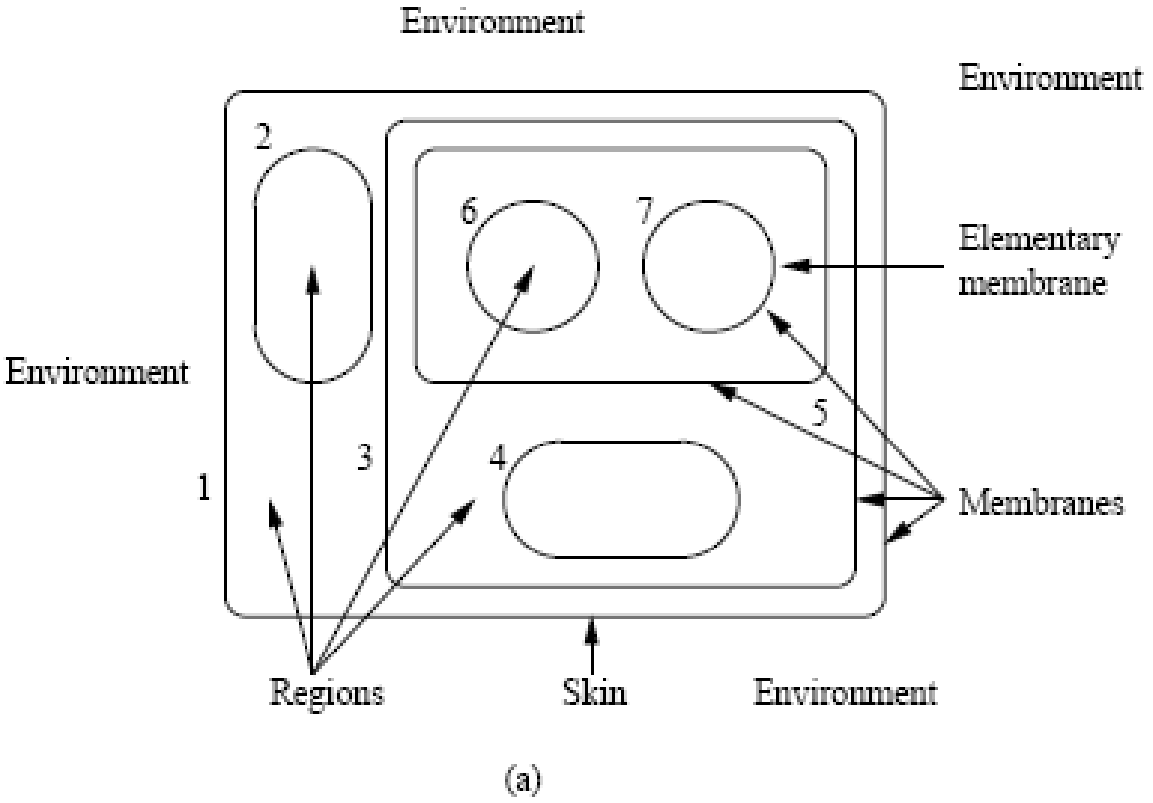
P systems

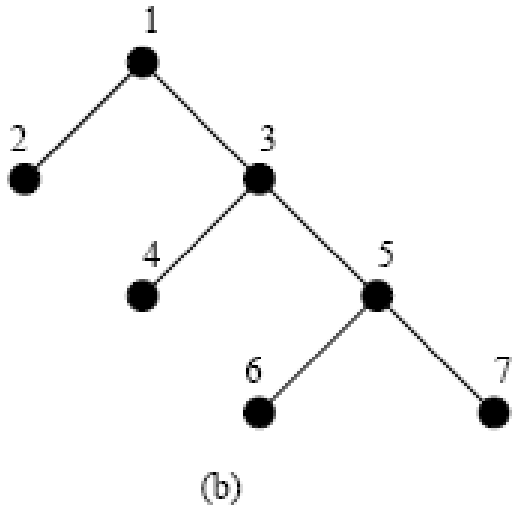
- Membrane structure of a P system is delimited by a *skin* that separates the internals of the system from its outside environment
- Within the skin lies a hierarchical arrangement of membranes that define individual regions
- An *elementary* membrane contains no other membranes, and its region is therefore defined by the space it encloses
- The region defined by a nonelementary membrane is the space between the membrane and the membranes contained directly within it

Manchester Metropolitan University

We attach an integer label to each membrane in order to make it addressable

Since each region is delimited by a unique membrane, we use membrane labels to label regions





Manchester Metropolitan University

P systems

- Each region contains a multiset of *objects* and a set of *rules*
- Objects are represented by symbols from a given alphabet
- Rules transform or “evolve” objects, and are of the form *before* →*after*, meaning “evolve every instance of *before* into an instance of *after*”
- As we are considering multisets, rules may be applied to multiple objects

