Introduction to Molecular Computing

Masami Hagiya
Professor, University of Tokyo
Graduate School of Information Science and Technology
Department of Computer Science

What is Molecular Computing?

- Biomolecule = information processing machine
  - Autonomous control of chemical reactions

    ⇒ Encoded in molecules themselves
    - Nanoscale, low energy
    - Massive parallelism
    - Physical and chemical functions of molecules
- Objectives of molecular computing
  - Scientific investigation of computational power of molecules and their reactions
  - Engineering realization of new computational paradigms based on molecular reactions
- References
  - M. Hagiya, T. Yokomori: DNA Computer, Baifukan, 2001.
  - M. Hagiya: Present and Future of Molecular Computer --- Progress towards Molecular Programming, Saiensu-sha, 2004.

Objectives of Molecular Computing

- Analysis of computational power of molecular reactions and Applications:
  - Molecular sensors using molecular computation

    ⇒ Application to biotechnology
  - Programmed self-assembly and molecular machines

    ⇒ Application to nanotechnology
  - Evolutionary computation by molecules

    ⇒ Application to molecular evolution
- New computational paradigms based on molecular reactions

Related Fields
（Biology & Information Technology）

	Raw object	Software
Analysis	Molecular Biology	Bioinformatics ~~Mathematical Biology~~
Synthesis	Molecular Computing Molecular Evolutionary Engineering	Evolutionary Computation Artificial Life

Evolution ⊂ Calculation

Related Fields
(Molecular Sciences)

- Molecular Electronics
  - Electronic circuit using molecular devices (existing computation paradigm)
  - Molecular computing as tecnhologoy for constructing molecular circuits (nanotechnology)
- Nanotechnology
- Supramolecular chemistry
- Quantum computing
- Optical computing
- Molecular biology, biotechnology
- Molecular evolutionary engineering

Hagiya’s wet laboratory

- School of Engineering, Bldg.９, Rm.５０１
- Earthquake-proof reconstruction of Bldg. 9 completed at the end of March
- Experiments resumed in April
- General cleaning of the bldg on April 19


National
ハイビーム電球
150
National
レフ電球
ホワイト
A40 Y007.
ハイビーム電球


TOMY

Introduction to Molecular Computing

Table of Contents:

Introduction to Molecular Computing

Table of Contents:

Analysis of Computational Power of Molecular Reactions

- Various computational models
  - Inter-molecular reactions vs. intra-molecular reactions
  - Liquid-phase vs. solid-phase
  - Test tube, membrane, cell
  - Heteronomous vs. autonomous
- Analysis of computational power

  (of various computational models)
  - Computability
  - Complexity --- time and space
  - Errors and yields --- probabilistic analysis
  - Analysis more faithful to actual molecular reactions

Analysis of Various Computational Models

- Adleman-Lipton
  - Random generation of solution candidates by hybridization of DNA
  - Extraction of solutions by data-parallel computation
  - Suyama --- Dynamic Programming
  - Sakamoto-Hagiya --- SAT Engine
  - Head-Yamamura --- Aqueous Computing
- Seeman-Winfree
  - Self-assembly of various forms of DNA molecules
  - Computation with self-assembly

Analysis of Various Computational Models

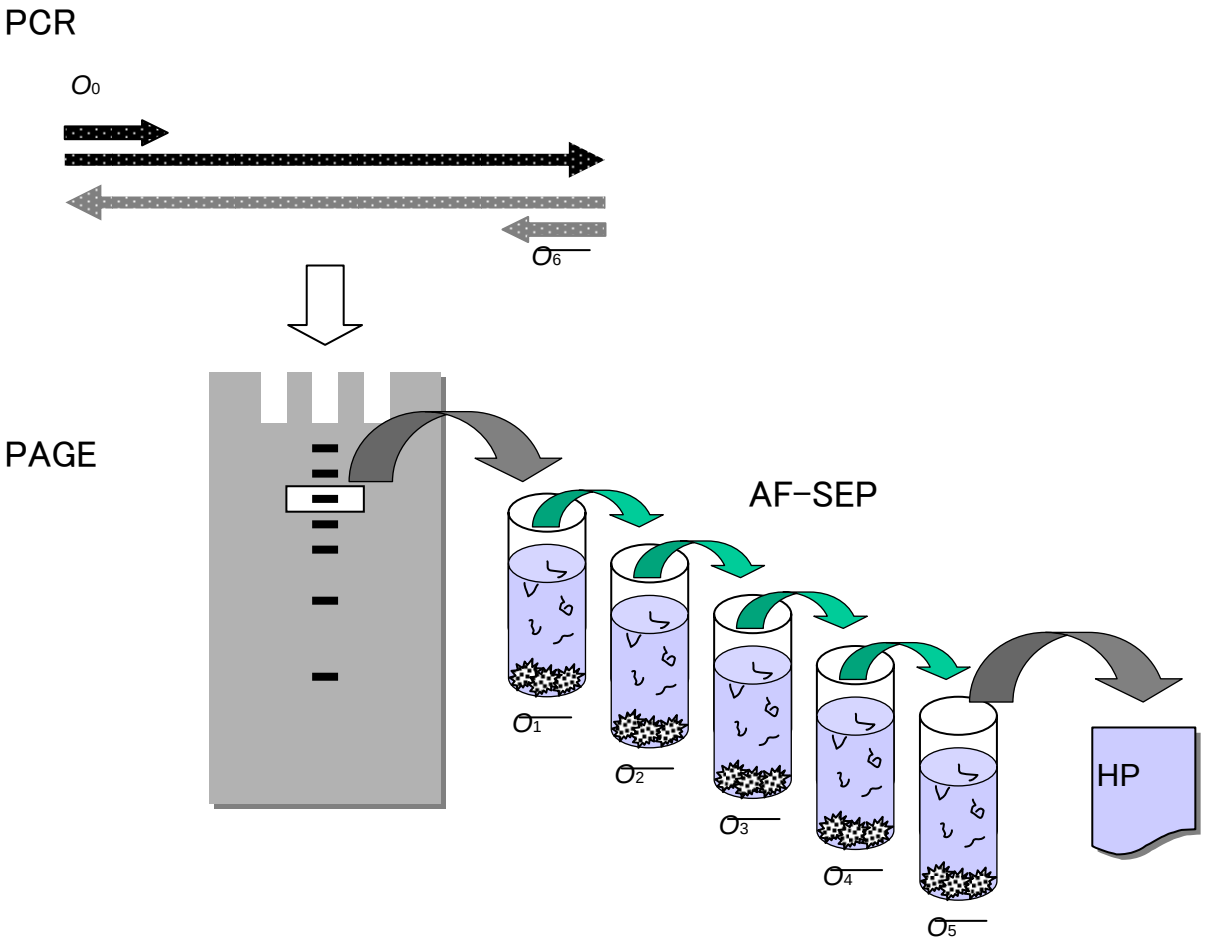
- Head
  - Language generation by gene splicing
- Ogihara-Ray
  - Parallel computation of Boolean circuits
- Hagiya-Sakamoto
  - State machine (Whiplash)
- Shapiro
  - Finite Automaton

} → molecular machines

Adleman-Lipton Paradigm

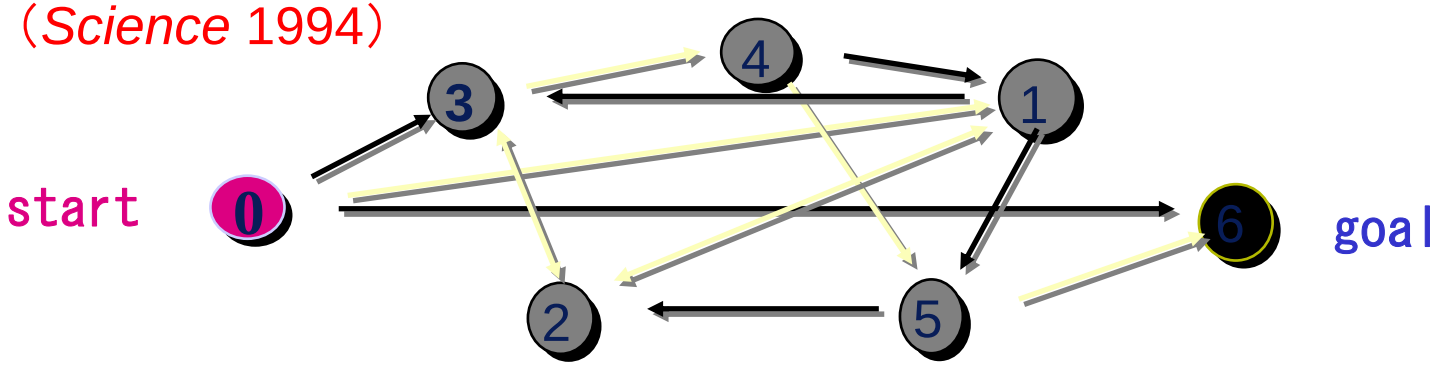
- Adleman (*Science* 1994)
  - Solving Hamilton path problem with DNA
- Lipton, *et al.*
  - Solving SAT problem with DNA
- Massive parallelism using molecules
  - Combinatorial optimization as a main purpose
  - Random generation by DNA self-assembly
    - solution candidate ＝ DNA molecule
  - Extraction of solution using biological experiment
- Currently recognized as a benchmark for biotechnology
  - Vision of the study: application to genomic analysis

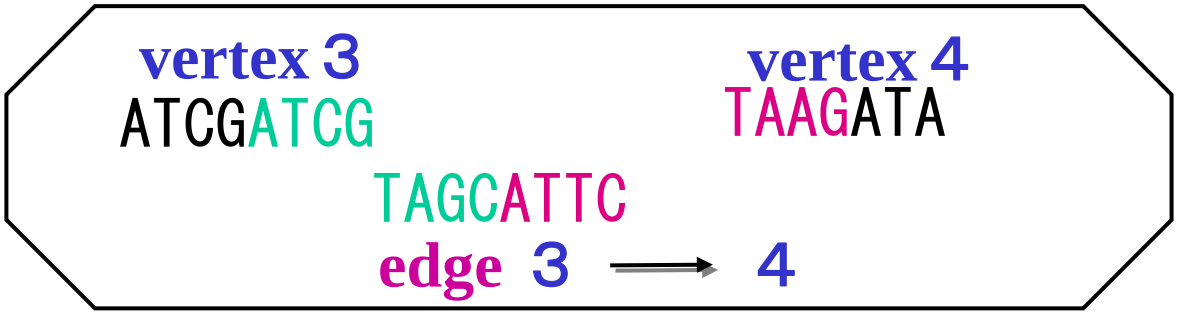
Adleman's First DNA Computer



Experiment by Adleman

（*Science* 1994）



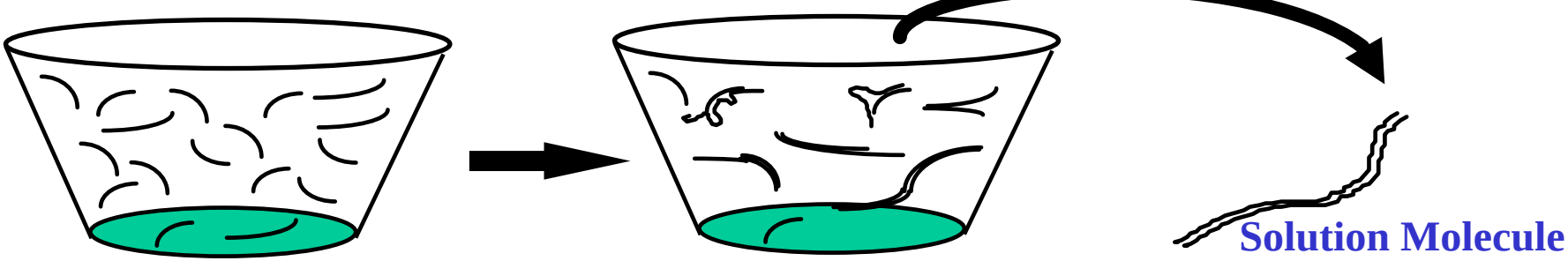


Hybridization

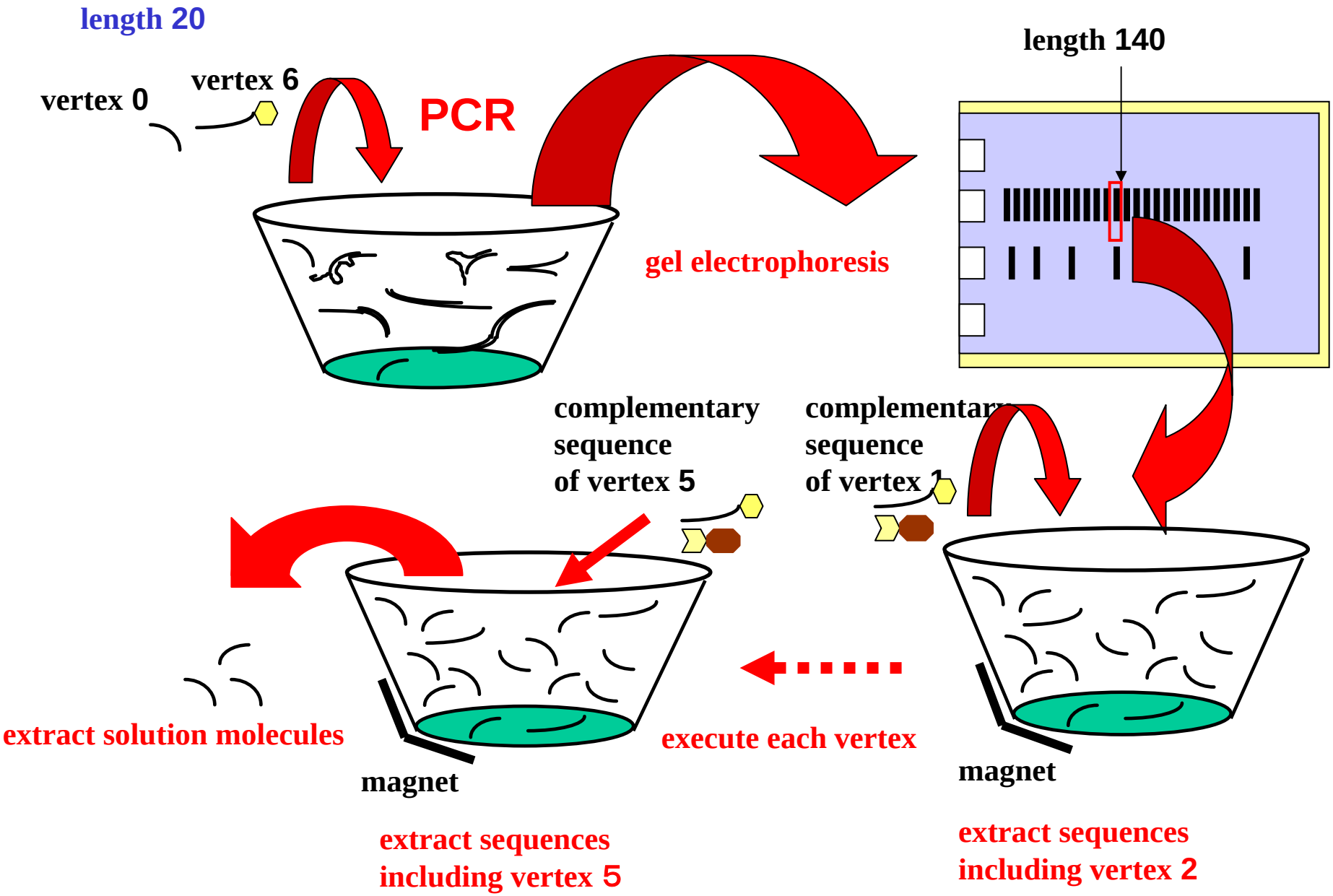
ATCGATCG•TAAGATA

TAGCATTC

Extract・Detect

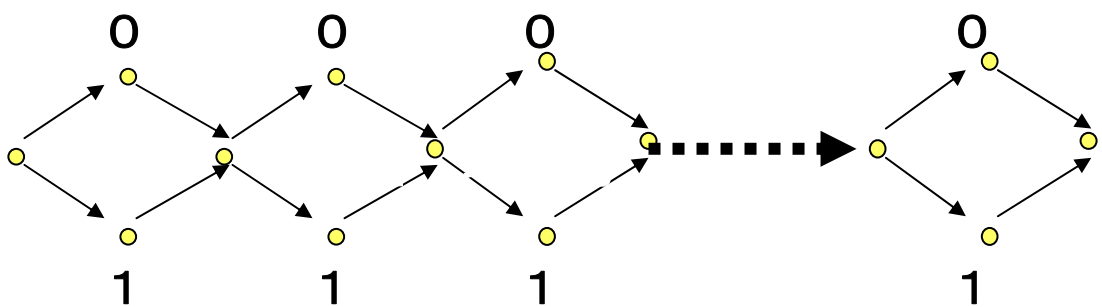


Extraction and Detection of Solutions



Adleman-Lipton Paradigm

generation of all solution candidates



generation of all assignments
(Lipton 1995,
satisfiability problem)

inspection and extraction of solutions

T_i : multiset of strings (test tube)

