

Computational Models using Peptide- Antibody Interactions

M. Sakthi Balan



**Department of Computer Science and Engineering
Indian Institute of Technology Madras
Chennai – 600036.**

Email: sakthi@cs.iitm.ernet.in

Homepage: theory.cs.iitm.ernet.in/~sakthi

TCS Lab, IITM



Organization

- Motivation of the work
- Background
- Objective of the work
- Peptide computing
- Solving NP- Complete problems
- Modeling gate operations
- Modeling arithmetic operations
- Automata motivated by peptide model
- Conclusion



Motivation

- Solving intractable problems
- Current technology is reaching a physical limit
- Search on for defining new computational models
 - DNA Computing
 - Quantum Computing
 - Peptide Computing (very recent model)
- The idea of using living cells and molecular complexes as potential computing components was given by Richard Feynman



Background

L. Adleman in 1994 solved an instance of Hamiltonian Path Problem using DNA strands. (Nature 1994)

H. Hug and R. Schuler proposed a model using interactions between peptides and antibodies to solve SAT problem. (Bioinformatics 2001)



Objectives

- Using peptide computing for solving NP-Complete problems.
- To show this model is universal.
- Modeling switching operations, arithmetical operations.
- Defining automata model inspired by interactions between peptides and antibodies – Binding-Blocking Automata and study its power and complexity.



Peptide Computing

- Uses peptides and antibodies
- Operation – binding of antibodies to epitopes in peptides
- *Epitope* – The site in peptide recognized by antibody
- Highly *non-deterministic*
- Massive *parallelism*



Peptide Computing Contd..

- Peptides – sequence of amino acids
- Twenty amino acids.
Example – Glycine, Valine
- Connected by covalent bonds



Peptide Computing Contd..

- Antibodies recognizes epitopes by binding to it
- Binding of antibodies to epitopes has associated power called *affinity*
- Higher priority to the antibody with larger affinity power



Solving NP- Complete Problems

Definition

- For finite sequence $M = m_1, m_2, \dots, m_n$ the *doubly duplicated sequence* is

$$MM = m_1, m_1, m_2, m_2, \dots, m_n, m_n$$

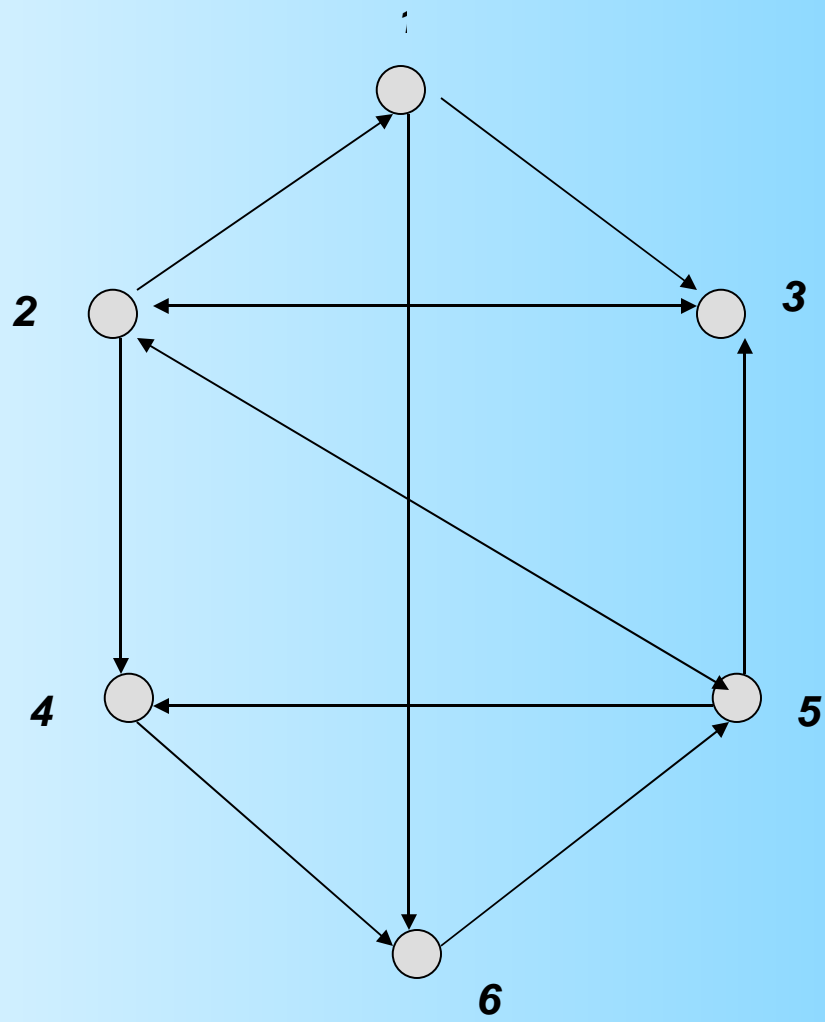
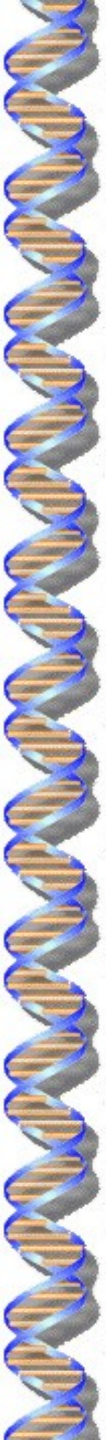
- Doubly duplicated permutation of a finite set S is

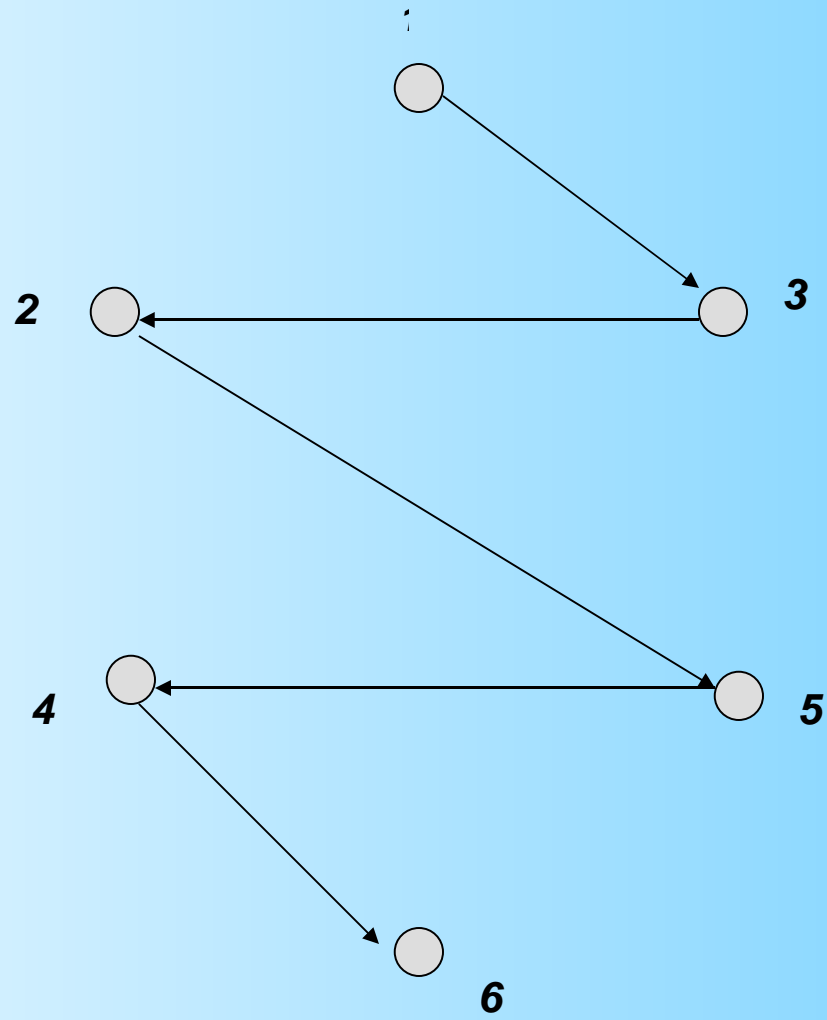
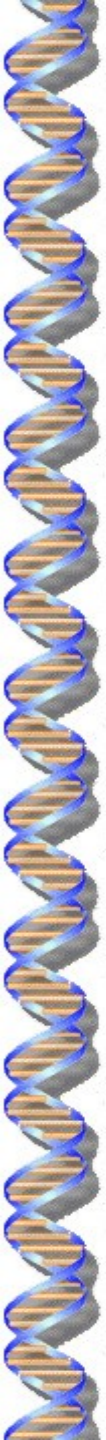
$$\{mm \mid m \text{ is a permutation of the set } S\}$$



Hamiltonian Path Problem

- $G = (V, E)$ is a directed graph
- $V = \{v_1, v_2, \dots, v_n\}$ is the vertex set
- $E = \{e_{ij} \mid v_i \text{ is adjacent to } v_j\}$ is the edge set
- v_1 - source vertex, v_n - end vertex
- **Problem** – Test whether there exists a Hamiltonian path between v_1 and v_n