Manchester Metropolitan University

P system rules

- Evolution rules are represented by a pair (u,v) of strings over the alphabet V
- v may be either v' *or* $v'\Delta$, where Δ is a special character not in V
- v' is a string over $\{a_{here}, a_{out}, a_{in\_j}\}$, where j is a membrane identifier
- a_{here} means “a copy of a remains in this region”
- a_{out} means “send a copy of a through the membrane and out of this region”
- a_{in_j} means “send a copy of a through the membrane of the region labelled j” (only possible if the current region contains j)

Manchester Metropolitan University

P system rules

- When the special symbol Δ is encountered, the membrane defining the current region (assuming it is not the skin) is "dissolved", and the contents placed in the "parent region"
- For simplicity, we omit the "here" subscript, so $a \rightarrow ab$ means "retain a copy of *a* here and create a copy of *b* here"
- $a \rightarrow b\Delta$ means "transform every instance of *a* into an instance of *b* and dissolve the membrane
- Objects on the LHS of evolution rules are "consumed" (removed) during evaluation

Manchester Metropolitan University

Priority

- We may also impose priorities upon rules
- This is denoted by >, and may be read as follows using the example

  $(ff \rightarrow f) > (f \rightarrow \Delta)$

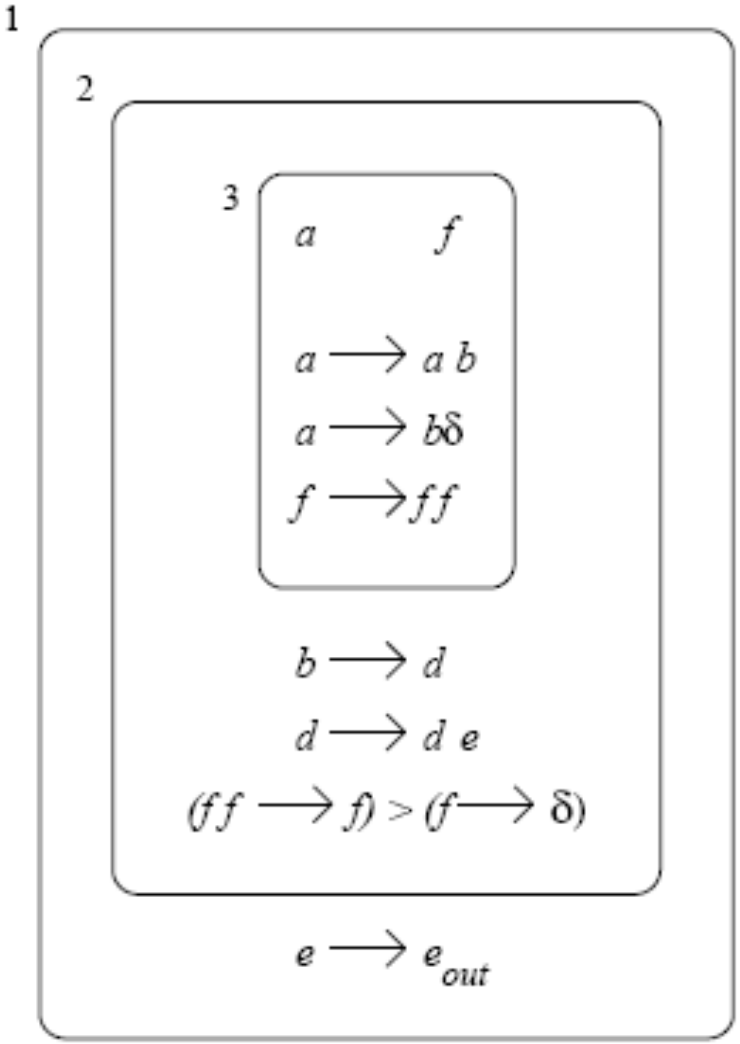
- "Transform *ff* to *f* as often as possible (halving the number of occurrences of *f*) until no instances of *ff* remain, then transform the one remaining *f* to Δ, dissolving the membrane"

Manchester Metropolitan University

Synchronization

- We assume the existence of a “clock” that synchronizes the operation of a system
- At each time step, the configuration of the system is transformed by the application of rules in each region to objects, nondeterministically in parallel, until no further assignment is possible
- A computation halts if no rules can be applied in any region
- The result of a halting computation is the number of objects sent out through the skin to the outside environment