[Separate] $T_2 = +(T_1, s)$: extract sequences including s

$T_2 = -(T_1, s)$: extract sequences without s

[Merge] $T_3 = T_1 \cup T_2$: merge T_1 and T_2

[Amplify] $(T_2, T_3) = T_1$: copy T_1

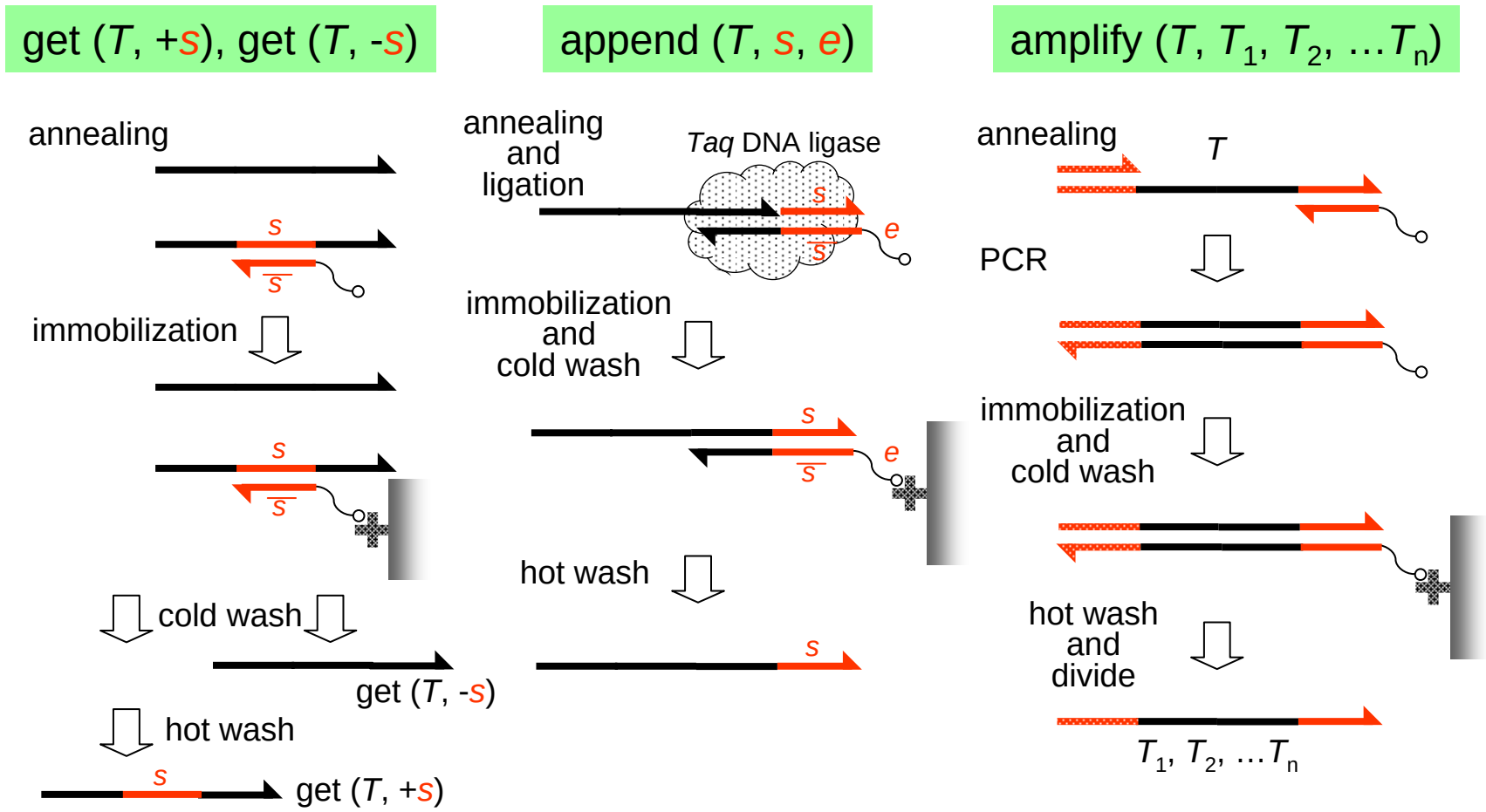
detection of solutions

[Detect]

Suyama's DNA Computer

- "Counting"（Ogihara and Ray）
  - $O(2^{0.4n})$ molecules for n-variable 3-SAT
- "Dynamic programming" (Suyama)
- Iteration of generation and selection
  - Partial generation of solution candidates
  - Selection of solution candidates
- Although both are exponential orders, $O(2^{0.4n})$ is far less than $O(2^n)$
- Solid-phase
  - Affinity separation with magnetic beads
  - Suitable for automation ⇒ Robot !

Implementation of Basic Operations



DP Algorithm for 3CNF-SAT on DNA Computers

function $\text{dna3sat}(u_1, v_1, w_1, \ldots, u_m, v_m, w_m)$
begin
  $T_2 = \{X_1^T X_2^T, X_1^F X_2^T, X_1^T X_2^F, X_1^F X_2^F\}$;
  for $k = 3$ **to** n **do**
    $\text{amplify}(T_{k-1}, T_w^T, T_w^F)$;
    for $j = 1$ **to** m **do**
      if $w_j = x_k$ **then**
        $T_w^F = \text{getuvsat}(T_w^F, u_j, v_j)$;
      end
      if $w_j = \neg x_k$ **then**
        $T_w^T = \text{getuvsat}(T_w^T, u_j, v_j)$;
      end
    end
    $T^T = \text{append}(T_w^T, X_k^T, \overline{X_{k-1}^{T/F} X_k^T})$; $T^F = \text{append}(T_w^F, X_k^F, \overline{X_{k-1}^{T/F} X_k^F})$;
    $T_k = \text{merge}(T^T, T^F)$;
  end
  return $\text{detect}(T_n)$;
end

function $\text{getuvsat}(T, u, v)$
begin
  $T_u^T = \text{get}(T, +X_u^T)$; $T'^F_u = \text{get}(T, -X_u^T)$;
  $T_u^F = \text{get}(T'^F_u, +X_u^F)$; /* *can be omitted* */
  $T_v^T = \text{get}(T_u^F, +X_v^T)$;
  $T^T = \text{merge}(T_u^T, T_v^T)$;
  return T^T;
end

Number of operations

$$(n-2) \times (\text{amplify} + 2 \times \text{append} + \text{merge}) + m \times (3 \times \text{get} + \text{merge})$$

3CNF-SAT Solution on DP DNA Computer

Problem : $4\,\text{variables}, 10\,\text{clauses}$

$$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge$$
$$(\neg x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge$$
$$(x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee \neg x_4) \wedge$$
$$(\neg x_1 \vee x_3 \vee \neg x_4) \wedge (x_2 \vee x_3 \vee x_4) \wedge$$
$$(x_2 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee \neg x_3 \vee x_4)$$

Solution :

YES

$\{X_1^T X_2^T X_3^F X_4^F\}$

DP Algorithm for 3CNF-SAT on DNA Computer

k's loop: k ranges over variable indices

j's loop: j ranges over clause indices

if x_k is the 3rd literal of the j-th clause **then**

remove those assignments which satisfy neither the 1st nor the 2nd literal

append X_k^F to the remaining assignments

(do similarly if $\neg x_k$ is the 3rd literal)

$k = 3$

x_3

$\cancel{X_1^T X_2^T} \longrightarrow X_1^T X_2^T X_3^F$

$\cancel{X_1^F X_2^T}$ $\quad (x_1 \vee \neg x_2 \vee x_3)$

$X_1^T X_2^F \longrightarrow X_1^T X_2^F X_3^F$

$\cancel{X_1^F X_2^F}$ $\quad (x_1 \vee x_2 \vee x_3)$

DP Algorithm for 3CNF-SAT on DNA Computer

k's loop: k ranges over variable indices

j's loop: j ranges over clause indices

if x_k is the 3rd literal of the j-th clause **then**

remove those assignments which satisfy

neither the 1st nor the 2nd literal

append X_k^F to the remaining assignments

(do similarly if $\neg x_k$ is the 3rd literal)

$k = 3$

$\neg x_3$

$\cancel{X_1^T X_2^T}$ $(\neg x_1 \vee \neg x_2 \vee \neg x_3)$

$X_1^F X_2^T \longrightarrow X_1^F X_2^T X_3^T$

$\cancel{X_1^T X_2^F}$ $(x_1 \vee x_2 \vee x_3)$

$X_1^F X_2^F \longrightarrow X_1^F X_2^F X_3^T$

DP Algorithm for 3CNF-SAT on DNA Computer

k's loop: k ranges over variable indices

j's loop: j ranges over clause indices

if x_k is the 3rd literal of the j-th clause **then**

remove those assignments which satisfy neither the 1st nor the 2nd literal

append X_k^F to the remaining assignments

(do similarly if $\neg x_k$ is the 3rd literal)

$k = 4$

x_4

$\cancel{X_1^F X_2^T X_3^T}$ $(\neg x_2 \vee \neg x_3 \vee x_4)$

$\cancel{X_1^F X_2^F X_3^T}$ $(x_2 \vee \neg x_3 \vee x_4)$

$X_1^T X_2^T X_3^F \longrightarrow X_1^T X_2^T X_3^F X_4^F$

$\cancel{X_1^T X_2^F X_3^F}$ $(x_2 \vee x_3 \vee x_4)$

10-variable and 43-clause instance of 3SAT

$$
\begin{aligned}
&(\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee \neg x_3 \vee \neg x_4) \\
&\wedge (x_1 \vee x_4 \vee x_5) \wedge (x_2 \vee x_3 \vee \neg x_5) \wedge (\neg x_2 \vee \neg x_3 \vee \neg x_5) \\
&\wedge (\neg x_1 \vee \neg x_3 \vee \neg x_5) \wedge (\neg x_2 \vee \neg x_4 \vee x_6) \wedge (\neg x_2 \vee x_3 \vee x_6) \\
&\wedge (x_2 \vee \neg x_3 \vee x_6) \wedge (\neg x_1 \vee \neg x_5 \vee \neg x_6) \wedge (x_2 \vee \neg x_6 \vee x_7) \\
&\wedge (x_1 \vee x_5 \vee x_7) \wedge (\neg x_1 \vee \neg x_5 \vee \neg x_7) \wedge (x_5 \vee \neg x_6 \vee \neg x_7) \\
&\wedge (x_1 \vee \neg x_2 \vee \neg x_7) \wedge (x_1 \vee x_6 \vee \neg x_7) \wedge (\neg x_4 \vee x_6 \vee \neg x_7) \\
&\wedge (x_1 \vee x_4 \vee x_8) \wedge (\neg x_1 \vee x_5 \vee x_8) \wedge (x_2 \vee \neg x_3 \vee x_8) \\
&\wedge (x_1 \vee x_6 \vee x_8) \wedge (x_2 \vee x_5 \vee \neg x_8) \wedge (x_1 \vee x_4 \vee \neg x_8) \\
&\wedge (\neg x_3 \vee \neg x_5 \vee \neg x_8) \wedge (\neg x_2 \vee x_4 \vee x_9) \wedge (x_4 \vee x_7 \vee x_9) \\
&\wedge (x_1 \vee x_7 \vee x_9) \wedge (\neg x_4 \vee x_6 \vee \neg x_9) \wedge (\neg x_1 \vee x_3 \vee \neg x_9) \\
&\wedge (\neg x_2 \vee x_3 \vee \neg x_9) \wedge (x_1 \vee \neg x_7 \vee \neg x_9) \wedge (\neg x_2 \vee x_4 \vee \neg x_9) \\
&\wedge (\neg x_1 \vee x_5 \vee \neg x_9) \wedge (\neg x_4 \vee x_8 \vee x_{10}) \wedge (x_3 \vee \neg x_6 \vee x_{10}) \\
&\wedge (\neg x_2 \vee \neg x_7 \vee x_{10}) \wedge (x_3 \vee \neg x_4 \vee \neg x_{10}) \wedge (\neg x_5 \vee \neg x_6 \vee \neg x_{10}) \\
&\wedge (x_4 \vee x_5 \vee \neg x_{10}) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_{10}) \wedge (x_2 \vee x_8 \vee \neg x_{10}) \\
&\wedge (\neg x_1 \vee x_8 \vee \neg x_{10})
\end{aligned}
$$

DNA Computer Robot based on MAGTRATION™ (Prototype No.1)

Programming DNA Computer

function dna3sat$(u_1, v_1, w_1, \ldots, u_m, v_m, w_m)$
begin
  $T_2 = \{X_1^T X_2^T, X_1^F X_2^T, X_1^T X_2^F, X_1^F X_2^F\};$
  for $k = 3$ **to** n **do**
    amplify$(T_{k-1}, T_w^T, T_w^F);$
    for $j = 1$ **to** m **do**
      if $w_j = x_k$ **then**
        $T_w^F = \text{getuvsat}(T_w^F, u_j, v_j);$
      end
      if $w_j = \neg x_k$ **then**
        $T_w^T = \text{getuvsat}(T_w^T, u_j, v_j);$
      end
    end
    $T^T = \text{append}(T_w^T, X_k^T, \overline{X_{k-1}^{T/F} X_k^T});$
    $T^F = \text{append}(T_w^F, X_k^F, \overline{X_{k-1}^{T/F} X_k^F});$
    $T_k = \text{merge}(T^T, T^F);$
  end
  return detect$(T_n);$
end

Pascal/C-level

```
[Instrument]
[Reset Counter] 0
[Home Position] 0
[MJ-Open Lid]
...
[Get1(0)]
[Get2(1)]
[Append(2)]
...
[Exit]
```

protocol-level

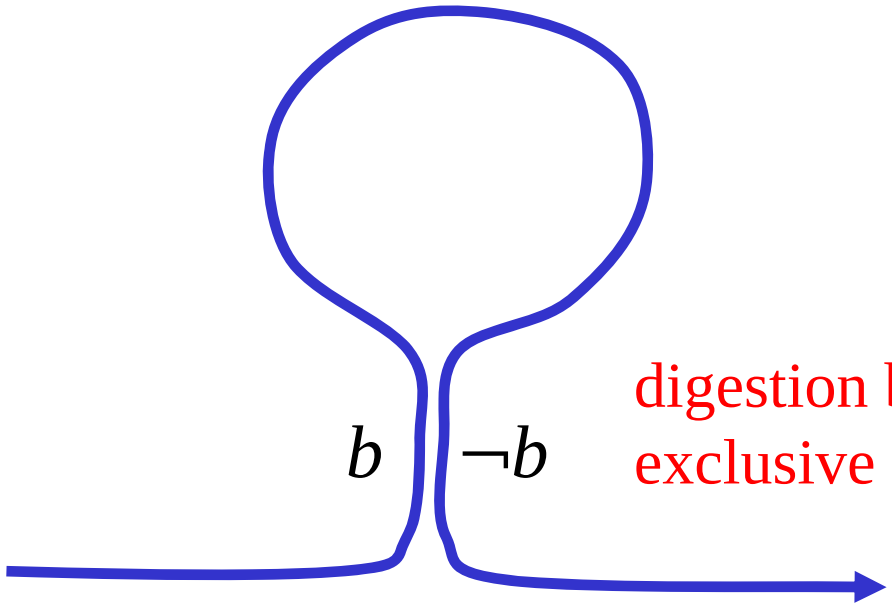
```
(1-1-4)    [MJ-Open Lid]
Do 2
  _SEND    "LID OPEN"
  Do       10
    _SEND "LID?"
    Wait_msec        500
    _CMP_GSTR        "OPEN"
    IF_Goto  EQ  0   ;open
    Wait_msec  1000
  Loop
Loop
; Time out
End
;open
```

script-level

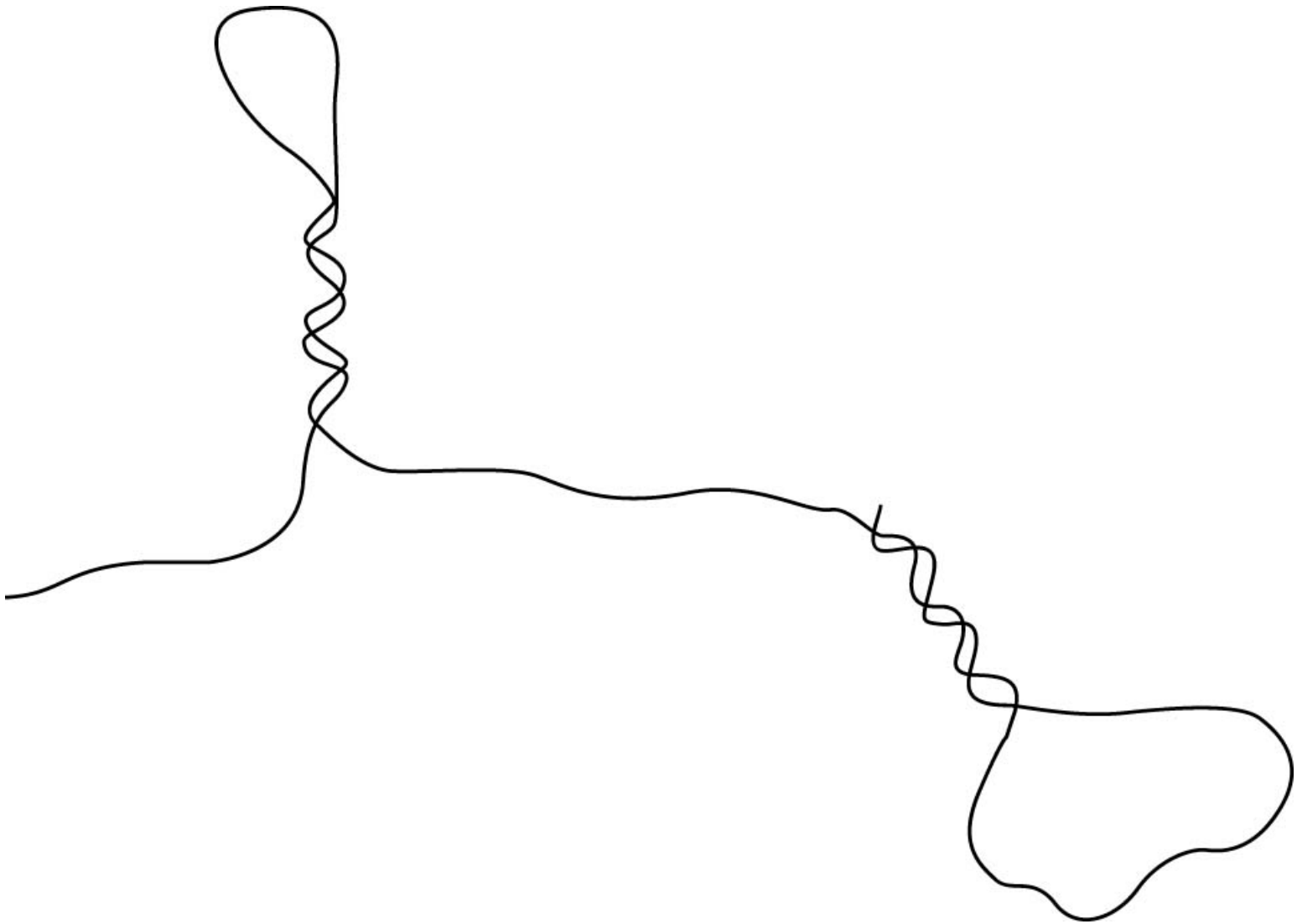
Hairpin Engine（SAT Engine）

- Sakamoto *et al.*, *Science*, May 19, 2000.
- Selection by DNA hairpin structures
  - Digestion by restriction enzyme
  - Exclusive PCR
- 3-SAT
  - Single-stranded DNA comprised of literals each selected from a clause
  - Complementary literal ＝ complementary sequence
  - Detection of inconsistency ⇒ hairpin
  - 6-variable 10-clause 3-SAT problem
- The essential part of SAT computation = hairpin formation
  - Number of steps is independent of the number of clauses/variables
  - Autonomous molecular computation

$$(a \vee b \vee c) \wedge (\neg d \vee e \vee \neg f) \wedge \ldots \wedge (\neg c \vee \neg b \vee a) \wedge \ldots$$



digestion by restriction enzyme
exclusive PCR



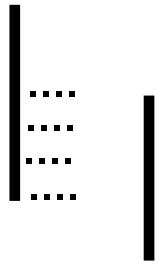
Selection by Hairpin Structures

- Restriction enzyme digestion
  - Hairpins are cut at the restriction site inserted in each literal sequence
- Exclusive PCR
  - PCR is inefficient for hairpins
  - In exclusive PCR, solution is diluted in each cycle to keep the difference in amplification
- Number of steps is independent of the number of variables/clauses