Peptides Formation

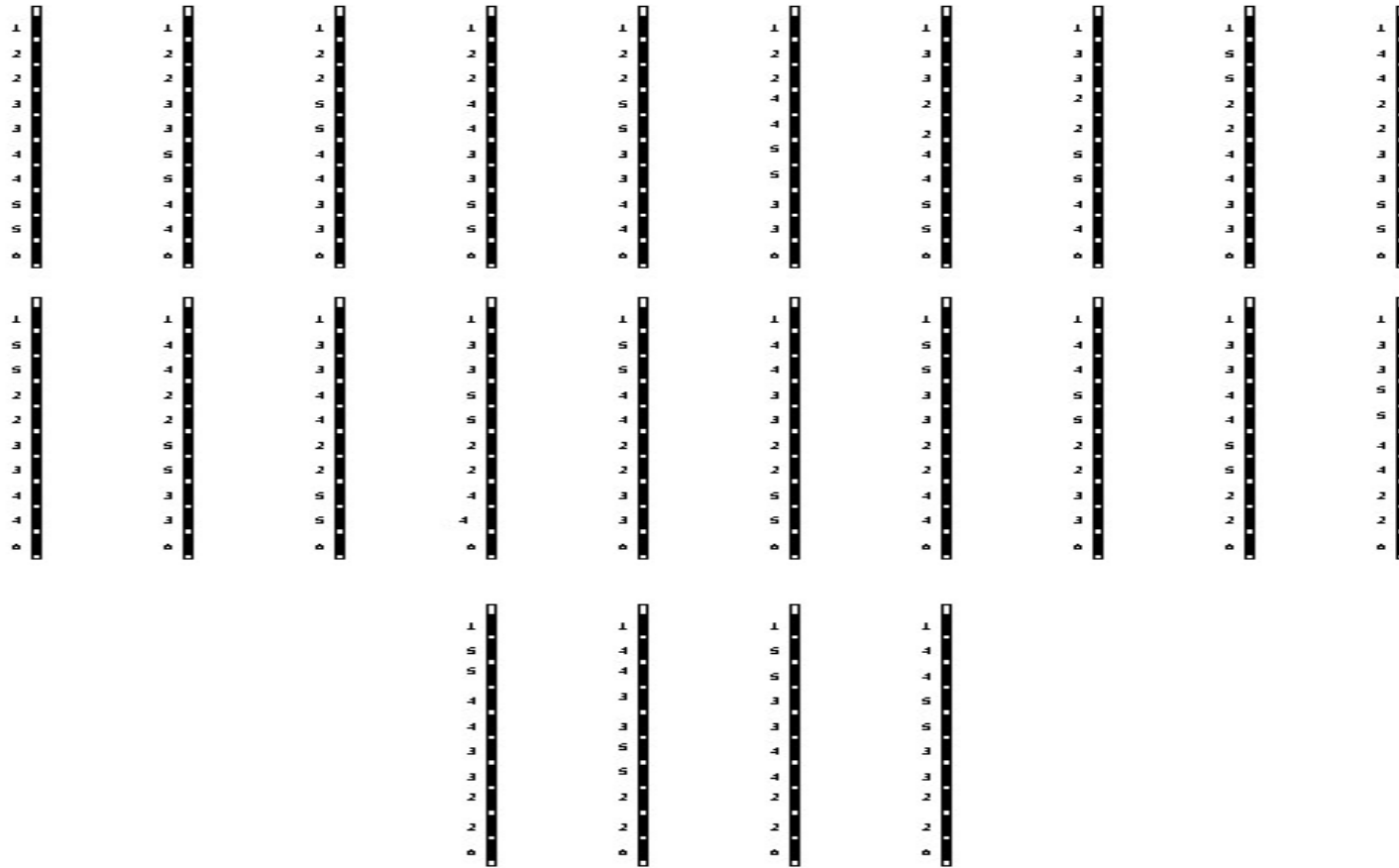
- Each vertex v_i has a corresponding epitope ep_i
- Each peptide has ep_1 on one extreme and ep_n on the other extreme
- All doubly duplicated permutations of $\{ep_2, \dots, ep_{n-1}\}$ are formed in each of the peptide in between ep_1 and ep_n



Antibody Formation

- Form antibodies A_{ij} – site = $ep_i ep_j$ s.t. v_j is adj. to v_i
- Form antibodies B_{ij} – site = $ep_i ep_j$ s.t. v_j is not adj. to v_i
- Form antibody C – site is whole of peptide
- $\text{Affinity}(B_{ij}) > \text{Affinity}(C)$
- $\text{Affinity}(C) > \text{Affinity}(A_{ij})$

