C Formal models

C.1 Sticker systems

C.1.a Basic operations

The *sticker model* was developed by Rosweis et al. in the mid-1990s. It depends primarily on separation by means of hybridization and makes no use of strand extension and enzymes. It implements a sort of random-access binary memory. Each bit position is represented by a substrand of length m. A *memory strand* comprises k contiguous substrands, and so has length $n = km$ and can store k bits. *Sticker strands* or *stickers* are strands that are complementary to substrands representing bits. When a sticker is bound to a bit, it represents 1, and if no sticker is bound, the bit is 0. Such a strand, which is partly double and partly single, is called a *complex* strand.

Computations begin with a prepared *library* of strings. A (k, l) library uses the first $l \leq k$ bits as inputs to the algorithm, and the remaining $k-l$ for output and working storage. Therefore, the last $k - l$ are initially 0. There are four basic operations, which act on multi-sets of binary strings:

Merge: Creates the union of two *tubes* (multi-sets).

Separate: The operation separate(N, i) separates a tube N into two tubes: $+(N, i)$ contains all strings in which bit i is 1, and $-(N, i)$ contains all strings in which bit i is 0.

Set: The operation set(N, i) produces a tube in which every string from N has had its ith bit set to 1.

Clear: The operation clear(N, i) produces a tube in which every string from N has had its ith bit cleared to 0.

C.1.b Set cover problem

The *set cover problem* is a classic NP-complete problem. Given a finite set of p objects S, and a finite collection of subsets of S $(C_1, \ldots, C_q \subset S)$ whose union is S, find the *smallest* collection of these subsets whose union is S. For an example, consider $S = \{1, 2, 3, 4, 5\}$ and $C_1 = \{3, 4, 5\}, C_2 = \{1, 3, 4\}, C_3 = \{1, 2, 5\}, C_4 = \{3, 4\}$. In this case there are three minimal solutions: $\{C_1, C_3\}, \{C_3, C_4\}, \{C_2, C_3\}$.

algorithm Minimum Set Cover:

Data representation: The memory strands are of size $k = p + q$. Each strand represents a collection of subsets, and the first q bits encode which subsets are in the collection; call them *subset bits*. For example 1011 represents $\{C_1, C_3, C_4\}$ and 0010 represents $\{C_3\}$. Eventually, the last p bits will represent the union of the collection, that is, the elements of S that are contained in at lease one subset in the collection; call them *element bits*. For example, 0101 10110 represents $\{C_2, C_4\}$ $\{1, 3, 4\}$.

Library: The algorithm begins with the $(p + q, q)$ library, which must be initialized to reflect the subsets' members.

Step 1 (initialization): For all strands, if the i subset bit is set, then set the bits for all the elements of that subset. Call the result tube N_0. This is accomplished by the following code:

Initialize $(p + q, q)$ library in N_0
for $i = 1$ **to** q **do**
  separate$(+(N_0, i), -(N_0, i))$ //separate those with subset i
  for $j = 1$ **to** $|C_i|$ **do**
    set$(+(N_0, i), q + c_i^j)$ //set bit for jth element of set i
  end for
  $N_0 \leftarrow$ merge$(+(N_0, i), -(N_0, i))$ //recombine
end for

Step 2 (retain covers): Retain only the strands that represent collections that cover the set. To do this, retain in N_0 only the strands whose last p bits are set.

for $i = q + 1$ **to** $q + p$ **do**
  $N_0 \leftarrow +(N_0, i)$ //retain those with element i
end for