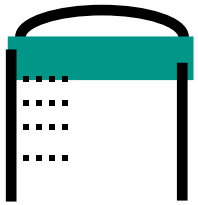
Current Consensus on Adleman-Lipton Paradigm

- Far from outperforming electronic computers
  - Scale-up problem
- Important as a first proof that;
  - Molecules can really compute
- At least serves as a benchmark for biotechnology
- Application to genomic analysis (Suyama)

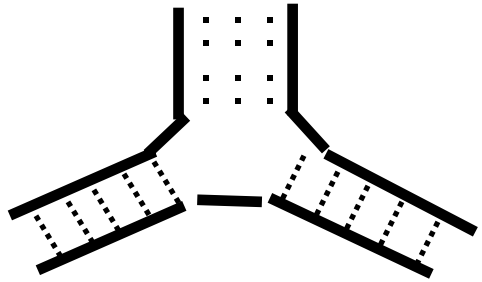
Seeman-Winfree's Computation by DNA Self-Assembly



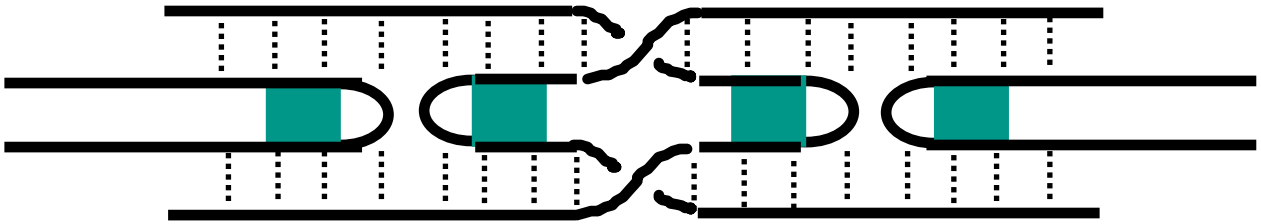
Linear



Hairpin



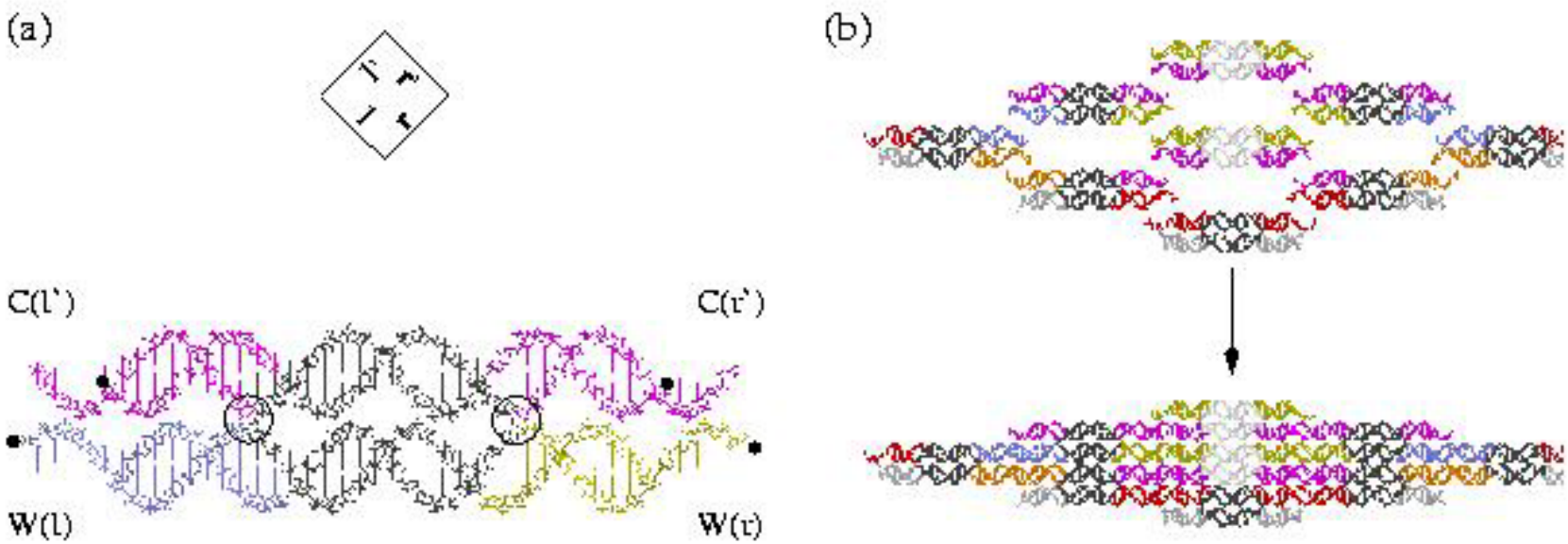
3-Junction



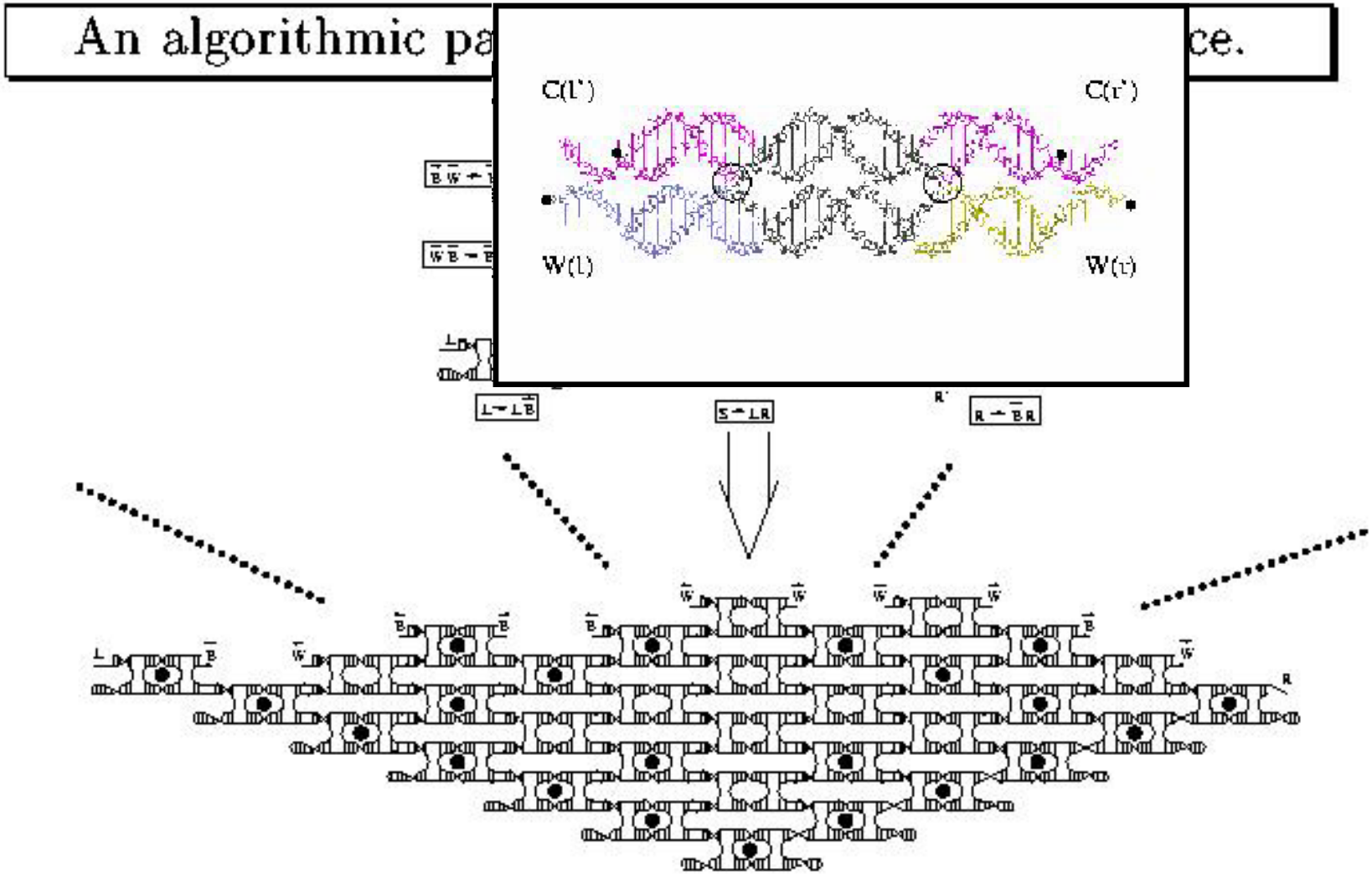
DX (Double Crossover)

Various DNA Molecular Structures

The DNA representation of Wang tiles.



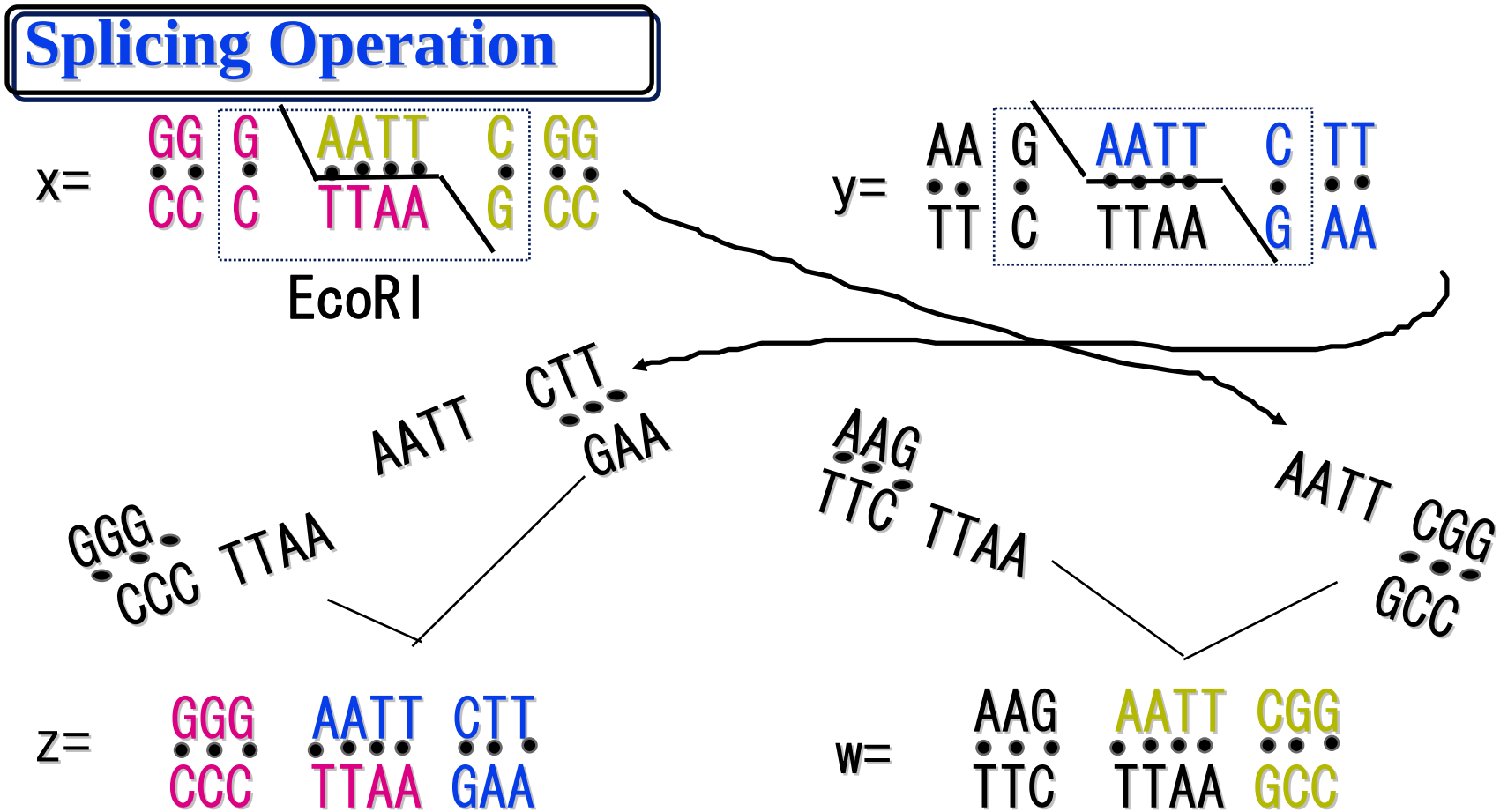
Winfree's Tiling



Sierpinski's Triangle

Head's Computation by Gene Splicing

- Theoretical model of gene splicing with restriction enzyme and ligase（Splicing Model）



Language Generation by Splicing

- Splicing rule: $r = u_1\$u_2\#u_3\$u_4$
- $(x_1u_1u_2x_2, y_1u_3u_4y_2) \vdash_r (x_1u_1u_4y_2, y_1u_3u_2x_2)$
- R: Set of splicing rules
- A: Set of strings (axiom)
- L: Language generated from R & A
  - If $x \in A$ Then $x \in L$
  - If $x, y \in L$ and $r \in R$ and $(x, y) \vdash_r (z,w)$
    Then $z, w \in L$
- If R and A are finite, then L is regular

Introduction to Molecular Computing

Table of Contents:

Computability in Molecular Computing

- Computability of DNA self-assembly
  - Winfree's results
- Computability of gene splicing
  - Various extensions of splicing model

Winfree's Results on Computability of DNA Self-Assembly

- Language generated by linear molecules

  = regular

- Language generated by

  linear + hairpin + 3-junction molecules

  = context-free

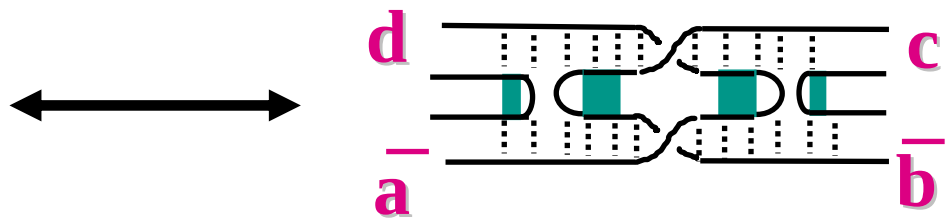
- Language generated by linear + DX molecules

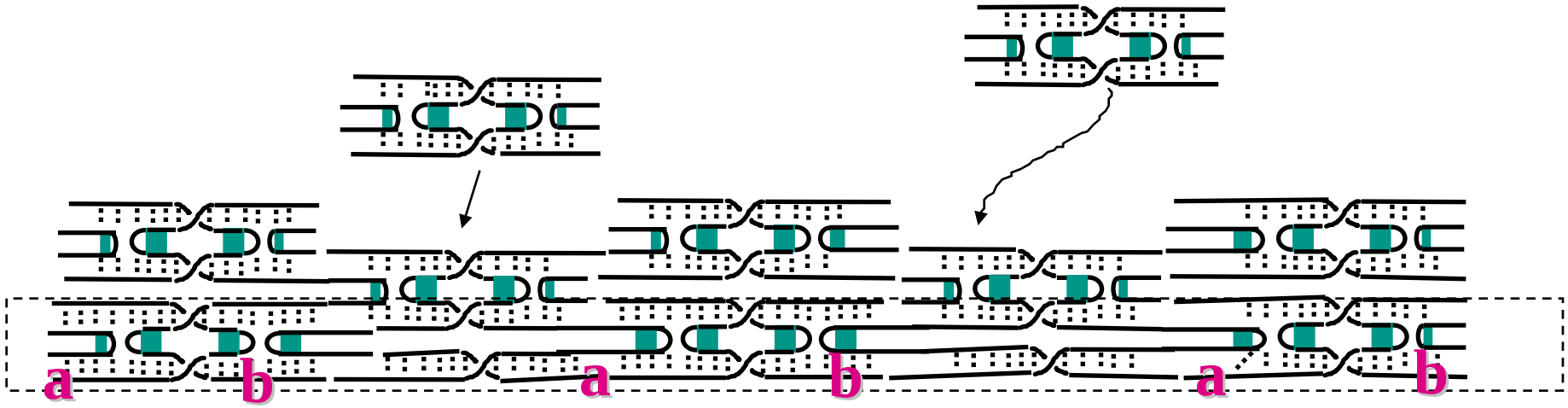
  = recursively enumerable

  = Turing computable

Winfree's Model:
Example of Computation Process

$c = f(a,b)$
$d = g(a,b)$





Initial Configuration

Imitation of 1-dimensional Cellular Automaton

Extentions of Splicing Model

Generative ability of splicing model $<$ Regular language

(splicing) $+ \alpha$? = Universal computational power

$+ \alpha$:	circular molecules
	multiple test tubes
	time-dependent rules etc.

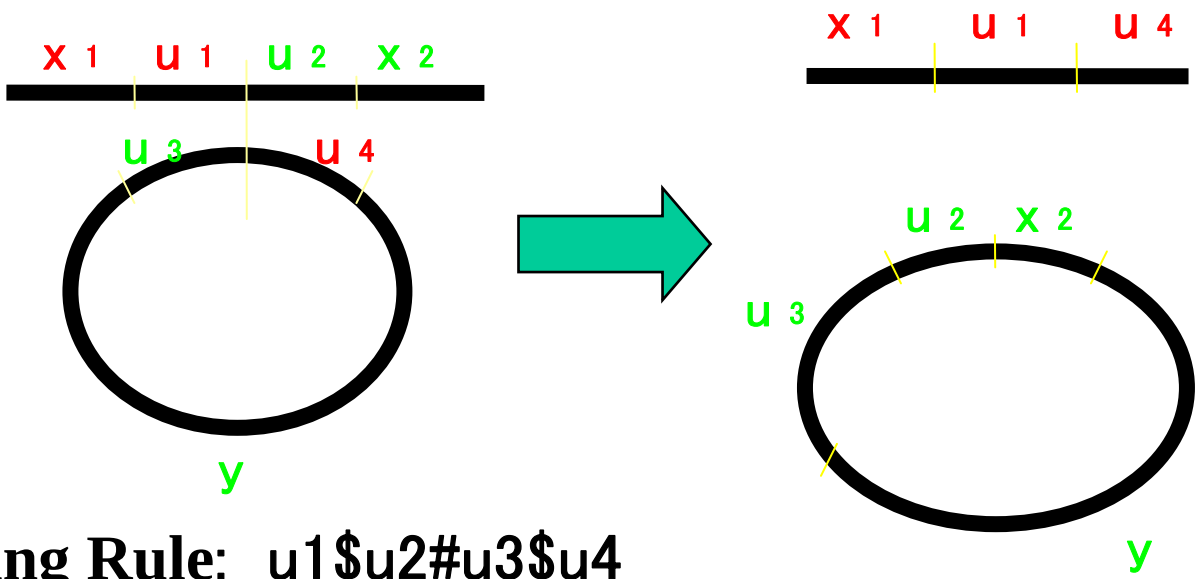
Circular Splicing System

+α:

Allows to use circular strings（circular DNA）

Allows to distinguish between terminal and non-terminal symbols

（ e.g. splicing of colon bacillus chromosome and F plasmid）



Splicing Rule: u1$u2#u3$u4

Introduction to Molecular Computing

Table of Contents:

Complexity of Molecular Computation

- Time
  - Number of experiment steps
  - Time required for each operation
    - Essential for the analysis of computational power of molecules
- Space (= degree of parallelism)
  - Number of molecules
    - Maximum
    - Total
  - Size of molecules (length)
- Trade-off analysis -- important

Complexity Analysis (Adleman-Lipton)

- Reif (SPAA'95)
  - A nondeterministic Turing machine computation with input size n, space s and time $2^{O(s)}$ can be executed in our PAM Model using $O(s)$ PA-Match steps and $O(s \log s)$ other PAM steps, employing aggregates of length $O(s)$.
- Beaver (DNA1, 1995)
  - Polynomial-step molecular computers compute PSPACE.
- Rooß and Wagner (I&C, 1996)
  - Exactly the problems in $P^{NP}=\Delta^{P}_{2}$ can be solved in polynomial time using Lipton's model.