Peptide Solution Space

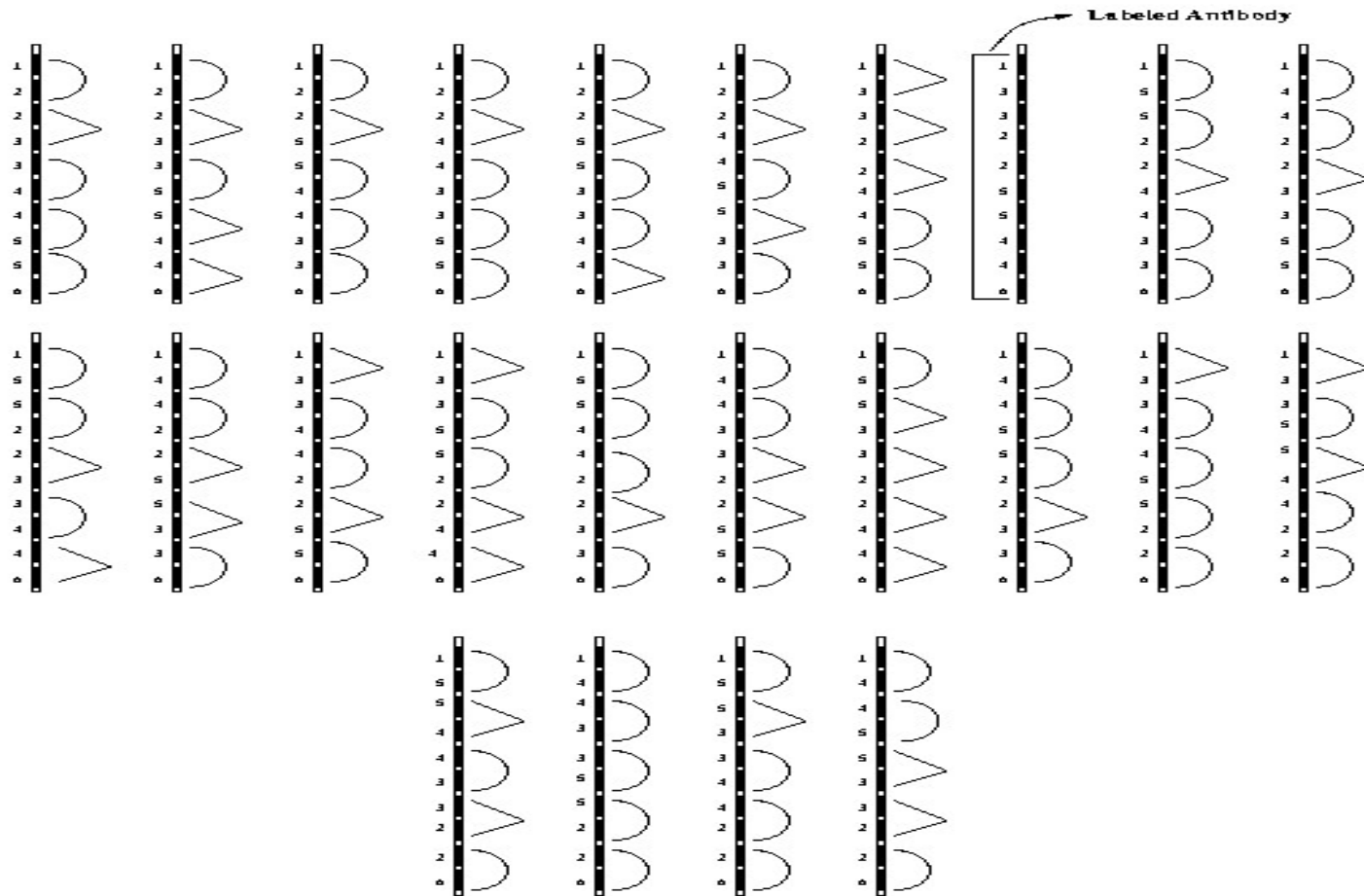




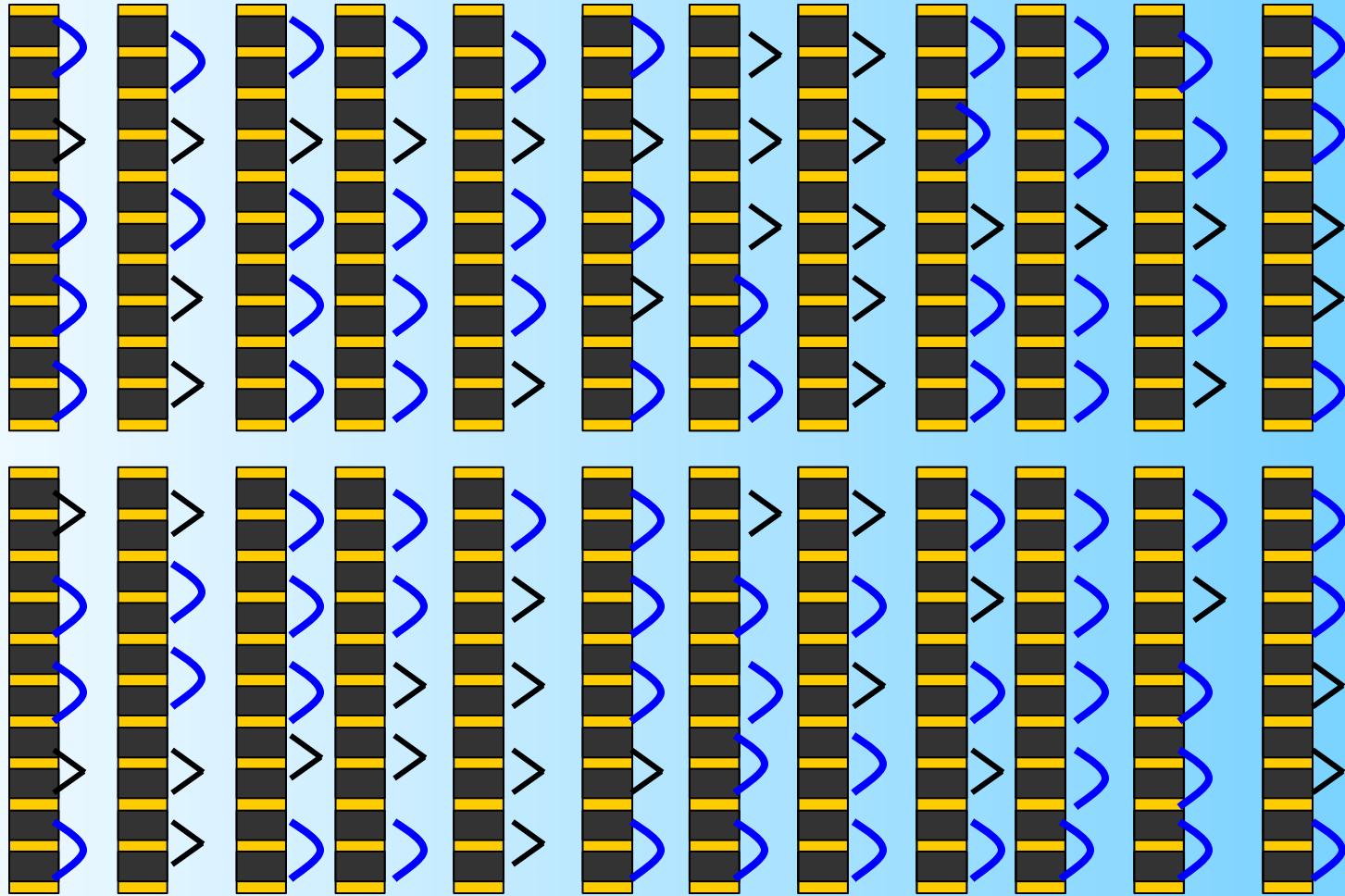
Algorithm

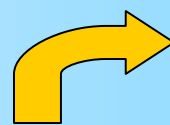
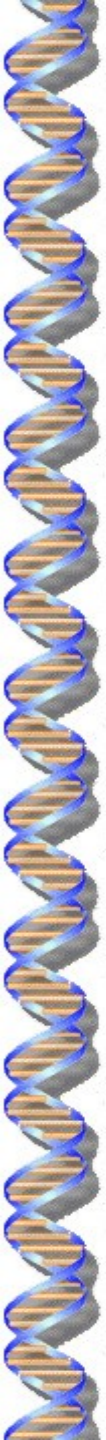
1. Take all the peptides in an aqueous solution
2. Add antibodies A_{ij}
3. Add antibodies B_{ij}
4. Add labeled antibody C
5. If fluorescence is detected answer is *yes* or else the answer is *no*

Peptides with Antibodies

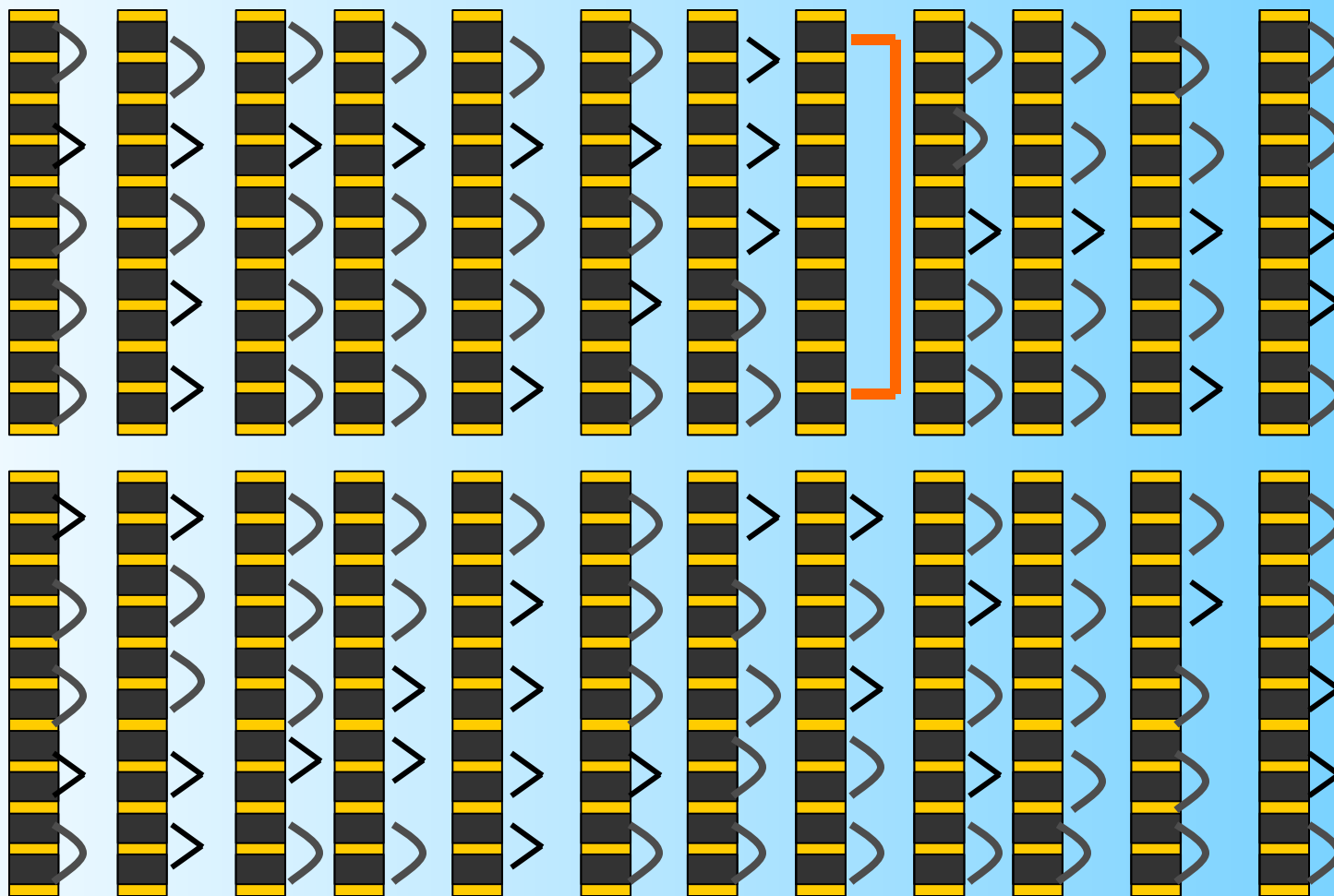


Peptide with Antibodies





labeled antibody





Complexity

- Number of peptides = $(n - 2)!$
- Length of peptides = $O(n)$
- Number of antibodies = $O(n^2)$
- Number of Bio-steps is *constant*



Exact Cover by 3-Sets Problem

- **Instance** A finite set $X = \{x_1, x_2, \dots, x_n\}$, $n = 3q$ and a collection C of 3-elements subsets of X
- **Question**: Does C contain an *Exact Cover* for X



Peptide Computing is Computationally Complete

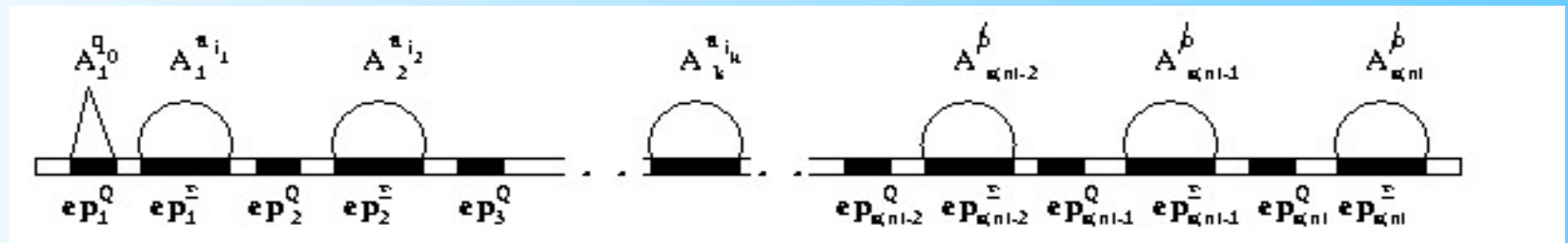
A Turing Machine can be
simulated by a Peptide System

Universality Result Contd..

Peptide without antibodies

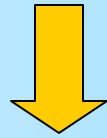


Initial Configuration of Peptide



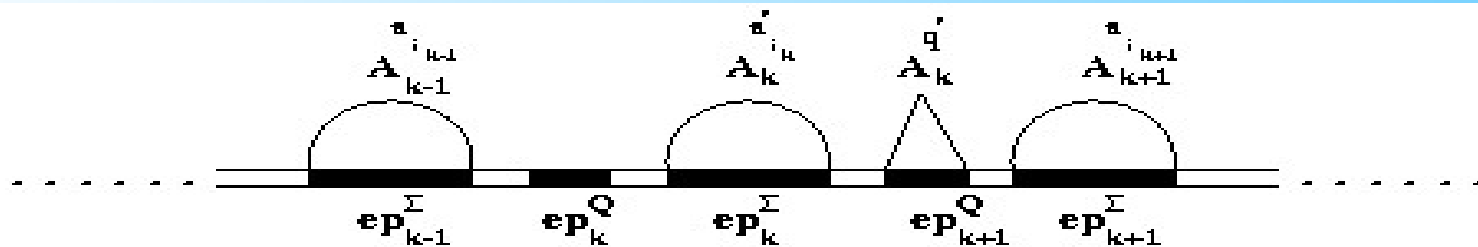
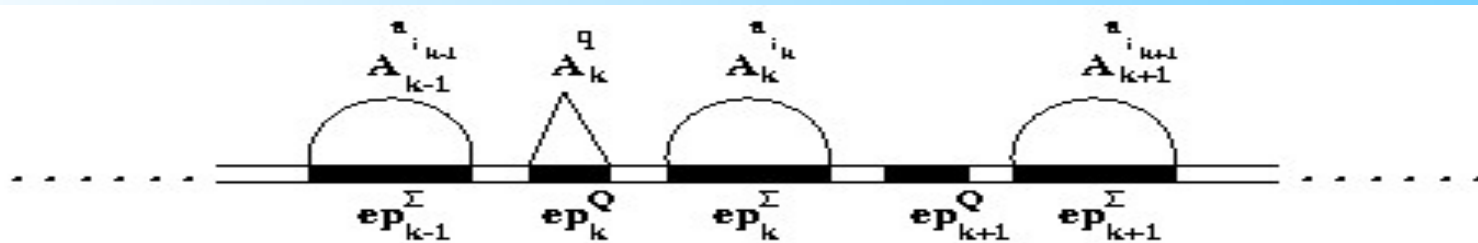
Simulating the Right Move

- M moves from $a_i q a_j a_j$ to $a_i a_j q' a_j$,



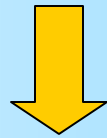
- Add excess of free epitopes ep_k^Σ and ep_k^Q
- Add antibodies $A_k^{a_j}$ and $A_{k+1}^{q'}$
- k is the position of the head prior to the right move

Simulating the Right Move Contd..