Example

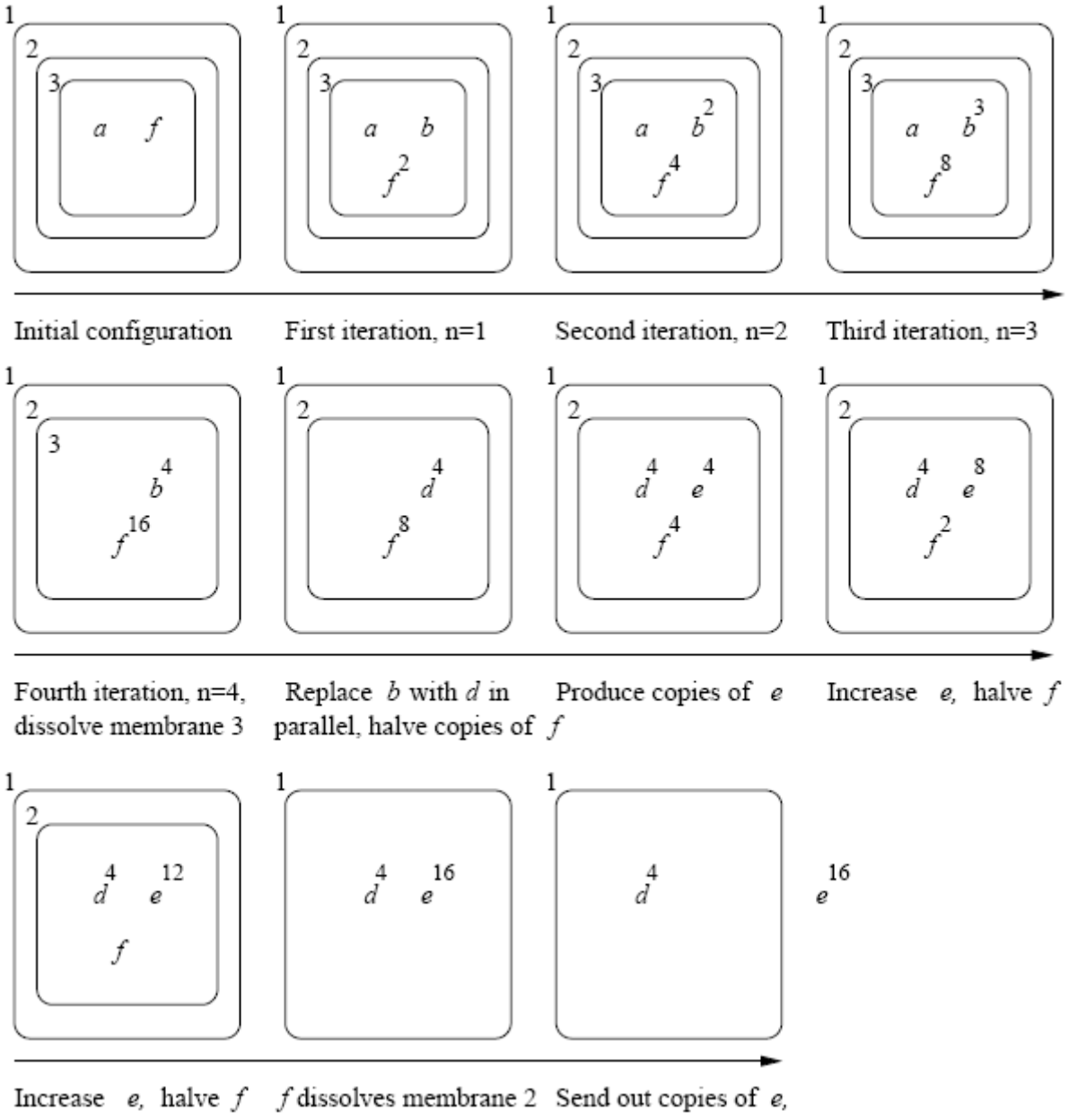


- This P system calculates n^2 for any $n \geq 0$
- Alphabet $V=\{a,b,d,e,f\}$
- Region 3 contains one copy each of *a* and *f*, and three evolution rules
- No objects are present in regions 1/2, so no rules can be applied until we reach region 3
- We iterate the rules $a \rightarrow ab$ and $f \rightarrow ff$ *n* times in parallel, where *n* is the number we wish to square
- This gives *n* copies of *b* and 2^n copies of *f*
- We then use $a \rightarrow b\Delta$, replacing the single *a* with *b* and dissolving the membrane
- This leaves $n+1$ copies of *b* and 2^{n+1} copies of *f* in region 2
- The rules from region 3 are then "destroyed", and the rules from region 2 are now used
- Priority dictates that we use $ff \rightarrow f$ as often as possible, so in one time step we halve the number of copies of *f* and, in parallel, transform b^{n+1} to d^{n+1}
- In the next step, using $d \rightarrow de$, $n+1$ copies of *e* are produced, and the number of copies of *f* again halved
- This step is iterated *n* times, producing $n+1$ copies of *e* at each step, then the membrane is dissolved
- All objects are deposited in the skin, which sends the copies of *e* out to the outside environment

Manchester Metropolitan University

Trace for *n*=4

Example from G. Paun, Computing with membranes. *J. Comp. Sys. Sciences* **61**:1, 108-143, 2000



Manchester Metropolitan University

Recent work

- Since the original development of P systems, several variants have emerged (for example, using “charges” rather than membrane labels)
- P systems have been applied to NP-complete problems, and are offering a rich vein of theoretical results and applications
- P.Frisco, D. W. Corne, Dynamics of HIV infection studied with Cellular Automata and conformon-P systems, *BioSystems* **91**:3, 2008, pp. 531-544