Rooß and Wagner (I&C, 1996)

- Exactly the problems in $P^{NP}=\Delta^P_2$ can be solved in polynomial time using Lipton's model.
- BIO({UN,BX,IN},{EM})-P = $P^{NP}= \Delta^P_2$
  - UN: union (merge) $T_3 = T_1 \cup T_2$
  - BX: bit extraction (separate)

    $T_2 = +(T_1, s)$ $T_2 = -(T_1, s)$
  - IN: initialization (random generation)
  - EM: emptiness test (detect)
  - -P: polynomial time
  - P^{NP}: polynomial time with NP-oracle

Complexity Analysis

- Rothemund and Winfree (STOC 2000)
  - For any $f(N)$ non-decreasing unbounded computable functions, the number of tiles required for the self-assembly of an $N \times N$ square is bounded infinitely often by $f(N)$.
- Winfree, Eng and Rozenberg (DNA6, 2000)
  - Linear assembly of string tiles can generate the output languages of finite-visit Turing Machines.

Errors and Yields of Reactions

- Yields
  - Equilibrium --- equilibrium constant (K)
  - Time to reach equilibrium --- reaction rate (k)
  - Example: A $\leftrightarrow$ B

$$[B] = (K/(1+K))(1-e^{-(k+k_{-1})t})$$

$$K = k/k_{-1}$$

- Errors
  - Example: mis-hybridization
  - Error probability is never 0
- Probabilistic analysis

Probabilistic Analysis

- Karp, Keynon and Waarts (SODA'96)
  - The number of extract operations required for achieving error-resilient bit evaluation is $\Theta(\lceil \log_{\varepsilon} \delta \rceil \times \lceil \log_{\gamma} \delta \rceil)$.
- Kurtz (DNA2, 1996)
  - Thermodynamical analysis of path formation in Adleman's experiment
  - Time needed to form a Hamiltonian path --- $\Omega(n^2)$
- Winfree (1998, Ph.D. Thesis)
  - Thermodynamical analysis of DNA Tiling
- Rose, *et al.* (GECCO'99, *etc.*)
  - Computational incoherency

    (Thermodynamical analysis of mis-hybridization)

Introduction to Molecular Computing

Table of Contents:

Computational Aspects of Molecular Systems Design

- Molecular programming
- Design of molecules
  - Design of DNA = sequence design
  - Structure ⇒ sequence (inverse folding)
  - Patterns for self-assembly
  - Design of molecular machines
- Design of reactions
  - Adjustment of reaction conditions
  - Scheduling of experimental operations
  - Simulation tool
- Molecular machines
  - One of the current objectives of molecular programming

Introduction to Molecular Computing

Table of Contents:

Sequence Design

- Evaluation of sequence set
  - Avoiding mis-hybridization
    - Hamming distance
    - Energy calculation ⇒ mfold (Zuker), Vienna Package
  - Uniform T_m (melting temperature)
- Searching for sequence set
  - Genetic algorithm
  - Coding theory --- Arita's template method
- Inverse problem
  - Structure ⇒ sequence (inverse folding)
  - Vienna group

Template Method

- Arita and Kobayashi, 2002

Same positioning of [AT] or [GC] in every sequence

(= “template”)

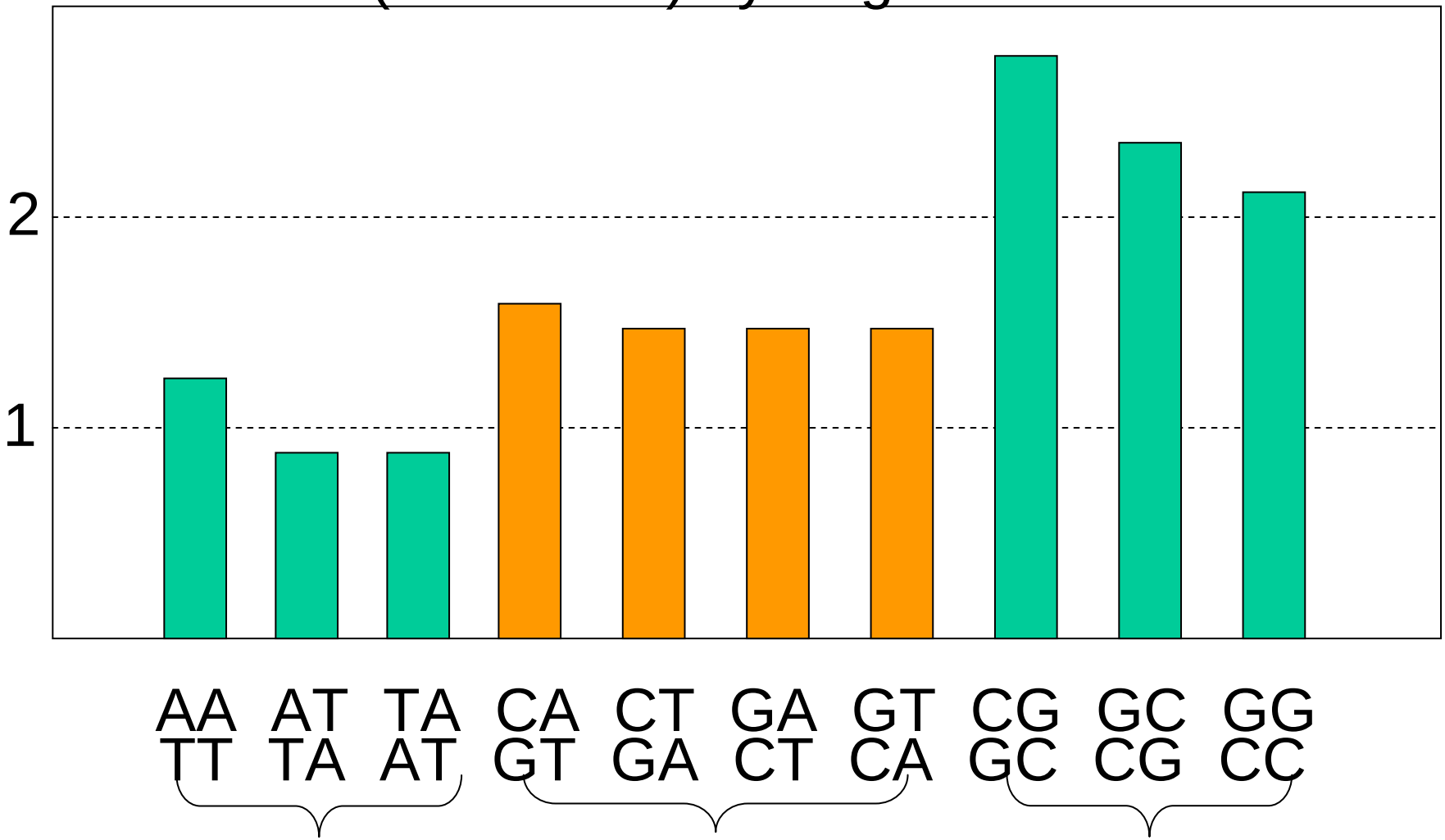
e.g. from 011010

ACCTGA, TGCTCA, TCGACA, etc.

→ Melting temperature of every sequence will be the same

Stacking Energy

- ΔG kcal/mol (DNA/DNA) by Sugimoto et al.



Template Including Mismatches

- Proper selection of template ensures mismatch(es) even with shift / reverse

e.g. when 110100

```
110100        110100        110100 110100
 110100         110100          110100
```

Includes at least 2 mismatches
even with any shifting or concatenatoin

Template Selection

- Selecting template T which will include minimum of (d) mismatches in each of the following patterns

  - T^R
  - TT^R , T^RT
  - TT , T^RT^R

T^R: reversed sequence of T

When $T=110100$, $T^R=001011$

Examples of Templates

- Length 6 (2 mismatches)

  110100 （of 2^6）

- Length 11 (4 mismatches)

  01110100100, 01011100010, 11000100101 （of 2^{11}）

- Length 23 (9 mismatches）

  01111010110011001010000, 10110011001010000011110, 11100000101001100110101 （of 2^{23}）

Design of DNA Sequence

"Template + Error Correcting Code"

1 1 0 1 0 0 (template)
+ 0 1 0 0 1 1 (any code)

A T C A G G (DNA sequence)

Any Error Correcting Code can be used

1. BCH Code
2. Golay Code
3. Hamming Code etc.

Inverse Folding

- Vienna group
- Using McCaskill's algorithm
- Sequence search by minimization of cost function

$$\Xi(x) = E(x, \Omega) - G(x) = -RT \ln p$$

– Ω : Target structure

– x: Sequence

– $E(x, \Omega)$: Free energy of Ω at x

– $G(x)$: ensemble free energy of sequence x (McCaskill)

– p: Probability of Ω at x

Introduction to Molecular Computing

Table of Contents:

Design of Molecular Reactions

- Condition of reactions
  - Temperature
  - Salt concentration
  - Time
- Operation scheduling
- Simulation
  - e-PCR
    - http://www.ncbi.nlm.nih.gov/genome/sts/epcr.cgi
  - VNA

VNA: Simulator for Virtual DNA

- Abstract, but sufficiently physical

  Bridging the gap between abstract models and actual reactions

  molecule — hybrid of virtual strands

```
abcd
  ||
CDEF
```

- Reactions
  - hybridization
  - denaturation
  - restriction
  - ligation
  - self-hybridization
  - extension

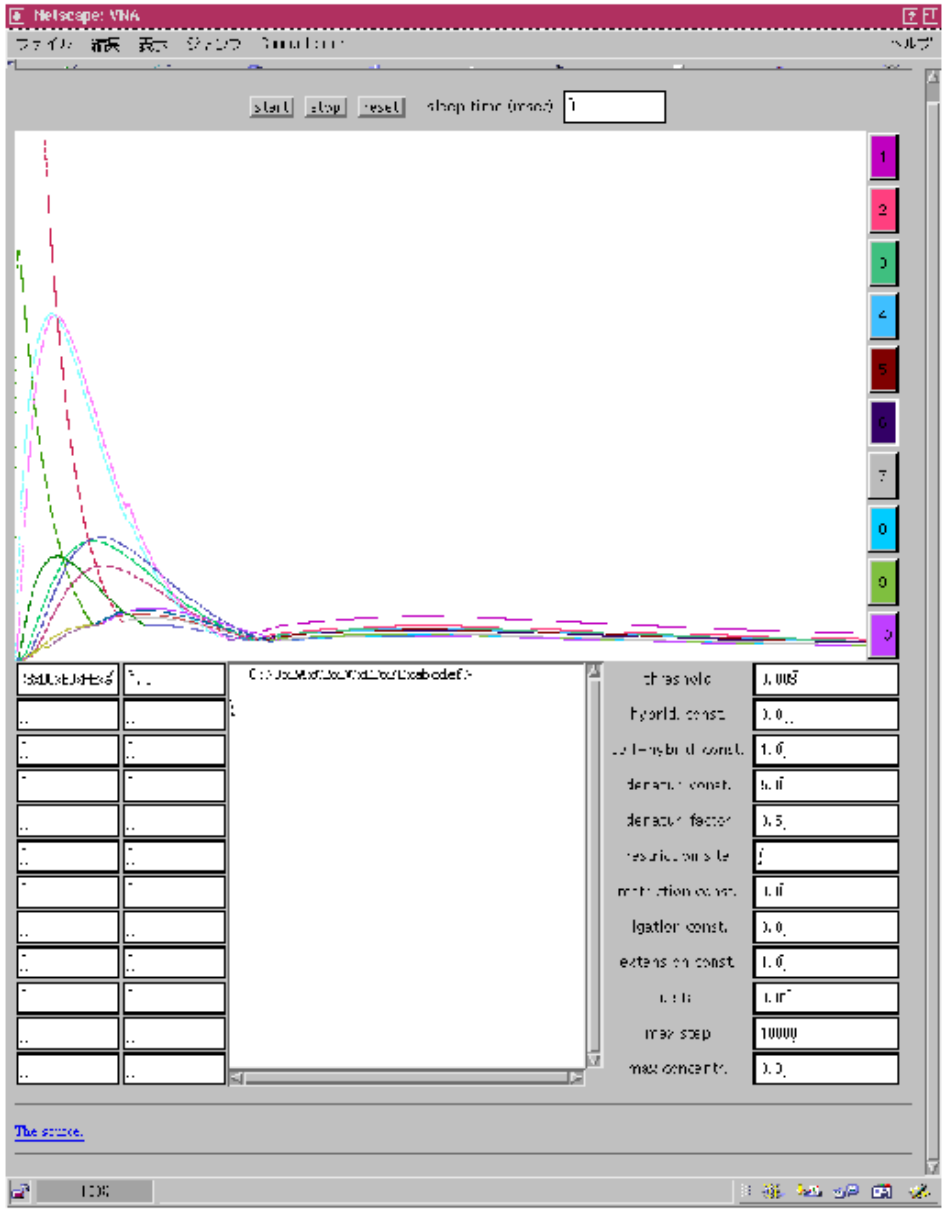
VNA (cont)

- Objectives
  - Verifying feasibility of algorithms for DNA computation
  - Verifying validity of molecular biology experiments (e.g., PCR experiments)
  - Parameter fitting in molecular biology experiments
- Examples
  - Ogihara and Ray's computation of Boolean circuits
  - Winfree's construction of double-crossover units
  - PCR experiments
- Implementation
  - Java ⇒ executable as an applet

VNA (cont)

- Methods
  - Combinatorial enumeration
  - Continuous simulation (diff. eq.)

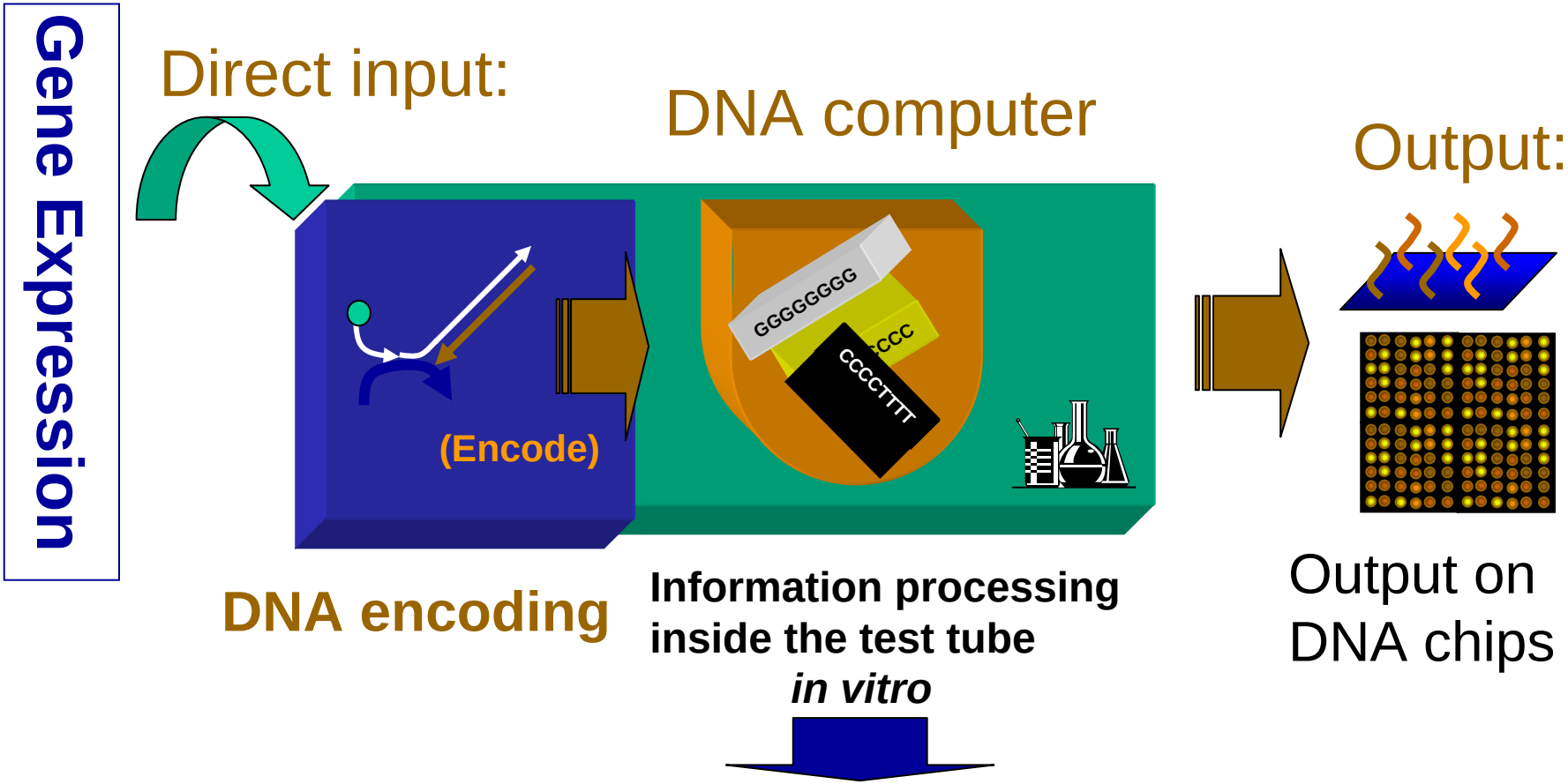
) unify
- Avoiding combinational explosion
- Contributions in simulation technology
  - Threshold
  - Stochastic
- Parameter fitting by GA
  - Optimizing amplification in PCR experiments


Netscape: VNA
start
stop
reset
max step
10000
The source.