Simulating the Left Move

- M moves from $a_i q a_j a_j$ to $q' a_i a_j a_j$




- Add excess of free epitopes ep_k^Σ and ep_k^Q
- Add antibodies $A_{k-j}^{a_j}$ and $A_{k-1}^{q'}$
- k is the position of the head prior to the right move




Complexity

- Peptide system takes $O(t(n))$ time
- Length of the peptide is $O(s(n))$
- Number of peptide is *one*
- Amount of antibodies is

$$O(m \cdot s(n) + l \cdot (s(n)))$$



Switching Operations - OR, AND & NOT gates



Why modeling gate operations?

- Peptide and antibody formation is dependent on the problem.
- Defining basic operations makes the model independent of the problem.



For DNA Computing

- Ogiwara et al, Martyn Amos et al have done the simulation of Boolean circuits for DNA computing.
- H.Hug et al have proposed a model to do parallel arithmetical operations using DNA operations on DNA strands.



Proposed Model

- Consists of a peptide sequence and some set of antibodies.
- Peptide sequence consists of five epitopes,

$$P = y x_1 x x_2 z$$

where each y , x_1 , x , x_2 and z are epitopes.

- Six antibodies denoted by A_1 , A_2 , B_1 , B_2 , C_{init} and C_f .
- Antibodies A_1 , A_2 , B_1 and B_2 denote the inputs.
- C_{init} denote the initial value of the result of the operation.
- C_{init} and C_f denote the output of the operation.

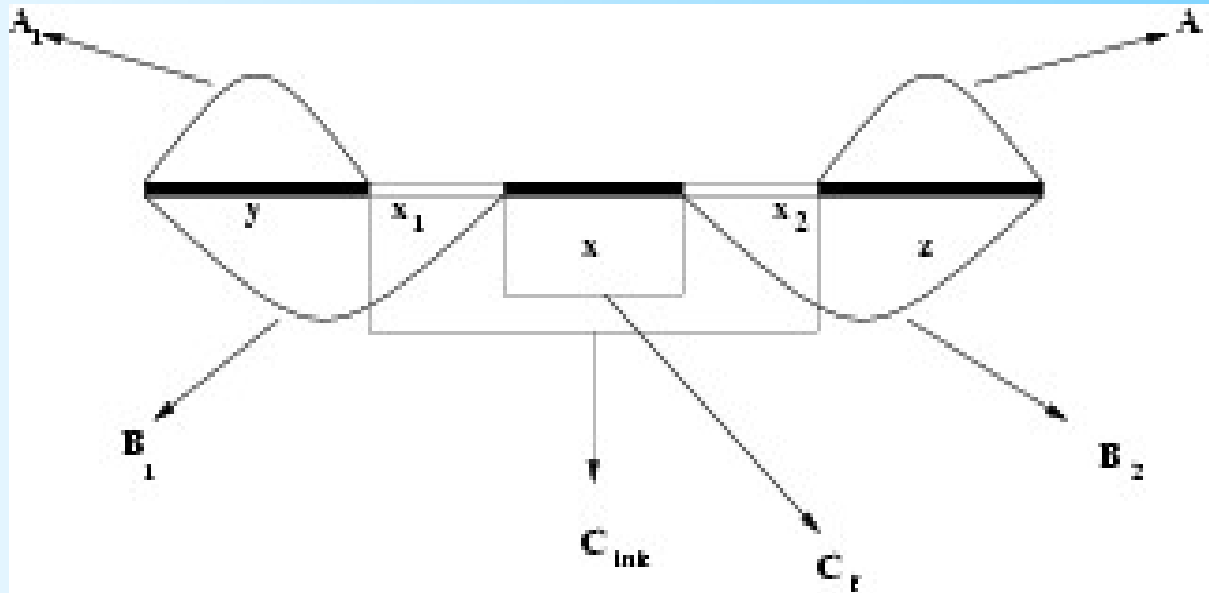


Proposed Model

- Epitopes x and y are the binding places for the antibodies denoting the input.
- Epitope x_1xx_2 is the place where the antibody representing the initial output binds.
- Epitopes x_1x , xx_2 and x are the binding places for the antibodies denoting the output.



Peptide sequence



Peptide Sequence with Antibodies

OR Gate

- Input bits 0 and 1 are represented by the antibodies A_i and B_i respectively where $1 \leq i \leq 2$.
- The antibody C_{init} denotes the bit 0 .
- The antibody C_f (labeled antibody) denotes the bit 1 .
- $\text{epitope}(A_1) = \{y\}$, $\text{epitope}(A_2) = \{z\}$,
- $\text{epitope}(B_1) = \{yx_1\}$, $\text{epitope}(B_2) = \{x_2z\}$,
- $\text{epitope}(C_{init}) = \{x_1xx_2\}$, $\text{epitope}(C_f) = \{x\}$,
- $\text{aff}(B_i) > \text{aff}(C_{init}) > \text{aff}(C_f)$, $1 \leq i \leq 2$.

OR Gate

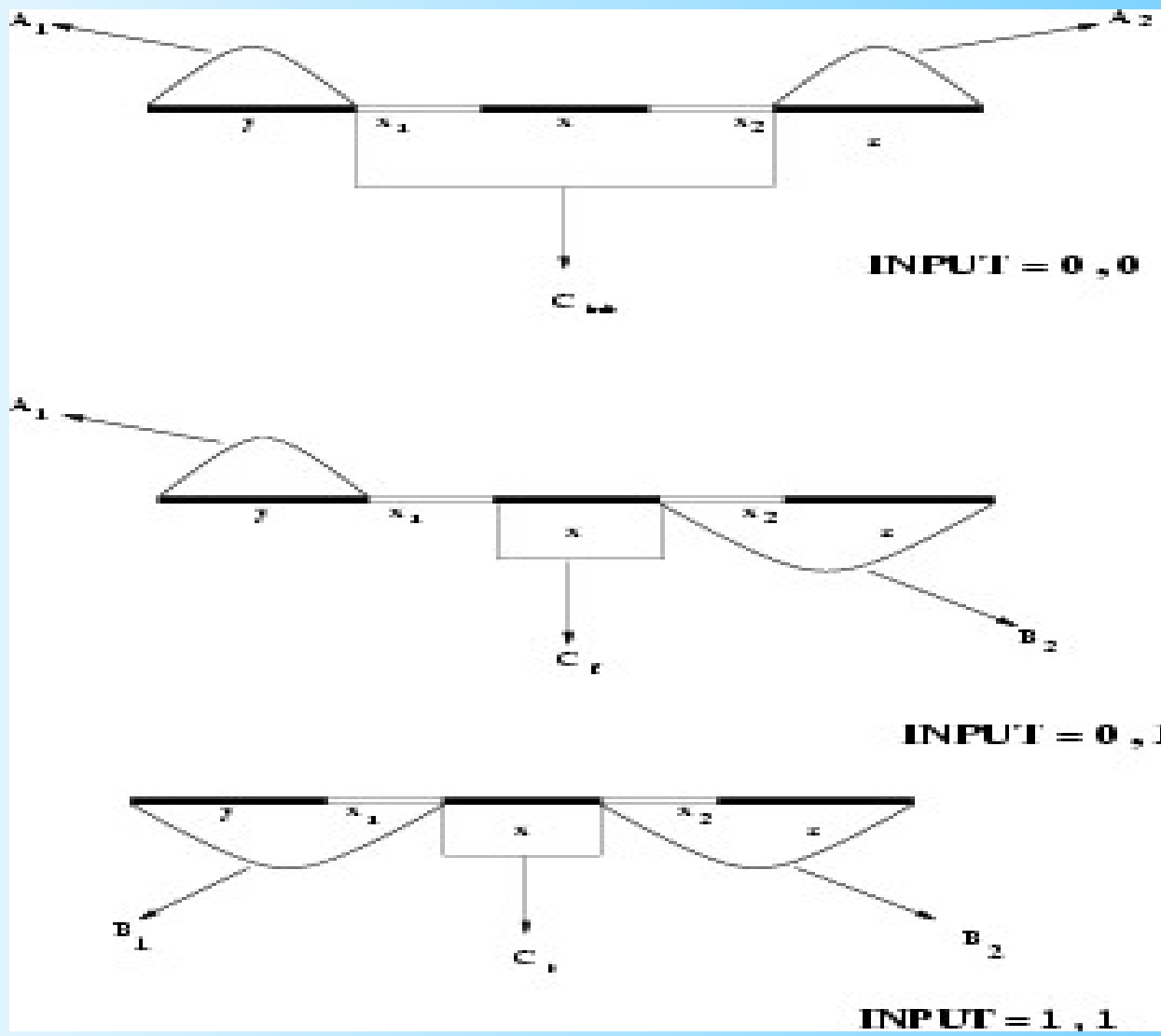
- The output 1 occurs if at least one of the inputs is 1 .
- Start with an initial output of 0 - antibody C_{init} binds to its epitope.
- C_{init} is toggled if at least one 1 comes as an input. For this to be carried out the epitopes for the antibody C_{init} and the antibody B_i , $1 \leq i \leq 2$ are taken as overlapping ones.
- $\text{aff}(B_i) > \text{aff}(C_{init}) > \text{aff}(C_f)$, $1 \leq i \leq 2$.
- This facilitates toggle of output bit to 1 - antibody C_f binds to its epitope.



Algorithm

- Take the peptide sequence P in an aqueous solution.
- Add the antibody C .
- Add antibodies corresponding to the input bits. For example if the first bit is 1 and the second bit is 0 then add antibodies B_1 and A_2 .

- Add antibody C_f .
If the output has to be seen the antibody C_f can be gives some color so that at the end of the algorithm if fluorescence is detected the output will be 1 or else it will be 0 .





Other Gates

- This model has been extended to other gates – *AND*, *NOT*, *NAND*, *NOR* and *XOR*
- We also extend this model to simulate Boolean circuits.

Boolean Circuit

- An n -input m -output Boolean circuit is modeled as a directed acyclic graph $G=(V,E)$.
- V consists of two disjoint sets X_n and G
- X_n denotes the n -inputs and G denotes all the gates, of which m are output gates.
- Each input vertex has in-degree 0 and is associated with a single Boolean variable x_i
- Every vertex denoting gate has in-degree 2 and is associated with *NAND* operation
- The m vertices denoting the output vertices have 0 out-degree

Boolean Circuit (Contd..)

- An assignment of Boolean variables from $\{0,1\}^n$ to X_n induces Boolean values at the output gates
- The Boolean network is said to construct the function, $\longrightarrow \longrightarrow$

$$f(x_n) : \{0,1\}^n \longrightarrow \{0,1\}^m = \langle f^{(i)}(X_n) : \{0,1\}^n \longrightarrow \{0,1\} \rangle,$$

where $i = 1$ to m

if $\forall \alpha \in \{0,1\}^n, t_i(\alpha) = f^{(i)}(\alpha)$ for all $i = 1$ to m

t_1, t_2, \dots, t_m are output gates



Boolean Circuit (Contd..)

- The size of the network $C(S)$, S – Boolean network
- The depth of circuit $D(S)$
- The breadth of the circuit $B(S)$
- We denote j^{th} gate at k level by $g_{k,j}$
- For every gate $g_{k,j}$ we form peptide sequence $p_{k,j}$

Formation of Peptide Sequence

- At level $k=0$, the j^{th} gate is denoted by the sequence

$$p_{0,j} = y_{0,j} x_{0,j,1} x_{0,j,2} z_{0,j}$$

- For level $k>0$ if $k=m$, $j=p$ then

$p_{m,p} = x_{m-1,n,1} w_{m-1,n} x_{m-1,n,2} w_{m,p} x_{m-1,s,1} w_{m-1,s} x_{m-1,s,2}$ if the gate $g_{m,p}$ gets inputs from the gates $g_{m-1,n}$ and $g_{m-1,s}$

$w_{0,i} = x_{0,i}$ for all i and $w_{m,n}$ unique for all m, n where $m > 0$.

Formation of Antibodies

- At level $k=0$, there are seven antibodies for each gates, $(A_1)^{0,j}$, $(A_2)^{0,j}$, $(B_1)^{0,j}$, $(B_2)^{0,j}$, $(C_{init})^{0,j}$, $(C_f)^{0,j}$ and $(R)^{0,j}$
- $epi((A_1)^{0,j}) = y_{0,j}x_{0,j,1}$ $epi((A_2)^{0,j}) = x_{0,j,2}z_{0,j}$
- $epi((B_1)^{0,j}) = y_{0,j}$ $epi((B_2)^{0,j}) = z_{0,j}$
- $epi((C_{init})^{0,j}) = x_{0,j,1}x_{0,j}x_{0,j,2}$
- $epi((C_f)^{m,j}, (R)^{m,j}) = w_{m,j}$

Formation of Antibodies

- At level $k > 0$, $epi((C_{init})^{k,j}) = x_{k-1,n,2} W_{k,j} x_{k-1,s,1}$

provided the gate $g_{k,j}$ get inputs from gates

$g_{k-1,n}$ and $g_{k-1,s}$

- The peptide sequence $g_{k,j}$ get antibodies $out(g_{k-1,n})$ and $out(g_{k-1,s})$ as inputs



Formation of Antibodies

- $affinity((R)^{m,j}) > affinity((C_{init})^{m,j})$
- $affinity((C_{init})^{m,j}) > affinity((C_f)^{m,j})$
- $affinity((C_{init})^{m-1,j}) > affinity((C_{init})^{m,i})$



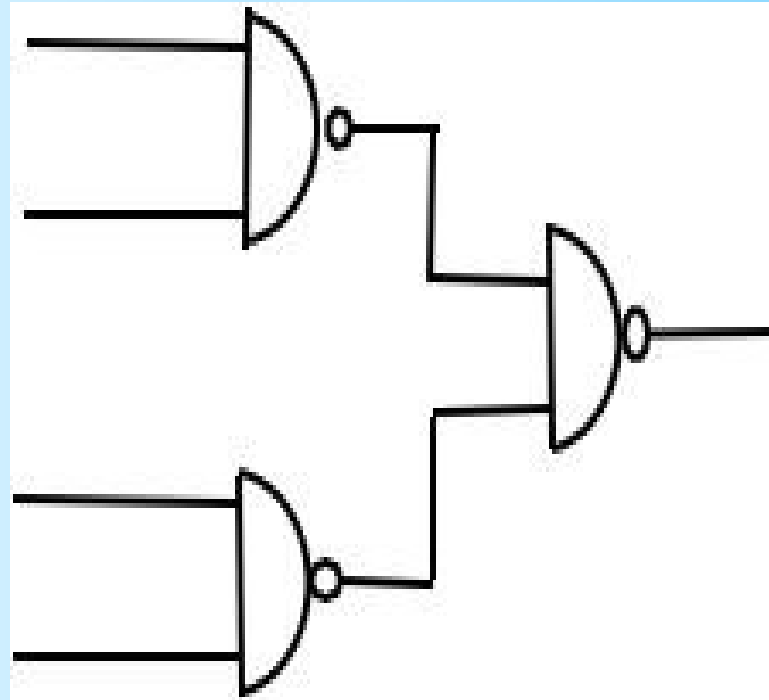
Simulating Boolean Circuit

- Antibodies denoting the inputs are put into the solution containing peptide sequences $p_{0,j}$
- All the peptides $p_{m-1,i}$ are initialized with antibody $(C_{init})^{m-1,i}$

Simulating Boolean Circuit

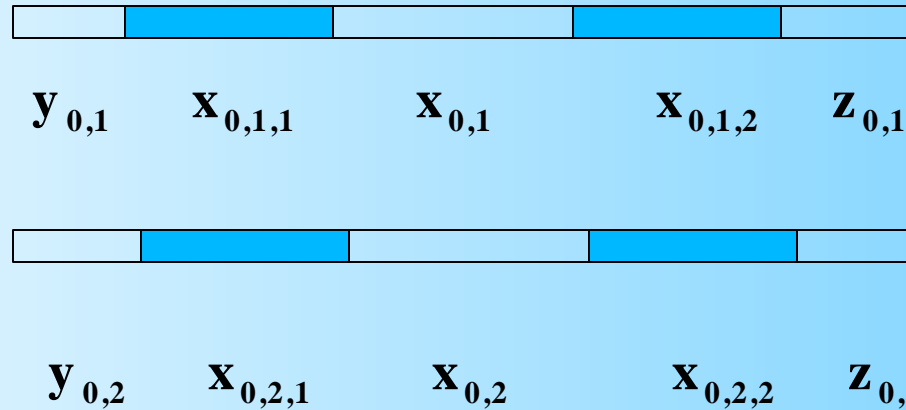
- Suppose gate $g_{k,j}$ gets inputs from $g_{k-1,n}$ and $g_{k-1,s}$
- Put $(R)^{m-1,n}$ and $(R)^{m-1,s}$ are put into the solution to get the outputs of the gates $g_{m-1,n}$ and $g_{m-1,s}$
- Put the antibody $(C_f)^{k,i}$

Example



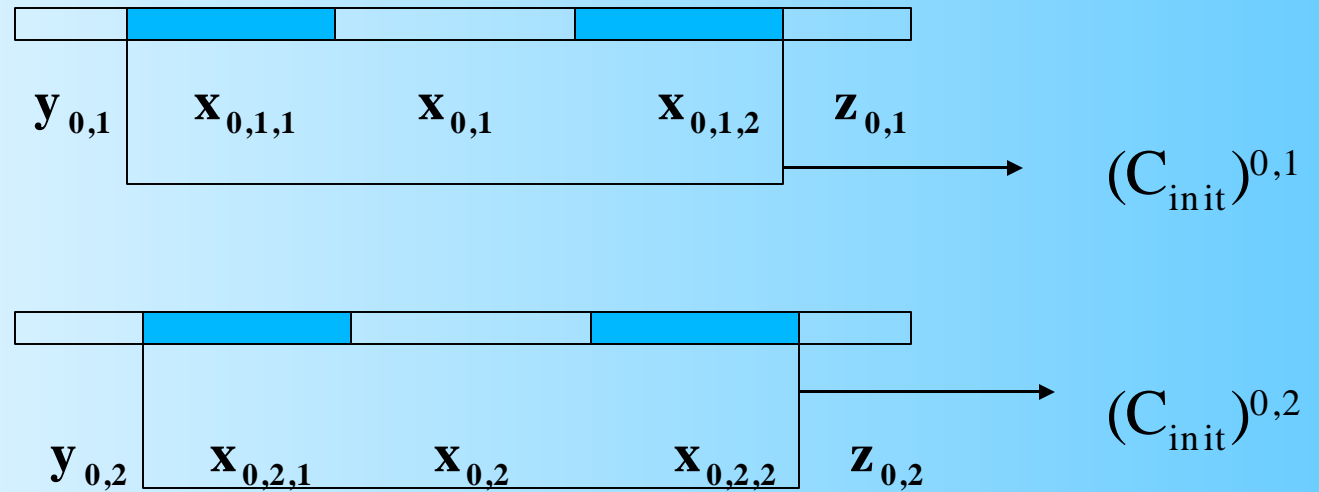
4- input and 1- output

Example (Contd..)