Introduction to Molecular Computing

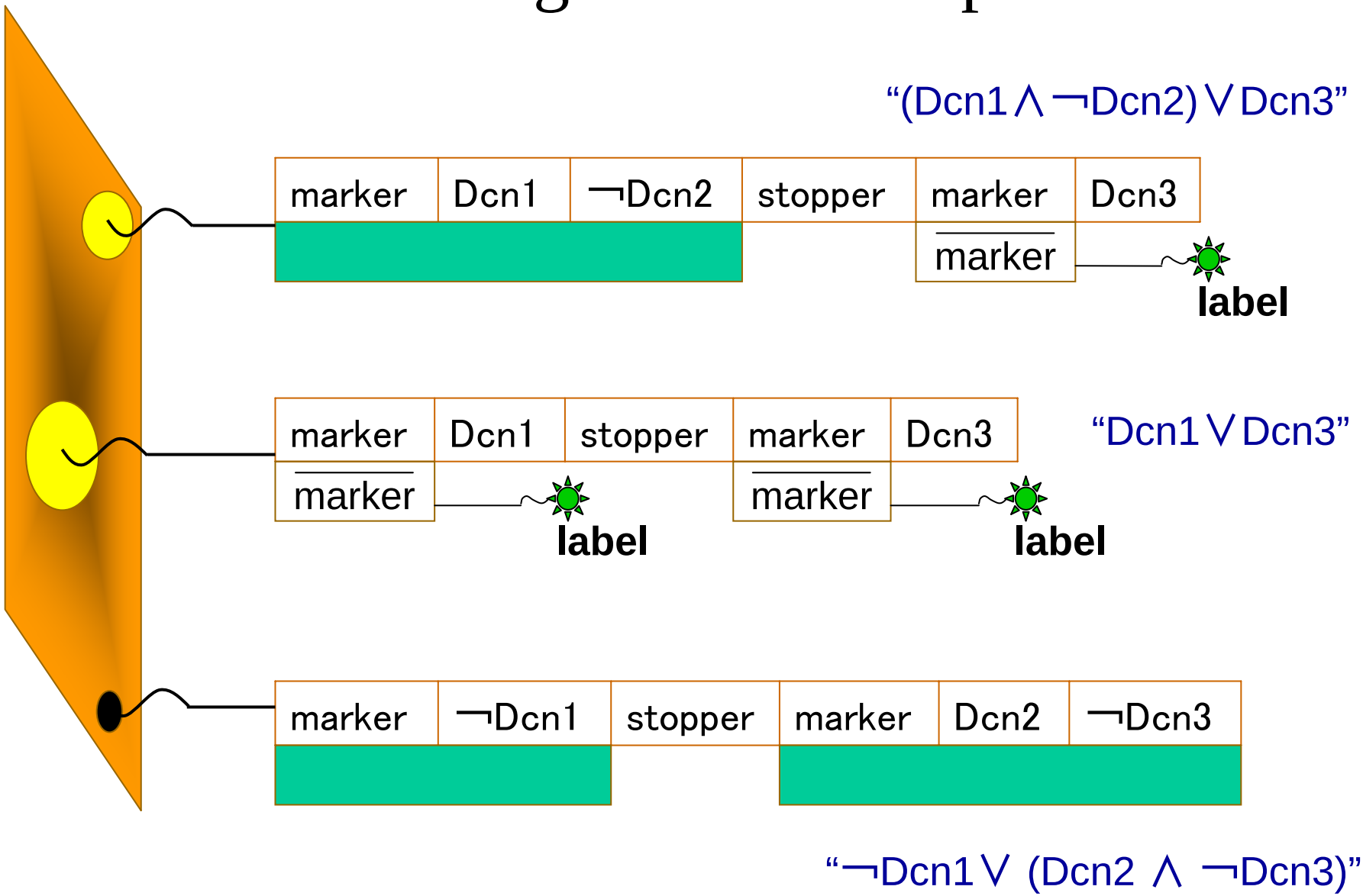
Table of Contents:

Information Processing of Gene Expression Analysis Using DNA Computation



- **Direct input of DNA molecule**
- **Massive parallelism**

Intelligent DNA Chip



Introduction to Molecular Computing

Table of Contents:

DNA Nanotechnology
DNA Self-Assembly

- DNA lattice
- DNA as a connector of molecules
  - Self-assembly of nanoparticles using DNA
  - Self-assembly of nanowires using DNA
- DNA tile
  - Structure formation by DNA itself
- Programmed self-assembly

DNA-Based Self-Assembly of Nanoparticles Early Studies

- C. A. Mirkin *et al.*

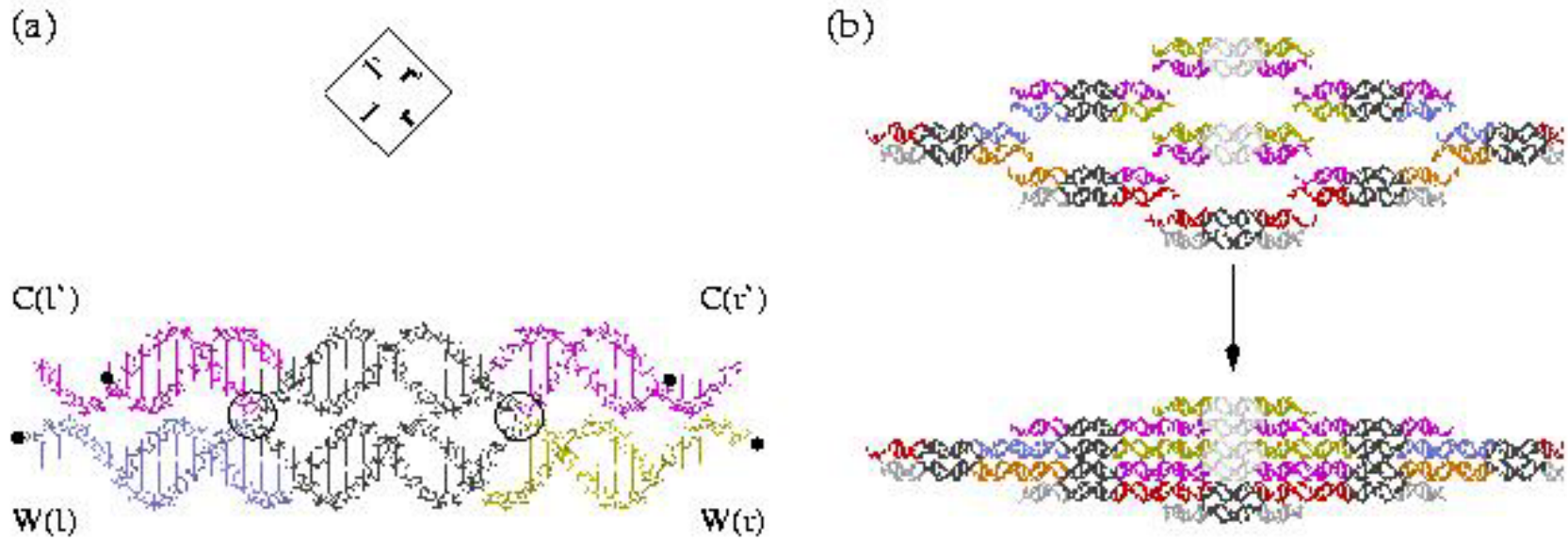
  DNA-based method for rationally assembling nanoparticles into macroscopic materials. *Nature* **382**, 607-609 (1996)

- A. P. Alivisatos *et al.*

  Organization of ‘nanocrystal molecules’ using DNA. *Nature* **382**, 609-611 (1996)

Winfree-Seeman's DNA Tiles (double crossover molecules)

The DNA representation of Wang tiles.



Introduction to Molecular Computing

Table of Contents:

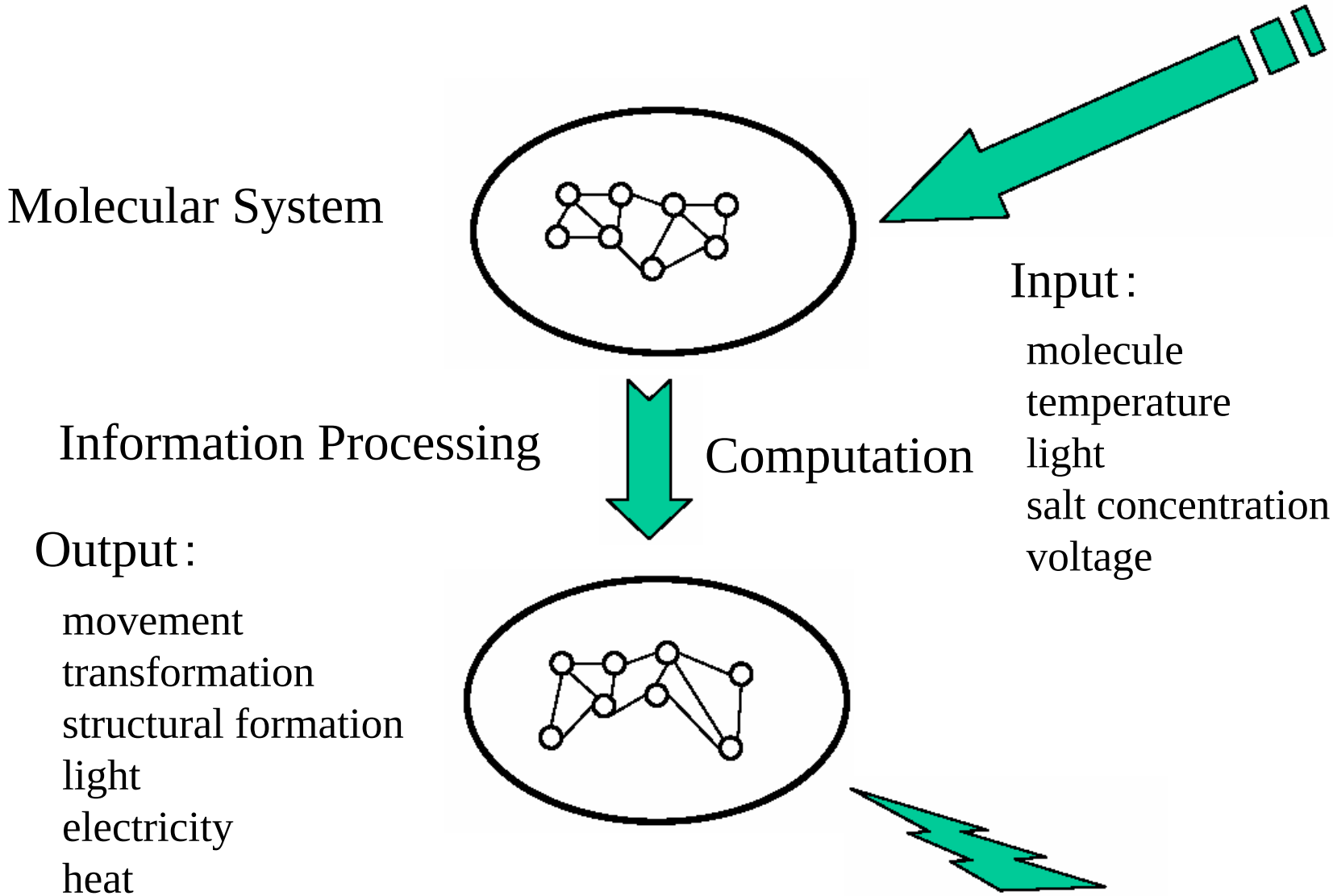
Molecular Machines

- Machines as Actuators
  - Motor
  - Transporter

- Abstract Machines --- Finite State Machines (Automaton)
  - Have a finite number of states
  - Change their state autonomously or according to inputs
  - May produce outputs
  - Are the first step towards general-purpose computers
  - Have many kinds of applications
    - Switch
    - Memory (both holding contents and addressing)

- The difference between the two is still unclear

Molecular System Consisting of Finite State Machines

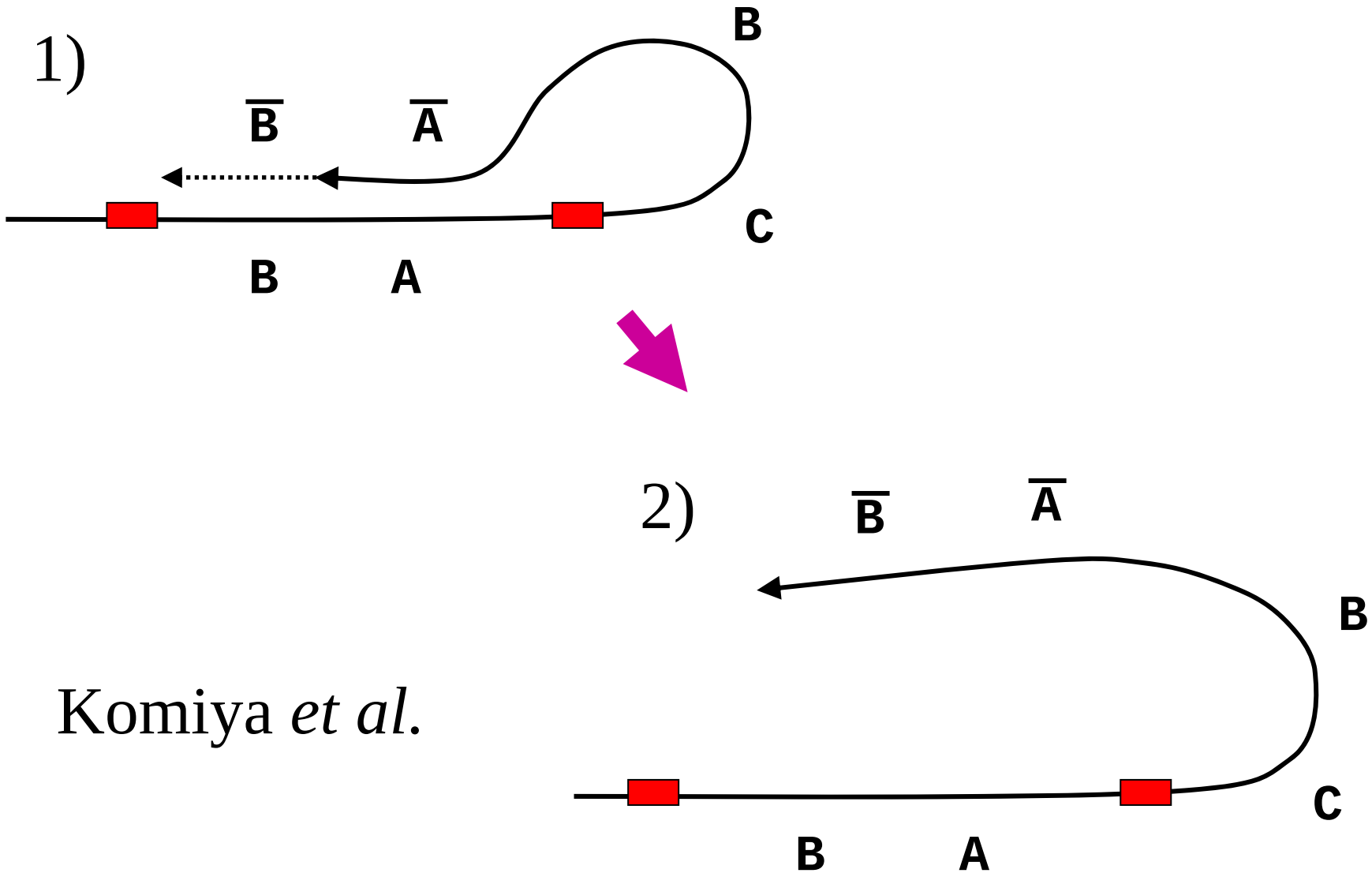


Molecular (DNA) State Machines

- Terminal-sequence machines
  - The terminal sequence encodes the state
  - Our whiplash machine
    - Gets longer as it changes the state
  - Shapiro's automaton
    - Gets shorter as it changes the state
- Conformational machines
  - The state is encoded as a structure
  - Yurk's molecular tweezers
  - Seeman's PX-JX_2 Switch
  - Our hairpin-based machine…

Whiplash PCR (WPCR)

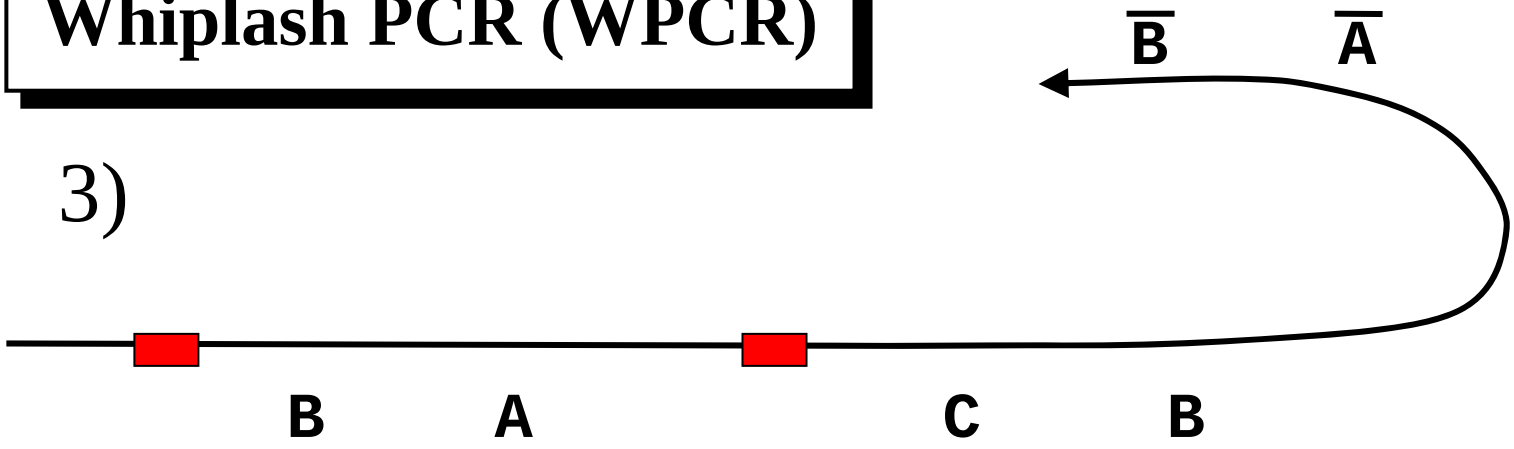
: stopper sequence



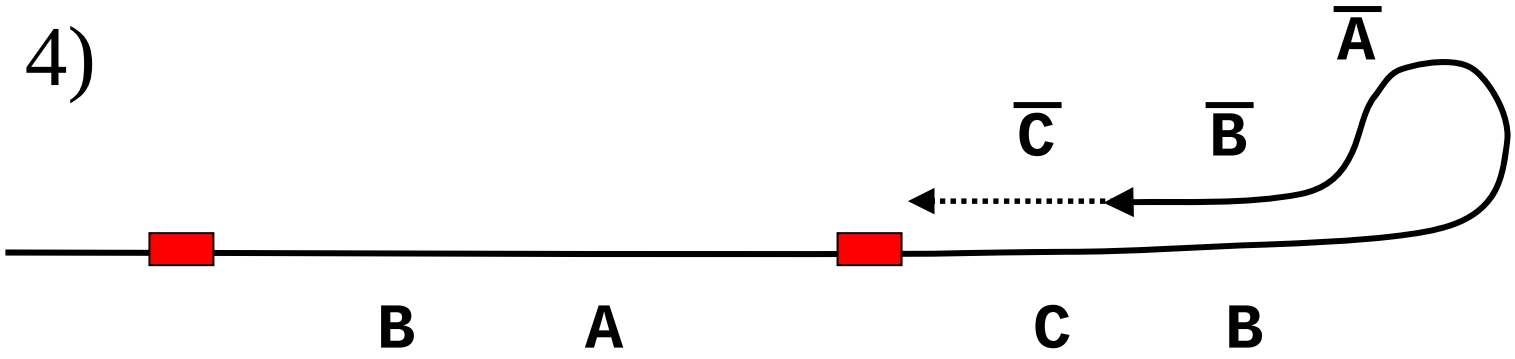
Komiya *et al.*

Whiplash PCR (WPCR)