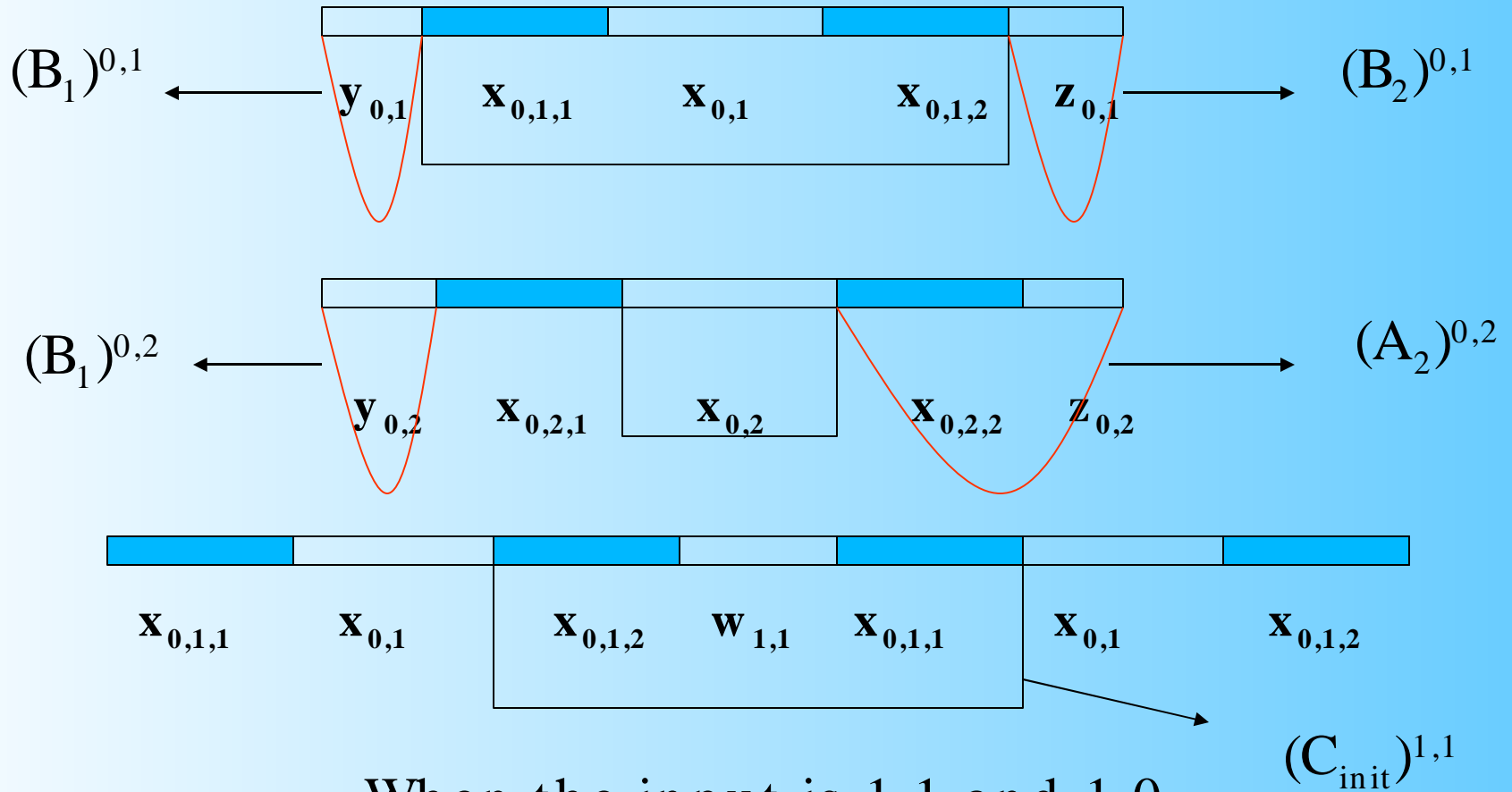
Peptide sequences representing
the first two gates at level 0

Example (Contd..)