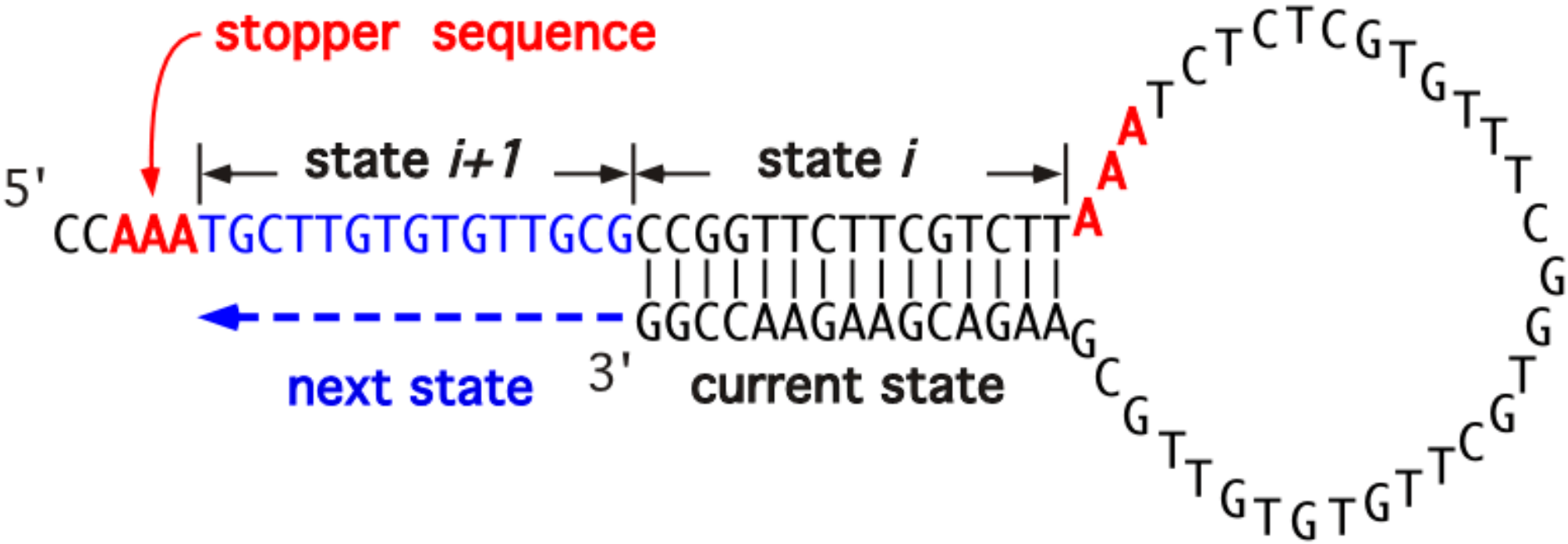
3)



4)

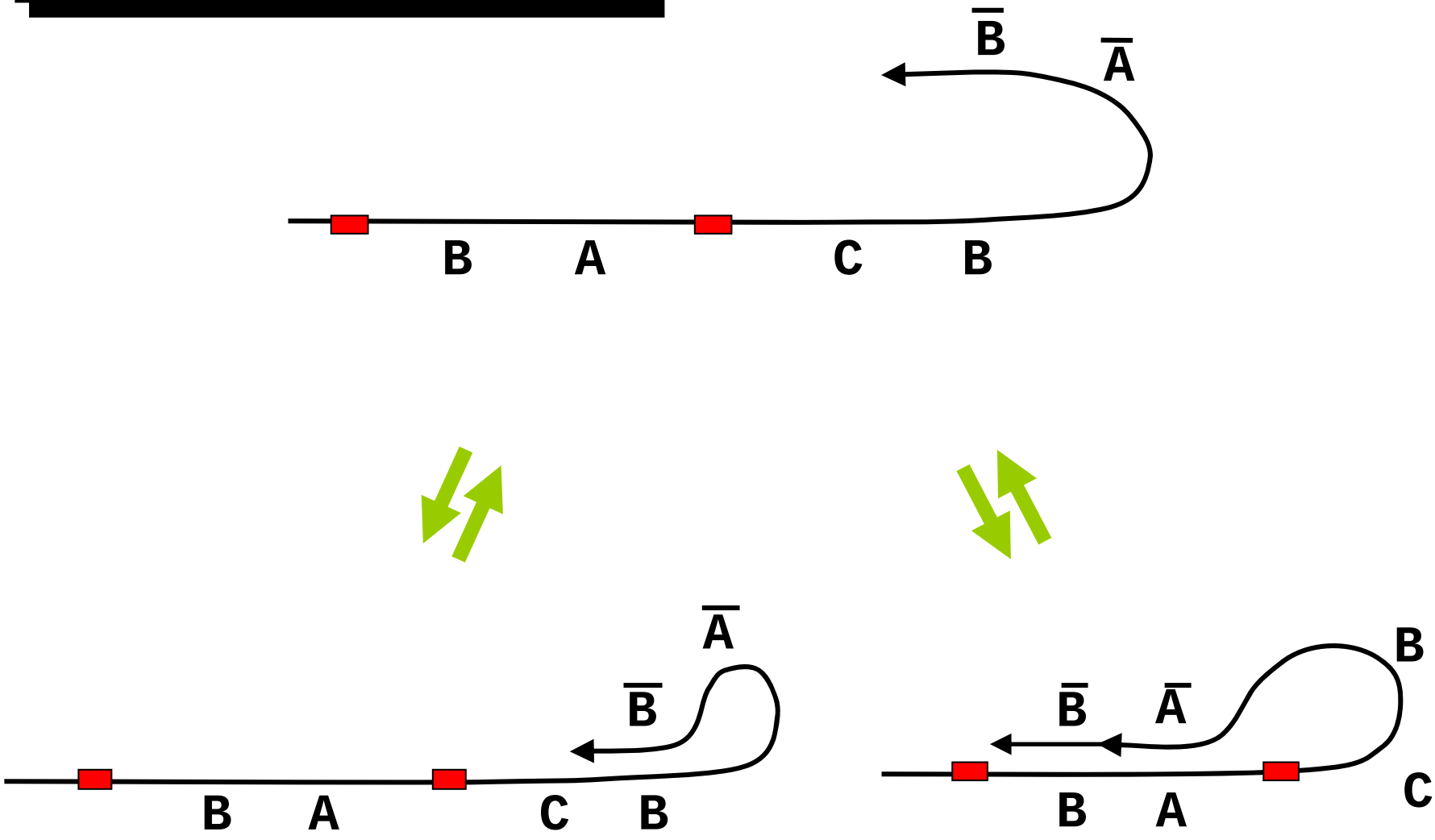


Polymerization Stop



Polymerization by DNA polymerase
with dATP, dCTP, dGTP

Back-hybridization

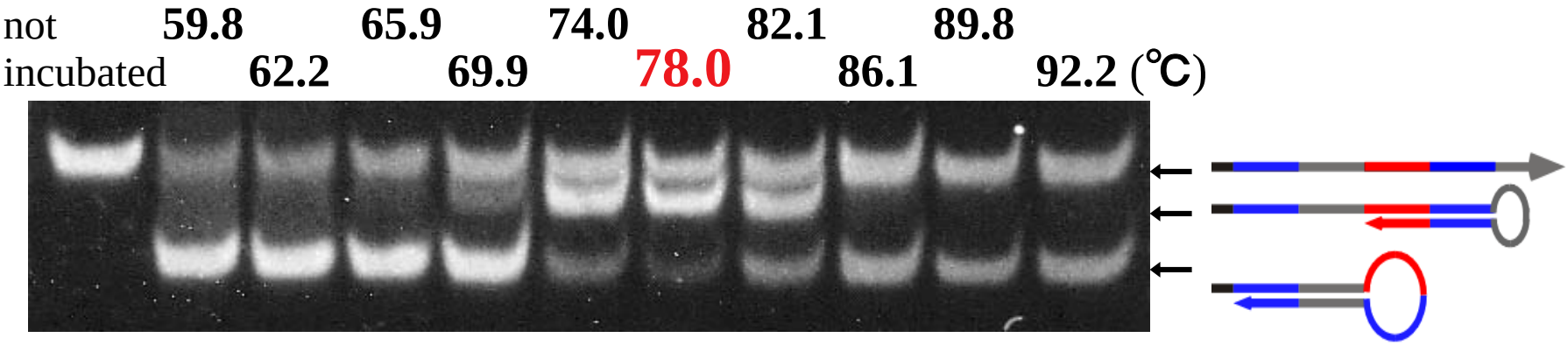


Competing Alternative Hairpin Forms

Temperature optimization for WPCR

- 8 M urea 8% PAGE

Komiya, et al.



in 1X *Pfx* buffer
(the composition unknown)
1 mM $MgSO_4$
0.2 mM dATP, dCTP, dGTP
1.5 units Platinum *Pfx* DNA polymerase

Thermal schedule

94℃ for 1 min.

↓

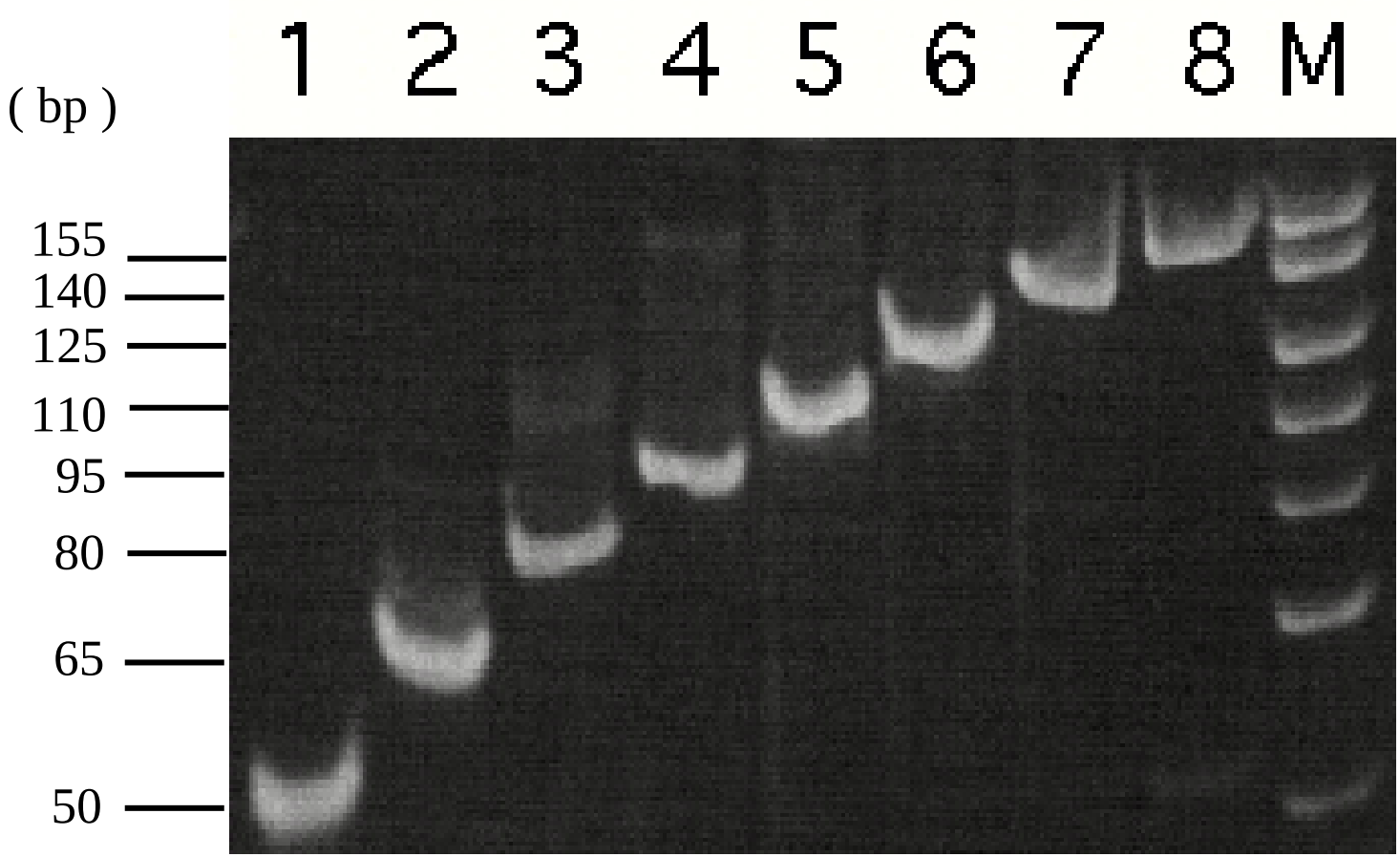
x ℃ for 5 min.

x =59.8 ~ 92.2

Successful implementation of transitions

- 12% PAGE

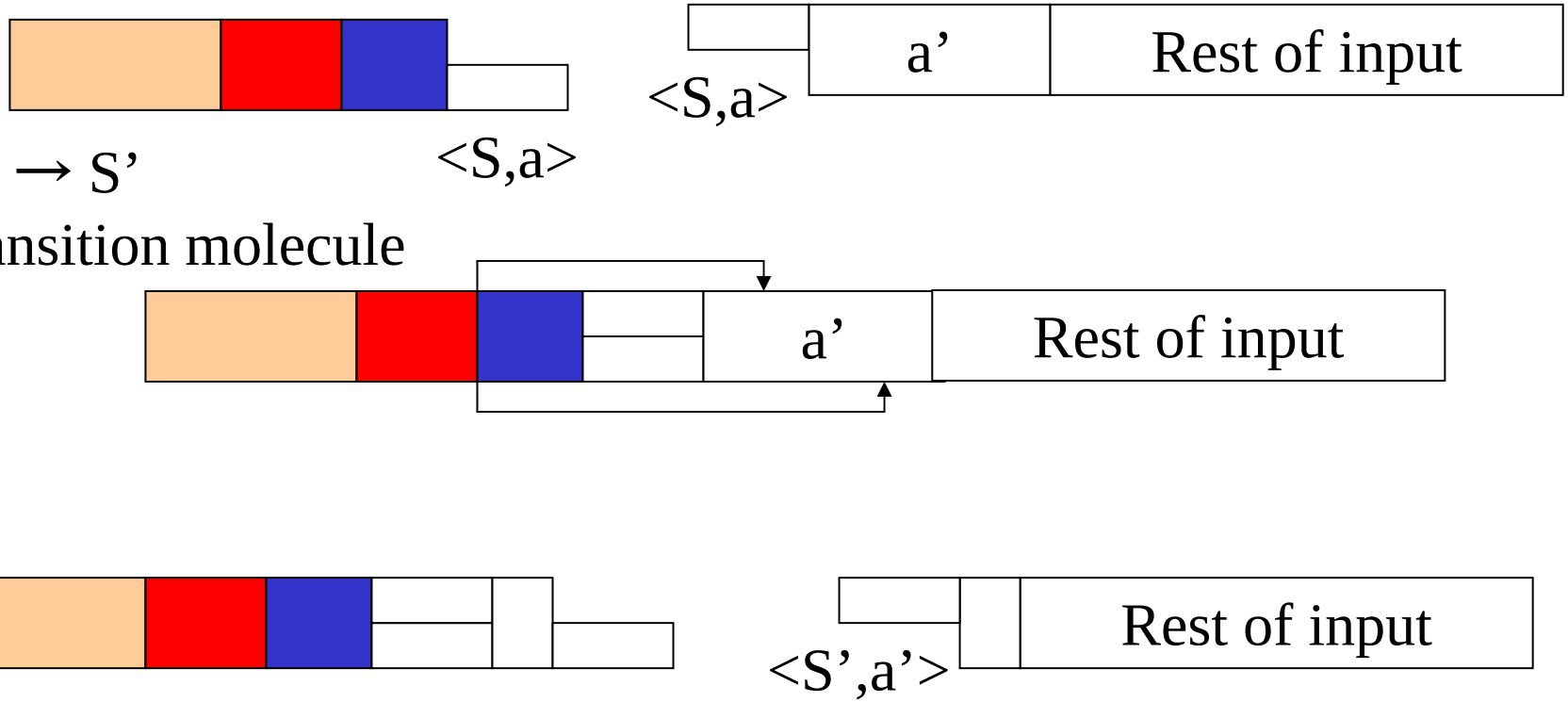
Komiya, et al.



Shapiro's DNA Automata

IIS-type restriction

Restriction cite Spacer



S,a → S'

Transition molecule

The input sequence encoding the symbol a' contains <S',a'> for each S'.
The transition molecule cuts the input at the site regulated by the spacer.

Shapiro's DNA Automata

- *Nature* 2001
- 2 input symbols, 2 states
- *Fok*I

a=**CTGGCT** b=**CGCAGC**

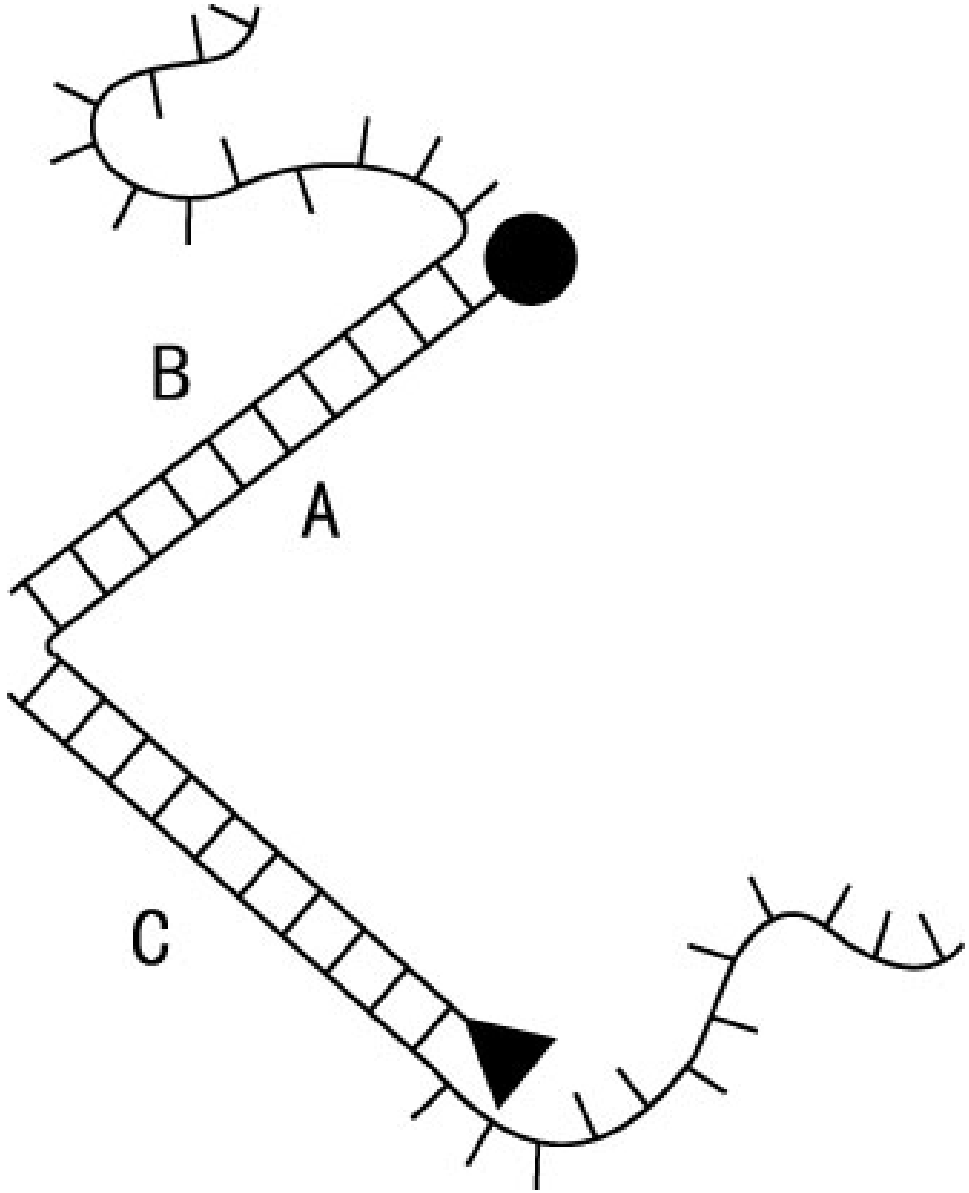
5′-p…22…GGATGTAC
3′-GGT…22…CCTACATGCCGAp

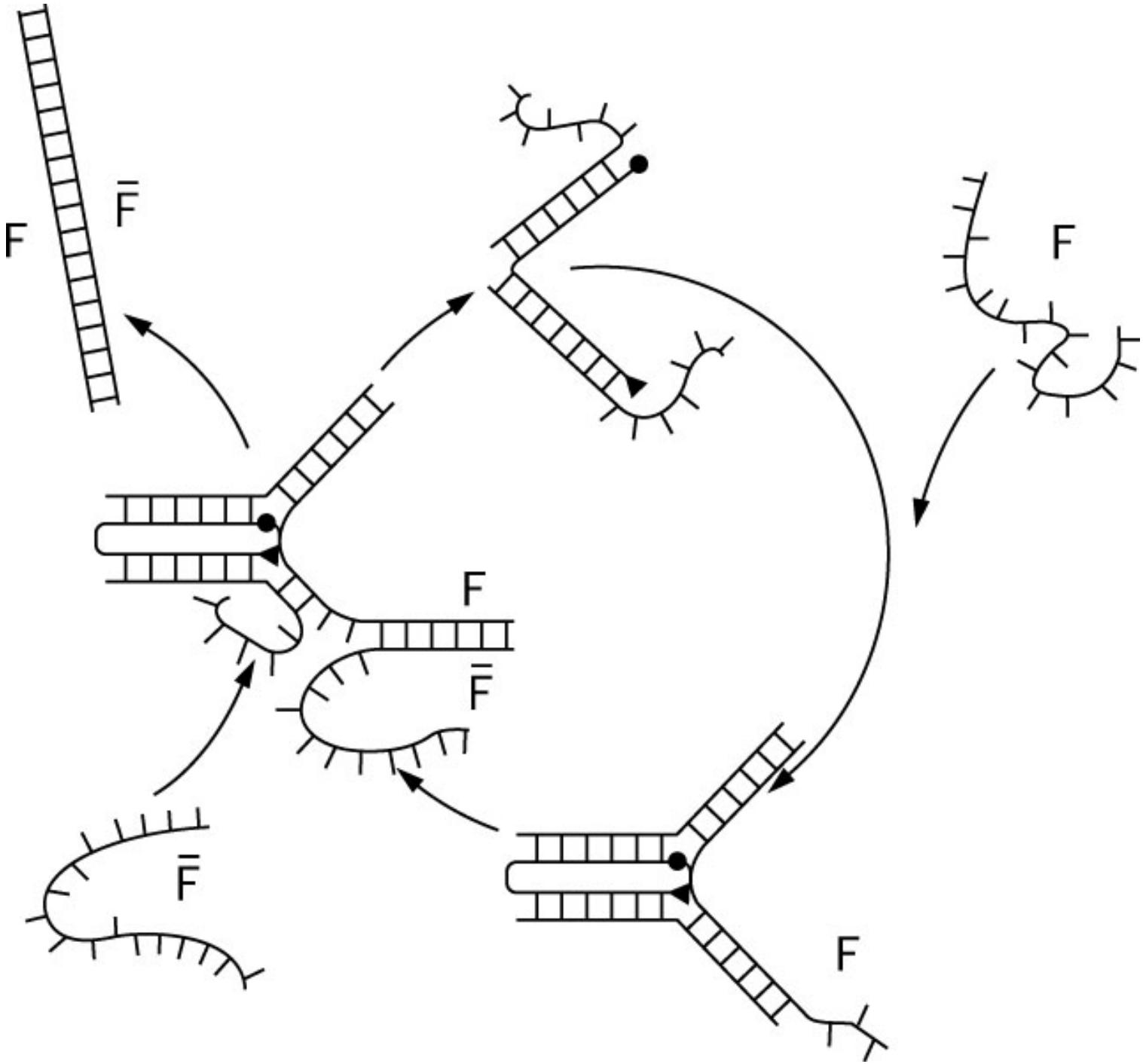
S0,a→S0

5′-p…22…GGATGACGAC
3′-GGT…22…CCTACTGCTGCCGAp

S0,a→S1

Yurke's Molecular Tweezers




F
F̄
F
F
F̄
F̄
F

Introduction to Molecular Computing

Table of Contents:

New Computational Paradigms

- Membrane Computing
  - Paun
- Amorphous Computing
  - MIT Group
    - Abelson & Sussman
    - Knight
- And others…
  - Smart Dust
  - Programmable Matter
  - Quantum-Dot Cell Automaton
  - …

Cell Membrane Model

- G. Paun (1998)
- Control of computation process using membrane
- Supercell system ＝ universal computation model

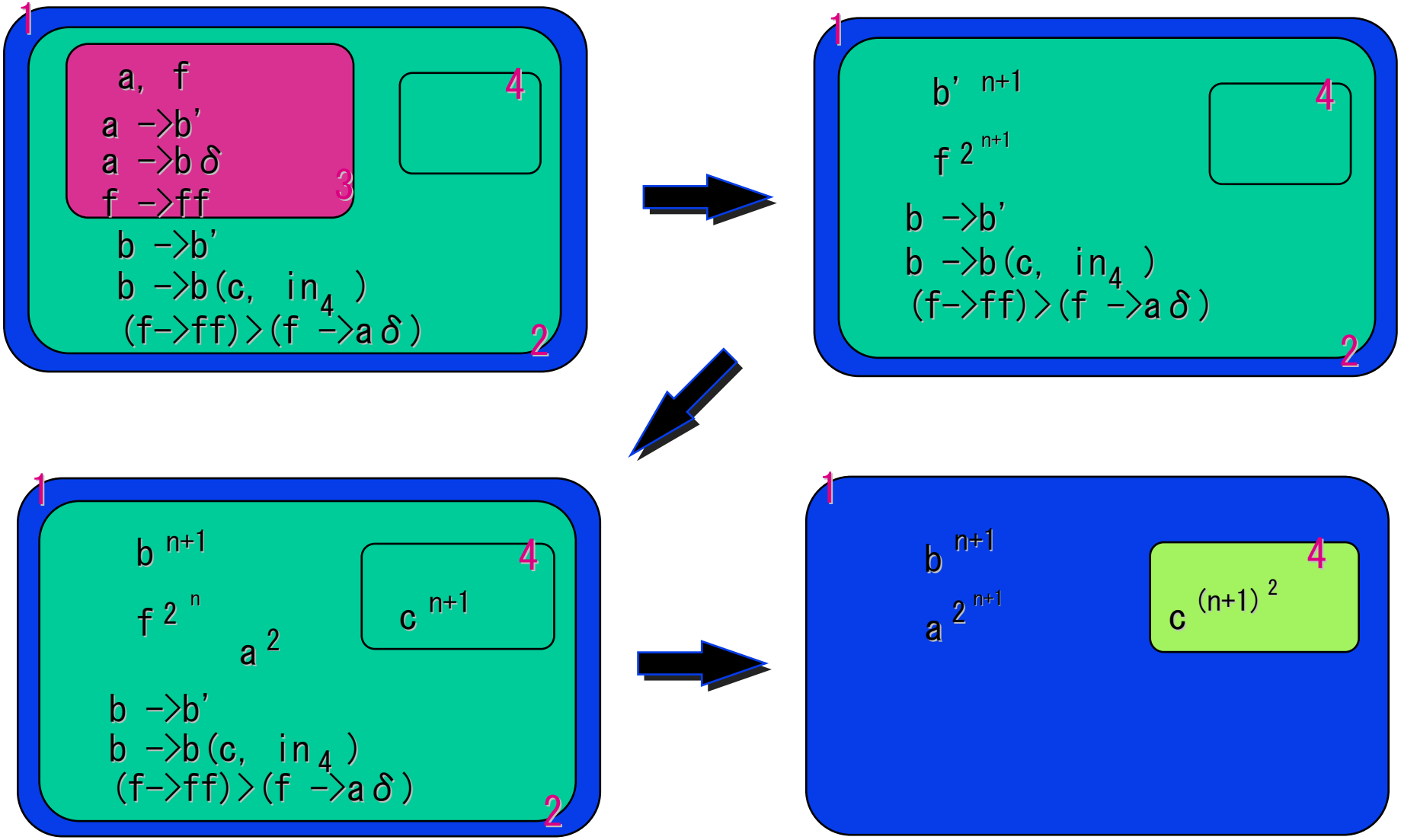
(e.g.) $G=(V, \mu, M_1, \ldots, M_4, (R_1, \rho_1), \ldots, (R_4, \rho_4), 4)$

$V=\{a, b, b', c, f\}$ alphabet

$\mu=[_1[_2[_3]_3[_4]_4]_2]_1$ membrane structure

M_i multiset of elements within membrane “i”

(R_i, ρ_i) ordered set of rules within membrane “i”



Computation of "n^2" using cell membrane model

Amorphous Computing

- New computation paradigm for self-assembly
  - Microfabrication and cytoengineering
  - Various processors at low cast
- Computational particle
  - Small computational power
    and small memory
  - Random distribution, mobility
  - Asynchronous, local interaction
  - Wrong behavior,
    environmental influence
  - Identical program
  - No knowledge of their location
    nor orientation
  - Short distance (radius: r) communication with the neighboring particles
- Massive parallel computation system as a whole
- Simulation of the self-assembly of a circuit

What is Amorphous Computing?

- Background
  - Microfabrication and cytoengineering
  - Developing various processors at low cost
    (Not necessary to work precisely)
  - Study as a new computational paradigm
- Developing the model as an aggregate of "computational particles" that are randomly distributed and interact locally and asynchoronously
- How can it be programmed effectively?
  - Relation to the formation of biological structure?
  - Is it possible to use biology for implementation, not just as a metaphor?

Characteristics of Computational Particles

- Have possibility of failure
- Will be influenced by the environment
- May make some movements
- May move around
- Have small computational power and small memory
- All particles are programmed identically
  （Capable of staying locally and generating random numbers）
- Have no knowledge about their location and orientation
- Make short distance (radius: r) communication with neighboring particles
- Massive parallel computational system as a whole

Pattern Formation Using Wave Propagation

- Start with first “anchor” particle, and convey the message（with information of the hop）
- Related to biological pattern formation
- “Impediment to growth” and “tropism” can be programmed using 2 anchor particles
- Program with Coore’s growing-point language(GPL), and compile to set into a particle

Quantum Dot Computer

- Bluffing?
- Different from quantum computers
- Quantum dot cell automaton (QCA)
  - Line up 4 quantum dots like dominoes
  - Electrons move inside dominoes (cells) by tunnel effect
  - The condition transmits by interaction of dominoes
- No need for wiring?
- Still, quantum dots must be arranged properly

Homework

① Explain methods of;
- DNA/RNA secondary structure prediction
- Minimum energy and partition function calculation using dynamic programming
- Sequence design using secondary structure
（use references）

② Explain DNA self-assembly and possibility of realization and applications of molecular machines

BASICS

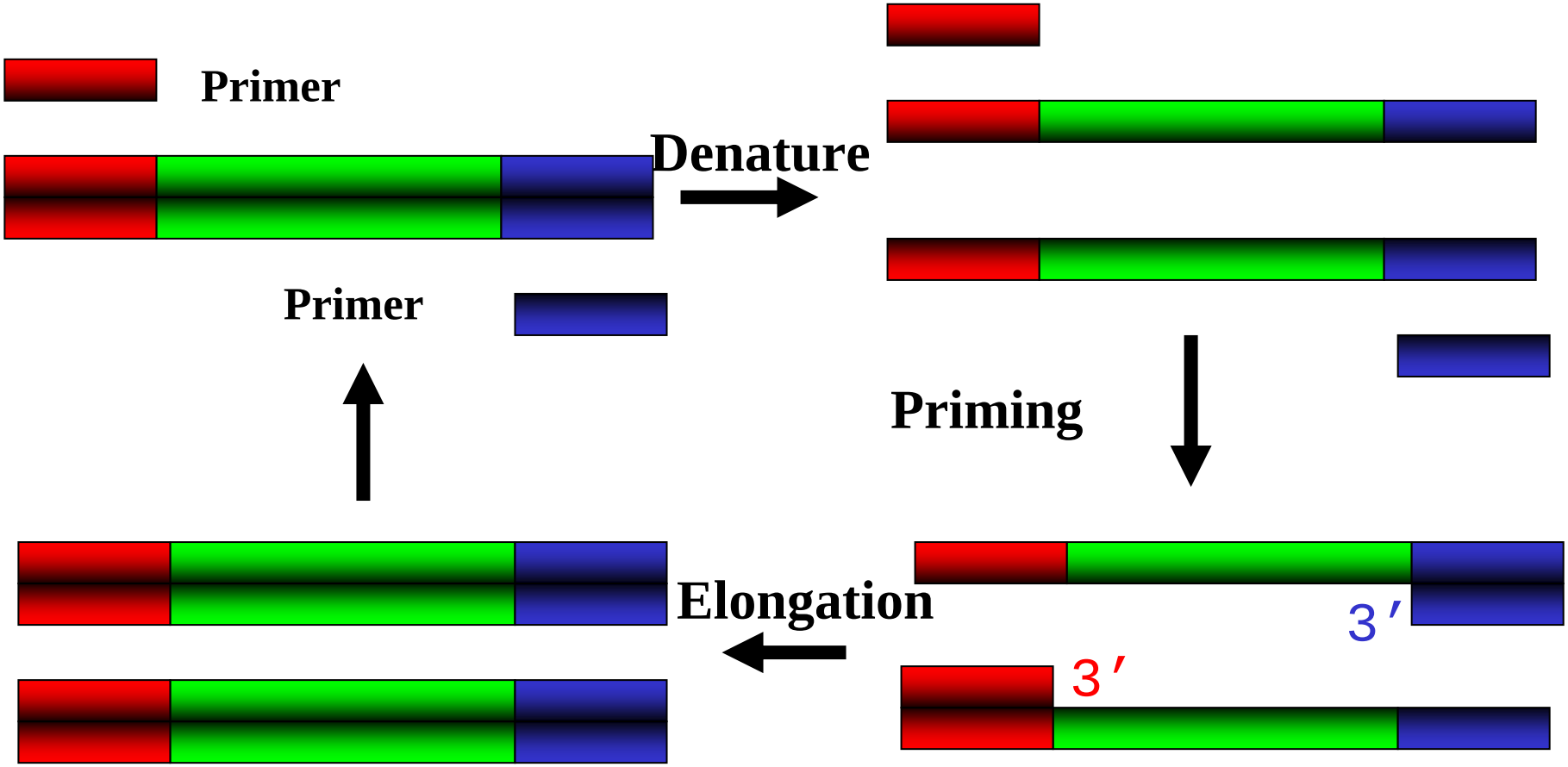
DNA

- Sugar
  - Deoxyribose
- Phosphate
- Bases
  - Purine Bases --- 2 rings (hexagon and pentagon)
    - Adenine (A)
    - Guaninen (G)
  - Pyrimidine Bases --- 1 ring (hexagon)
    - Thymine (T)
    - Cytosine (C)