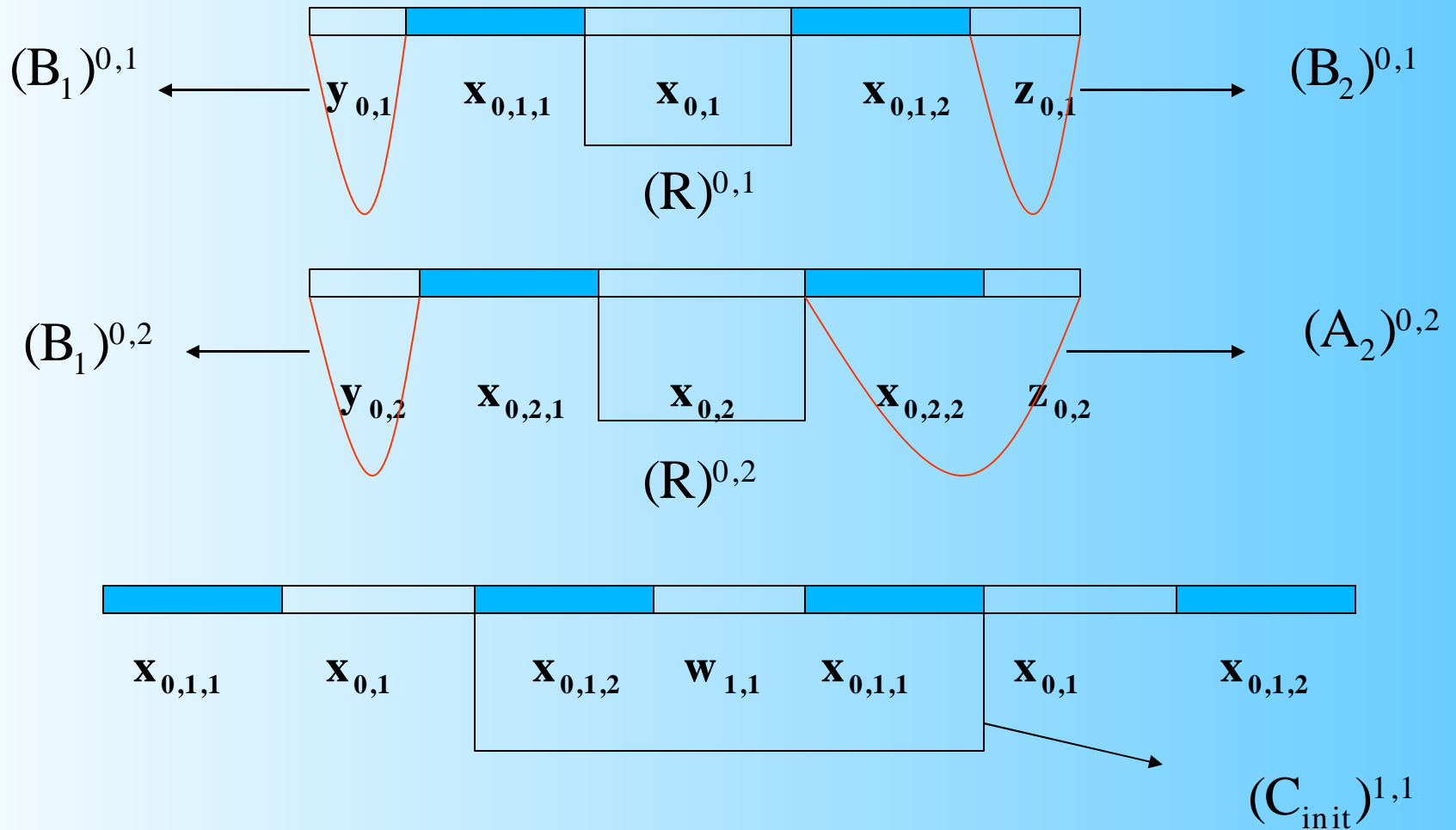
Gates initialized at level 0

Output in the first two gates

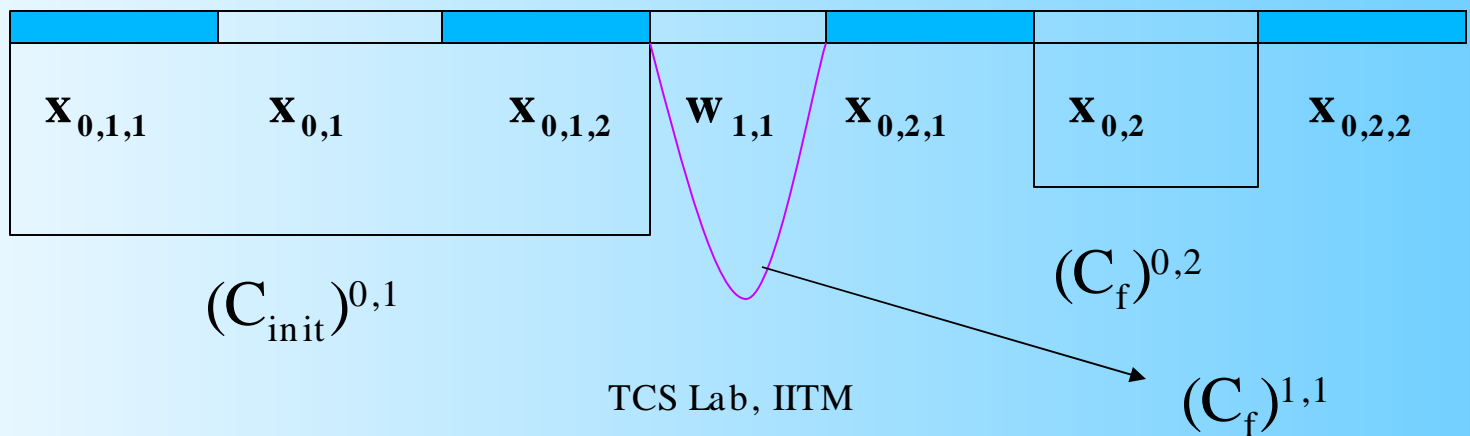
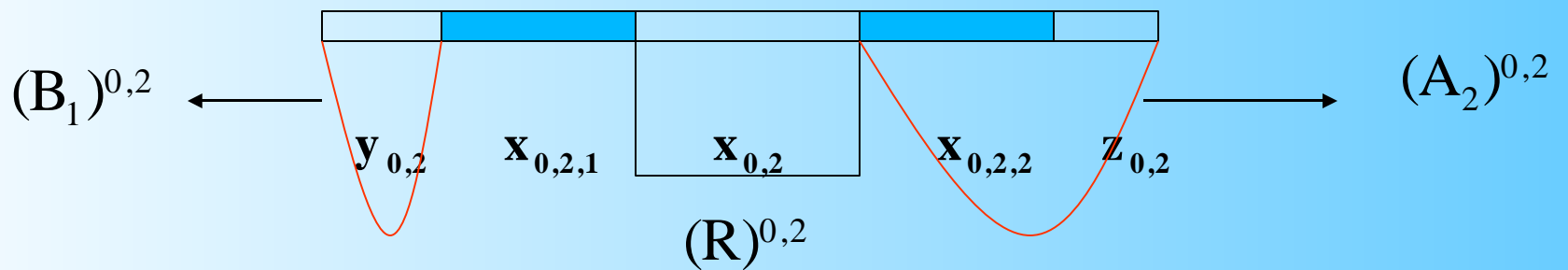
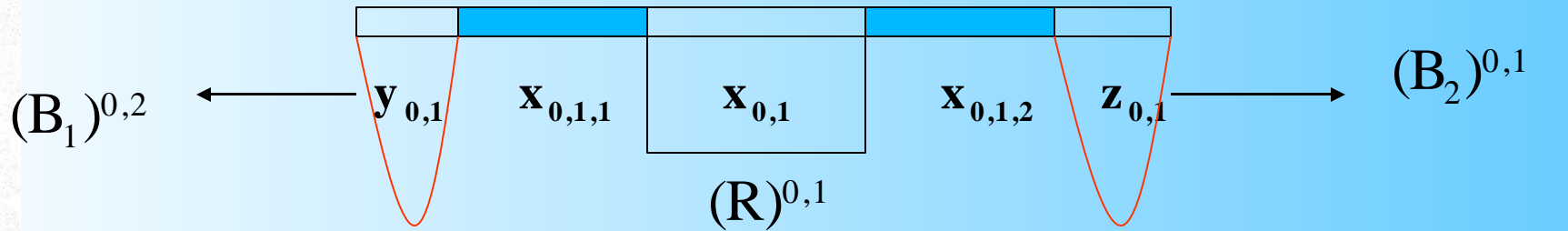


When the input is 1,1 and 1,0

Outputs removed



Output getting as input and output at the third gate





Modeling Arithmetic Operations



Arithmetic Operations Proposed Model


- Consists of a peptide and set of antibodies
- Peptide sequence has n position specific epitopes
- Epitopes $ep_i = y_i x_i z_i$, y_i and z_i are *switching epitopes* for the i^{th} bit.



y_4 x_4 z_4

y_0 x_0 z_0

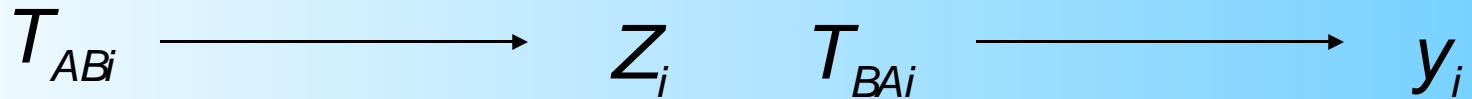
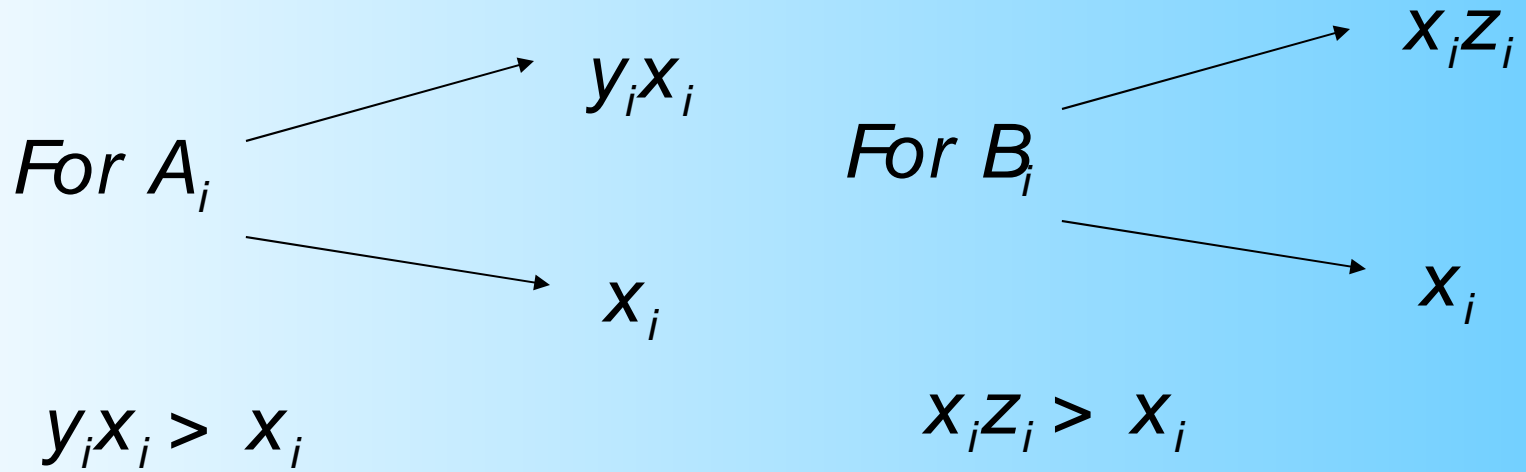
Peptide Sequence for a 5-bit number



Antibodies

- $\mathcal{A} = \{A_0, A_1, \dots, A_{n-1}\}$
- $\mathcal{B} = \{B_0, B_1, \dots, B_{n-1}\}$
- $\mathcal{T}_{\mathcal{A}\mathcal{B}} = \{T_{AB0}, T_{AB1}, \dots, T_{AB(n-1)}\}$
- $\mathcal{T}_{\mathcal{B}\mathcal{A}} = \{T_{BA0}, T_{BA1}, \dots, T_{BA(n-1)}\}$

Binding Sites






Affinity

- $aff(T_{ABi}) > aff(A_i)$
- $aff(T_{BAi}) > aff(B_i)$
- $aff(T_{ABi}) = aff(T_{BAi})$

What it denotes?

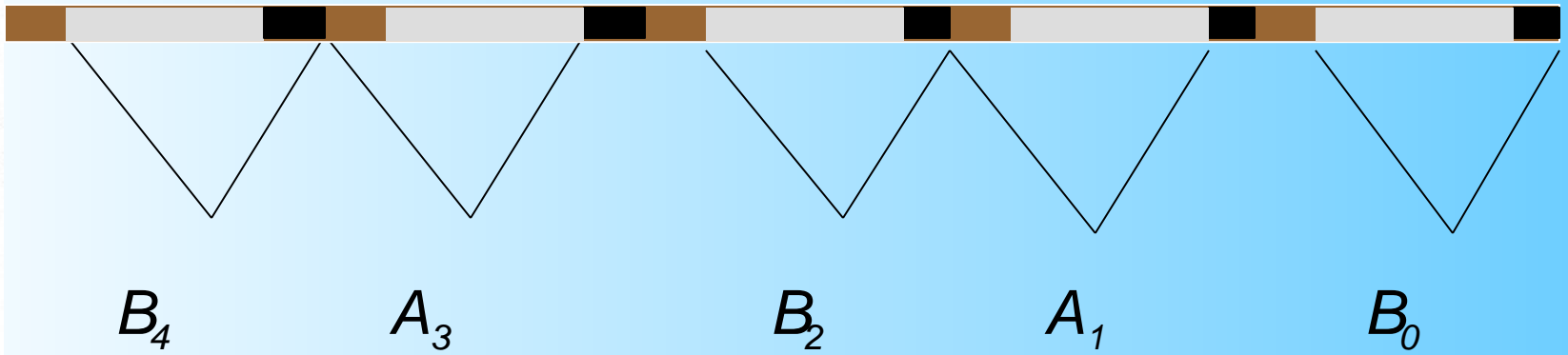
- A_i – denotes i^{th} bit is zero
- B_i – denotes i^{th} bit is one
- T_{ABi} – used to switch i^{th} bit from zero to one
- T_{BAi} – used to switch i^{th} bit from one to zero



Representation of Binary Numbers

- If the i^{th} bit is 0 then the antibody A_i is bounded to the epitope $y_i x_i$
- If the i^{th} bit is 1 then the antibody B_i is bounded to the epitope $x_i z_i$

Example



10101



Addition of Two Binary Numbers

$$A = a_{n-1}a_{n-2} \dots a_0$$

$$B = b_{n-1}b_{n-2} \dots b_0$$

$$C = c_n c_{n-1} c_{n-2} \dots c_0$$

XOR

	a_i	b_i	C_i
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0



Addition (Contd..)

- First step – guessing equivalent to XOR gate.
- The bit c_n is initialized to zero.
- Carry propagation.

Addition (Contd..)

- Carry occurs only when both the bits a_i and b_i are 1 .
- Carry is propagated to the left until both the bits a_j and b_j ($j > i$) are 0 .
- If no such j exists then propagation stops making n^{th} bit 1 .
- $j, j-1, \dots, i+1$ is called the carry block.
- For each carry block $j, j-1, \dots, i+1$ invert the digits c_k
($i+1 \leq k \leq j$)



Algorithm

1. Add antibodies A_i where $a_i = 0$ and $b_i = 0$ or $a_i = 1$ and $b_i = 1$.
2. Add antibodies B_i where $a_i = 0$ and $b_i = 1$ or $a_i = 1$ and $b_i = 0$.
3. For all carry block $j_k j_{k-1} \dots i_k + 1$ do the following in parallel. For $i_k + 1 \leq s \leq j_k$
 - a) Add antibodies T_{ABs} ,
 - b) Add antibodies B_s ,
 - c) Add antibodies T_{BA_s} , and
 - d) Add antibodies A_s .

Example

1011

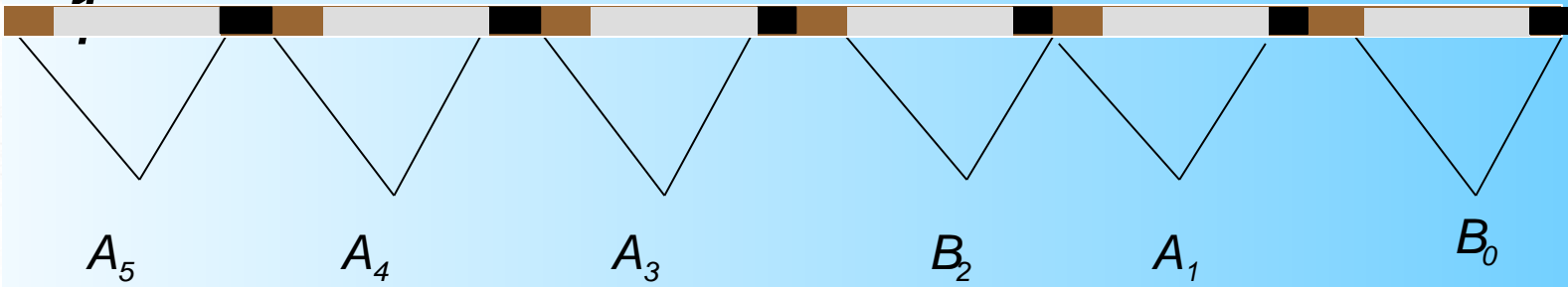
0

+

000101

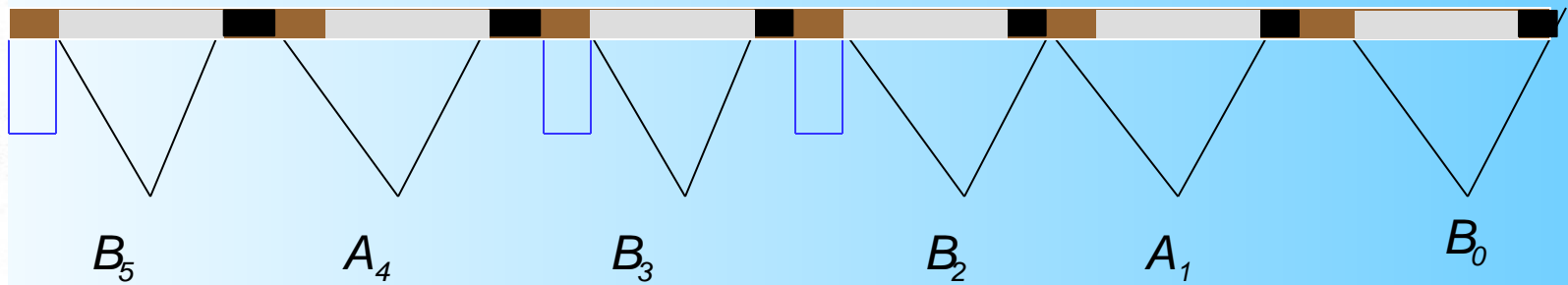
1001

1



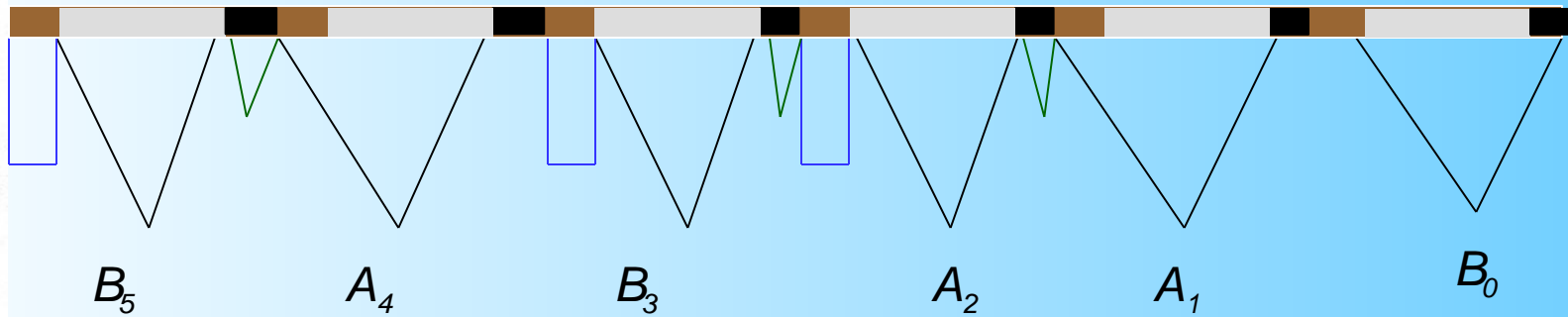
Example (Contd..)

101101



Example (Contd..)

101001



Algorithm

ADD(A,B,C)

1. XOR(A,B,C)
2. BlockInversion(I_1, I_2, \dots, I_k, C) where I_j are carry blocks and k is the number of carry blocks.



Binding-Blocking Automata



Binding-Blocking Automata

- finite control
- finite tape
- tape head
- finite tape symbols
- transition function
- partial order relation
- blocking and unblocking functions

BBA Formal Definition

$$P = (Q, V, E, \delta, q_0, R, \beta_b, \beta_{ub}, Q_{accept}, Q_{reject})$$

where $Q = Q_{block} \cup Q_{unblock} \cup Q_{general}$,

$q_0 \in Q$ (start state),

V is a finite set of symbols,

E is the finite subset of V^* ,

δ is the transition function from $Q \times E \rightarrow Q$,

$R \subseteq E \times E$ is the partial order relation,

β_b is the blocking function from $Q_{block} \rightarrow 2^V$,

β_{ub} is the unblocking function from $Q_{unblock} \rightarrow 2^V$,

$Q_{accept} \cup Q_{reject} \subseteq Q_{general}$ where Q_{accept} is the set of accepting states and Q_{reject} is the set of

rejecting states.




Results in BBA

- Power of the system is analyzed in four variants,
({leftmost, locally leftmost} X {blocked and free transitions})
- Locally leftmost is more powerful.
- The acceptance power of k- NFA is in between the power of BBA in leftmost and locally leftmost.

Complexity Issues

- Blocking number
 - Blocking instant
 - Blocking quotient
- *Infinite hierarchy*
- *Hierarchy collapses*



String Binding- Blocking Automata

- String of symbols (starting from the head's position) can be blocked from being read by the head.
- Only those symbols which are not marked and not blocked can be read by the head.
- Blocking is maximal.

Results in StrBBA

- The power of *StrBBA* in l transition is strictly more than *BBA* in l transition.
- The power of *StrBBA* in ll transition is strictly more than languages not accepted by *BBA* in ll .
- For every $L \in \text{StrBBA}_l$, there exists a random-context grammar *RC* with Context-free rules such that
$$L(\text{RC}) = L.$$
- The set of all languages accepted by $\text{strbba}_l(\text{Fin})$ is equal to the set of all regular languages.



Rewriting BBA

- A 2-way tape head which scans a cell to its right at a time.
- Marking is done with help of a particular set called Marker set
- There is a set of poset relations, where each poset is defined on the marker set.
- This poset relation, depending on the state, helps to replace the markers .




RBBA - Result

RBBA is Universally Complete

- Simulate a Turing machine using a RBBA
- Rewriting of Turing machine is taken care by the markers.

Conclusion

- The peptide computational model has the potential to solve difficult combinatorial problems.
- Will it be a complete computational model or a hybrid version with silicon computers or a model used to solve some subset of problems?
- Defining basic operations for the model.
- Solving hard problems using these basic operations.
- Preprocessing time – time for building peptides and antibodies is more.
- Implementation difficulties.
- Can exhaustive search be replaced by other efficient methods.



“Sciences reach a point where they become mathematized..... It occurred in physics about the time of the Renaissance; it began in chemistry after John Dalton developed atomic theory; and it is just now happening in biology”

“They were built by 3 billion years of evolution, and we’re just beginning to tap their potential to serve non-biological purposes. Nature has given us an incredible toolbox, and we’re starting to explore what we might build.”

Leonard Adleman

Thank You