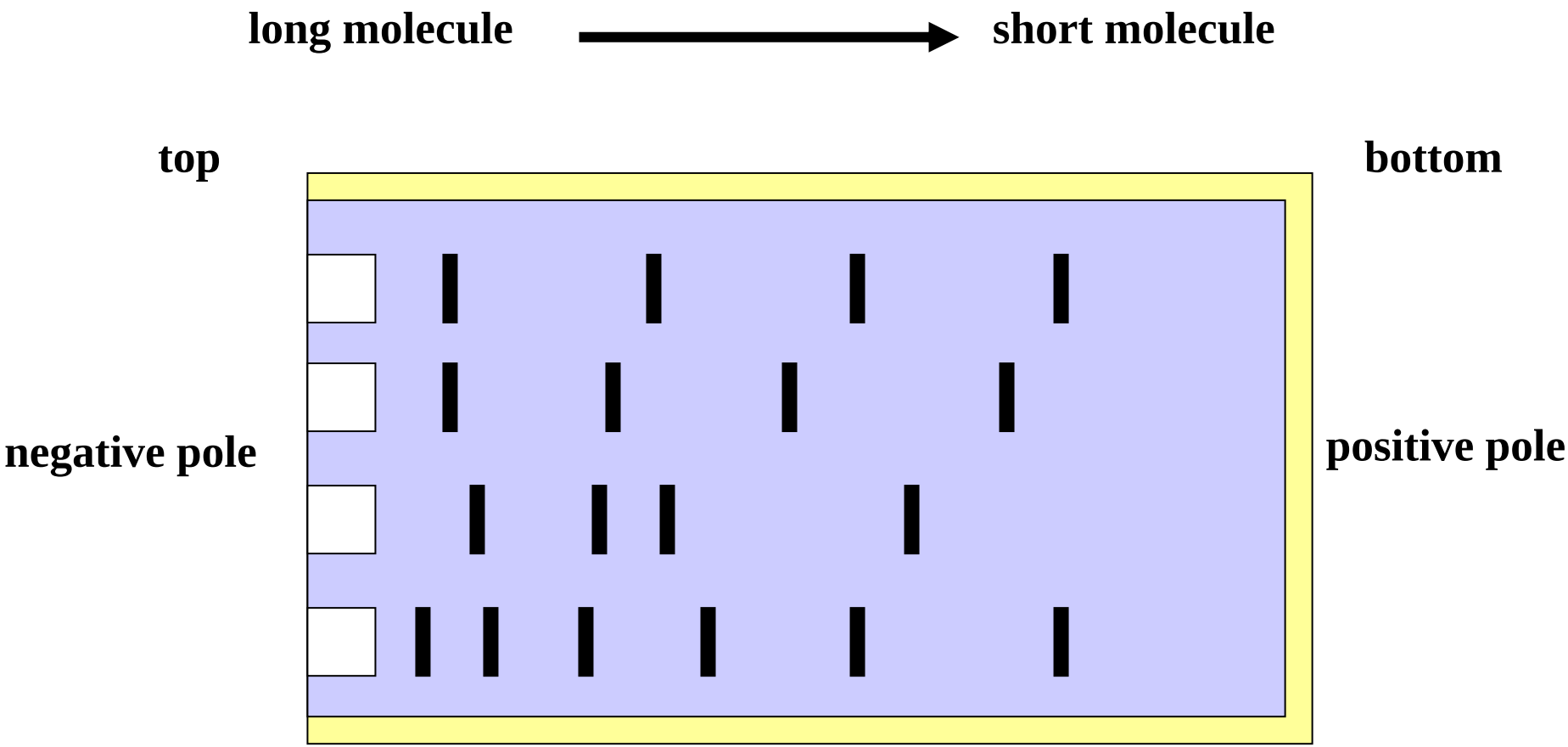
Experimental Operations

- PCR（Polymerase Chain Reaction）
- Gel electrophoresis
- Affinity separation
- Restriction enzyme digestion
- Coupling with ligase
- Cloning and sequencing

PCR (Polymerase Chain Reaction)

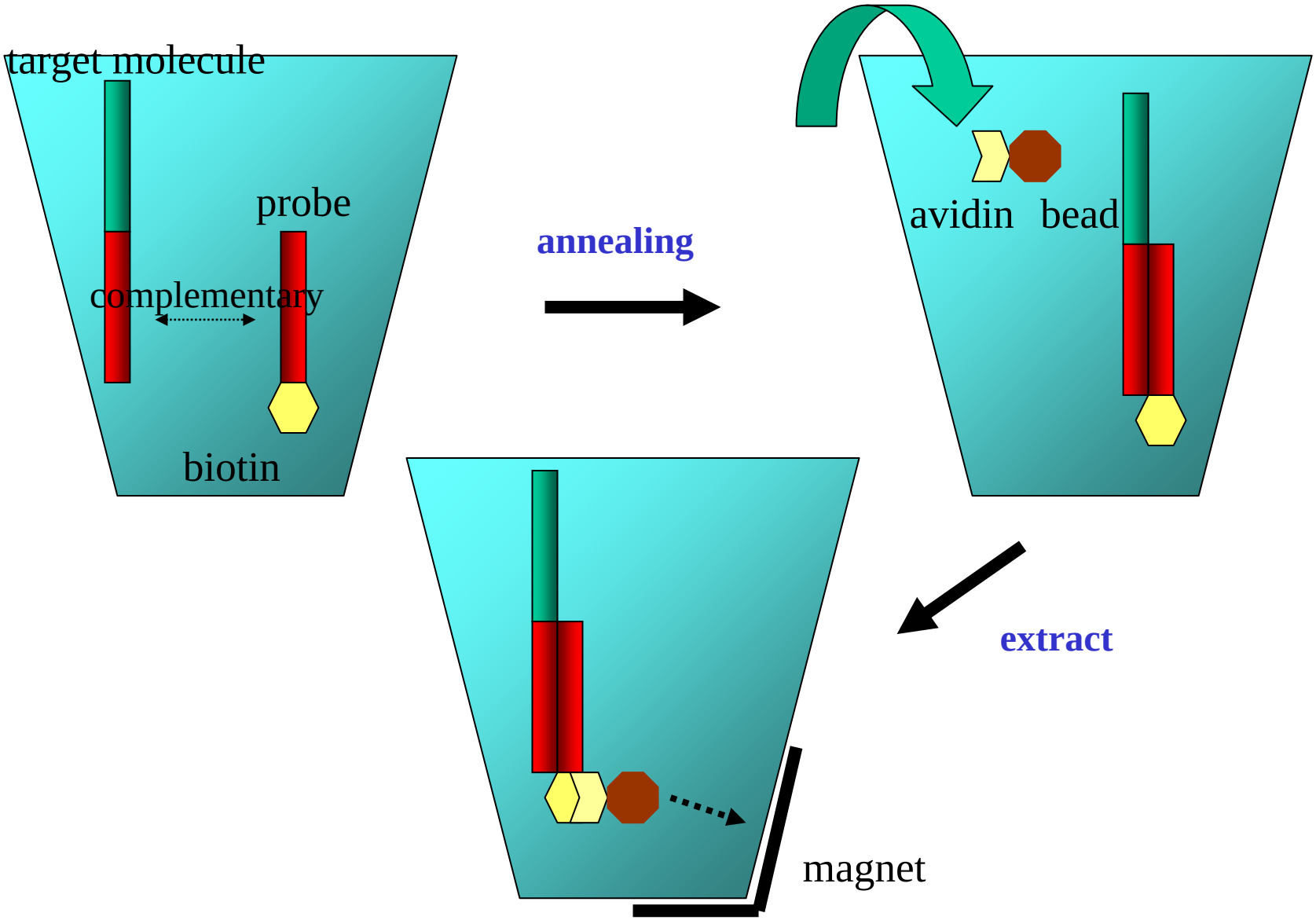


Gel Electrophoresis



polyacrylamide gel electrophoresis

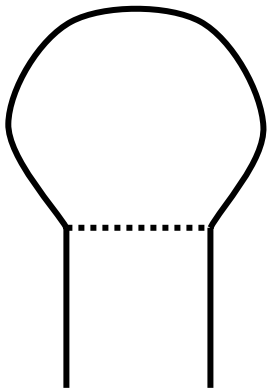
Dissociation of Single-Stranded DNA



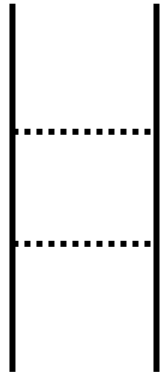
Secondary Structure of DNA（RNA） and Its Prediction

Secondary Structure of DNA（RNA)

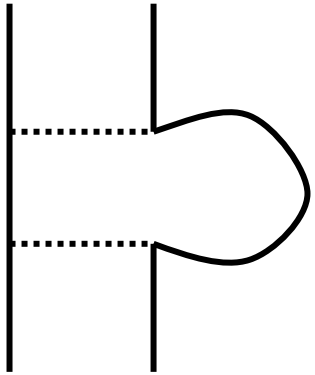
- A set of base pair *i.j*
- *k-loop* --- a loop closed by *k* base pairs
  - 1- loop
    - Hairpin
  - 2- loop
    - Stack
    - Bulge
    - Interior
  - Multiple loop
- Energy is assigned to each loop



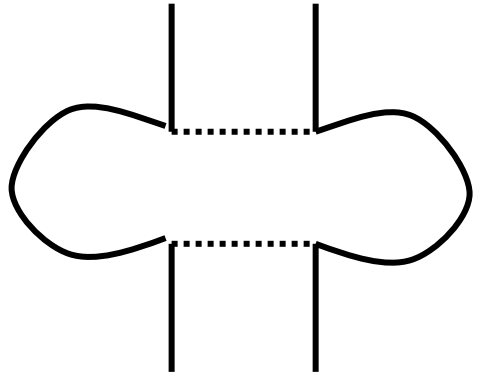
Hairpin



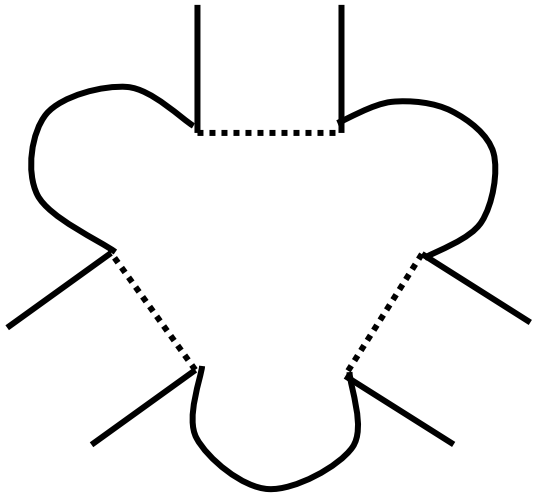
Stack



Bulge



Interior



3-loop

Assign energay to each of these structures
（nearest neighbor model）

Dynamic Programming

- $W(i, j)$: Minimum energy between i-th and j-th bases
- $V(i, j)$: Minimum energy when i, j form a pair

- $$W(i, j) = \min(W(i+1, j), W(i, j-1), V(i, j), \min_{i \le k < j}(W(i, k)+W(k+1, j)))$$

- $$V(i, j) = \min(\mathrm{eh}(i, j), \mathrm{es}(i, j)+V(i+1, j-1), VBI(i, j), VM(i, j))$$
  - $\mathrm{eh}(i, j)$: Hairpin energy
  - $\mathrm{es}(i, j)$: Stack energy

Dynamic Programming

- $VBI(i, j) = \min\limits_{\substack{i<i'<j'<j \\ i'-i+j-j'>2}}(\mathrm{ebi}(i, j, i', j')+V(i', j'))$
  - $\mathrm{ebi}(i, j, i', j')$: Interior loop energy

    $\rightarrow$ $O(n^4)$

- $VM(i, j) = \min\limits_{i<k<j-1}(W(i+1, k)+W(k+1, j-1))$
  - When multiple loop energy is 0

Interior Loop

- If interior loop energy ebi(i, j, i', j') is proportional to the length of the loop, $(i'-i+j-j') \times c$

- $VBI(i, j) = \min_{l}(VBI(i, j, l))$
- $VBI(i, j, l) = \min(VBI(i+1, j, l-1) + c,$
  $VBI(i, j-1, l-1) + c,$
  $c \times l + V(i+1, j-l+1),$
  $c \times l + V(i+l-1, j-1))$

$\rightarrow \quad O(n^3)$

Multiple Loop

- Approximate energy of a multiple loop:

$$a + b \times k' + c \times k$$

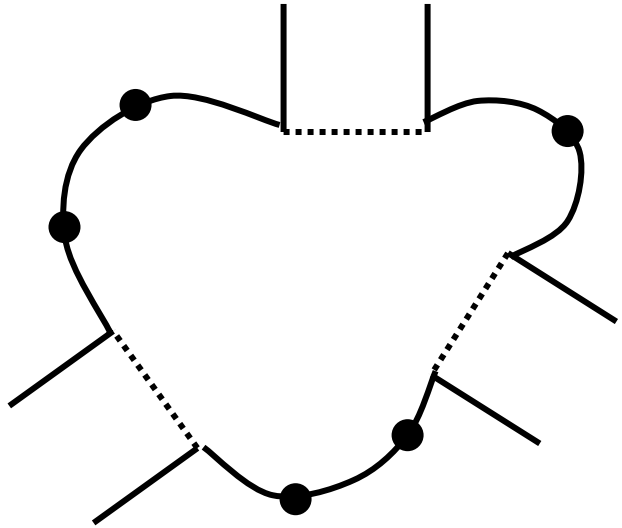
k' : Number of bases outside pairs

k : Number of pairs

$\rightarrow$ $O(n^3)$

$k' = 5$

$k = 3$



McCaskill's Algorithm

- Rather than calculating energy of each structure,

  Calculate energy distribution of all possible structures
  - Partition function
  - Probability of the formation of specific base pair
- Both can be calculated using dynamic programming

Basic Arrays

- w[i,j]

  minimum energy between i and j
- ww[k,j]

  w[k,j] on condition that k forms a pair

  -- reusable, only needed to memorize

  the case of j and j-1
- v[i,j]

  minimum energy between i and j

  when i, j form a pair

- Initialize all arrays to INF (infinite)

```
for (j=2; j<=n; j++)
    for (i=j-1; i>=1; i--) {
        ww[i,j] = ww[i,j-1];
        if (i.j is a pair)
          ww[i,j] = min(ww[i,j], v[i,j]);
        for (temp=INF, k=i+1; k<=j; k++)
          temp = min(temp, w[i,k-1]+ww[k,j]);
        w[i,j] = min(temp, ww[i,j]);
    }
```

Arrays for Multiple Loop

- v[i,j]

  minimum energy between i and j

  when i, j form a pair
- vm[i,j]

  minimum energy under assumption

  that the i, j pair belongs to a multiple loop

  -- includes at least one pair
- vvm[k,j]

  vm[k,j] on condition that k forms a pair

  -- reusable, only needed to memorize

  the case of j and j-1

```
for (j=2; j<=n; j++)
  for (i=j-1; i>=1; i--) {
    if (i.j is a pair) {
      v[i,j] = min(v[i,j], Hairpin energy);
      for (l=i+2; l<j-1; l++)
        for (k=l-1; k>i; k--)
          if (k.l is a pair)
            v[i,j] = min(v[i,j],
                         v[k,l]+ 2-loop energy);
      for (temp=INF, k=i+2; k<=j-1; k++)
        temp = min(temp, vm[i+1,k-1]+vvm[k,j-1]);
      v[i,j] = min(v[i,j], temp+MLclosing+MLintern);
  }
  set vm and vvm;
}
```

Multiple Loop

MLclosing

- Approximate energy of a multiple loop:

$$a + b \times k' + c \times k$$

k' : Number of bases outside pairs

k : Number of pairs

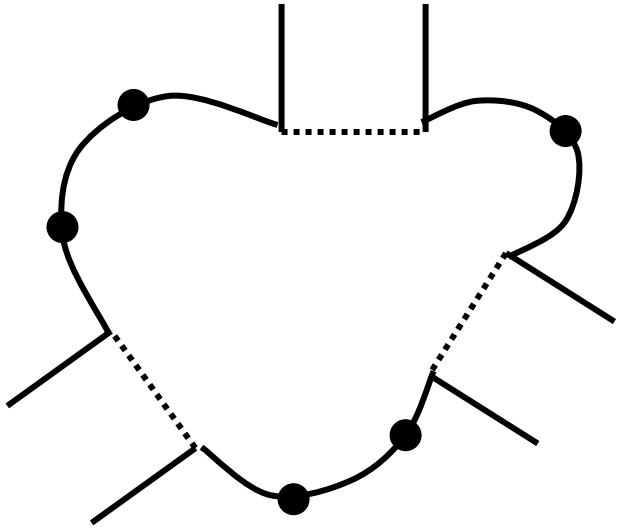
MLintern

$\rightarrow \quad O(n^3)$

MLbase

$k' = 5$

$k = 3$



Setting vm and vvm:

```
vvm[i,j] = vvm[i,j-1]+MLbase;
if (i.j is a pair)
    vvm[i,j] = min(vvm[i,j], v[i,j]+MLintern);
for (temp=INF, k=i+1; k<=j; k++) {
    temp = min(temp, vm[i,k-1]+vvm[k,j]);
    temp = min(temp, MLbase*(k-i)+vvm[k,j]);
}
vm[i,j] = min(temp, vvm[i,j]);
```

Partition Function

- With state energy G,
  the probability of state occurrence is
  proportional to Boltzmann factor $\exp(-G/kT)$

- Partition function Z is a sum of
  the Boltzmann factors of all states

- Probability of state occurrence of energy G is
  given by $\exp(-G/kT)/Z$

Calculation of Partition Function

- Instead of calculating the minimum energy while traversing secondary structures, calculate the sum of the Boltzmann factors while traversing secondary structures

Minimum Energy	Partition Function
G	exp(-G/kT)
initial value INF	initial value 0
min	+
+	*

Basic Arrays

- w[i,j]

  partition function between i and j
- ww[k,j]

  w[k,j] on condition that k forms a pair

  -- reusable, only needed to memorize

  the cases of j and j-1
- v[i,j]

  partition function between i and j

  when i, j form a pair

- Initialize all arrays to INF（infinite）

```
for (j=2; j<=n; j++)
    for (i=j-1; i>=1; i--) {
        ww[i,j] = ww[i,j-1];
        if (i.j is a pair)
            ww[i,j] = ww[i,j]+v[i,j];
        for (temp=0, k=i+1; k<=j; k++)
            temp = temp+w[i,k-1]*ww[k,j];
        w[i,j] = temp+ww[i,j];
    }
```

Arrays for Multiple Loop

- v[i,j]

  partition function between i and j

  when i, j form a pair
- vm[i,j]

  partition function under the assumption

  that the i, j pair belongs to a multiple loop

  -- includes at least one pair
- vvm[k,j]

  vm[k,j] on condition that k forms a pair

  -- reusable, only needed to memorize

  the cases of j and j-1

```
for (j=2; j<=n; j++)
    for (i=j-1; i>=1; i--) {
        if (i.j is a pair) {
            v[i,j] = v[i,j]+ Hairpin partition function;
            for (l=i+2; l<j-1; l++)
                for (k=l-1; k>i; k--)
                    if (k.l is a pair)
                        v[i,j] = v[i,j]+v[k,l]* 2-loop partition function;
            for (temp=0, k=i+2; k<=j-1; k++)
                temp = temp+vm[i+1,k-1]*vvm[k,j-1];
            v[i,j] = v[i,j]+temp*expMLclosing*expMLintern;
        }
        set vm and vvm;
    }
```

Setting vm and vvm:

```
vvm[i,j] = vvm[i,j-1]*expMLbase;
if (i.j is a pair)
    vvm[i,j] = vvm[i,j]+v[i,j]*expMLintern;
for (temp=0, k=i+1; k<=j; k++) {
    temp = temp+vm[i,k-1]*vvm[k,j];
    temp = temp+expMLbase^(k-i)*vvm[k,j];
}
vm[i,j] = temp+vvm[i,j];
```