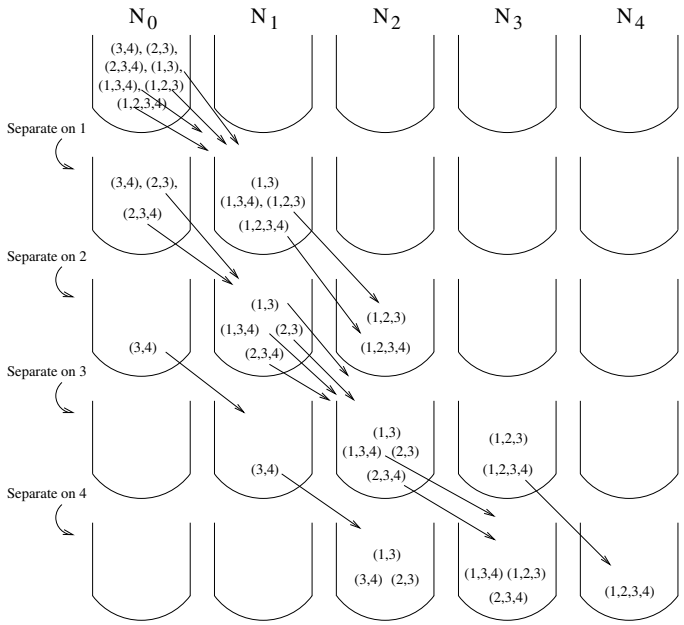

Figure IV.11: Sorting of covers by repeated separations. [source: Amos, Fig. 3.4]

Step 3 (isolate minimum covers): Tube N_0 now holds all covers, so we have to somehow sort its contents to find the minimum cover(s). Set up a row of tubes $N_0, N_1, \ldots, N_q$. We will arrange things so that N_i contains the covers of size i; then we just have to find the first tube with some DNA in it.

Sorting: For $i = 1, \ldots, q$, "drag" to the right all collections containing C_i, that is, for which bit i is set (see Fig. IV.11). This is accomplished by the following code:[10]

for $i = 0$ **to** $q - 1$ **do**
  for $j = i$ **down to** 0 **do**
    separate$(+(N_j, i+1), -(N_j, i+1))$ //those that do & don't have i

[10]Corrected from Amos p. 60.

$N_{j+1} \leftarrow \text{merge}(+(N_j, i+1), N_{j+1})$ //move those that do to N_{j+1}
$N_j \leftarrow -(N_j, i+1)$ //leave those that don't in N_j
end for
end for

Detection: Find the minimum i such that N_i contains DNA; N_i contains the minimum covers.
□

The algorithm is $\mathcal{O}(pq)$.

C.2 Splicing systems

It has been argued that the full power of a TM requires some sort of string editing operation. Therefore, beginning with Tom Head (1987), a number of *splcing systems* have been defined. The splicing operations takes two strings $S = S_1S_2$ and $T = T_1T_2$ and performs a "crossover" at a specified location, yielding S_1T_2 and T_1S_2. *Finite extended splicing systems* have been shown to be computationally universal (1996).

The *Parallel Associative Memory (PAM) Model* was defined by Reif in 1995. It is based on a restricted splicing operation called *parallel associative matching* (PA-Match) operation, which is named *Rsplice*. Suppose $S = S_1S_2$ and $T = T_1T_2$. Then,

$$\text{Rsplice}(S, T) = S_1T_2, \quad \text{if } S_2 = T_1,$$

and is undefined otherwise. The PAM model can simulate nondeterministic TMs and parallel random access machines.

Figure IV.12: The *fok*I restriction enzyme bound to DNA. [source: wikipedia]

D Enzymatic computation

The molecular computation processes that we have seen so far are externally controlled by a person or conventional automatic controller sequencing the chemical operations.[11] In *autonomous molecular computation* the chemical processes sequence themselves so that they do not require external control. This is also called "one-pot" molecular computation; that is, you put all the reactants in one pot, and the molecular processes do the rest. Autonomous molecular computation is essential for, example, controlling drug delivery in the body.

Shapiro and his colleagues have demonstrated how to implement finite state machines (FSMs) by autonomous molecular computation. In addition to DNA, it uses a restriction enzyme, ligase, and ATP (for fuel).

The implementation is based on the *fok*I restriction enzyme. "Once the

[11]This section is based primarily on: (1) Yaakov Benenson, Tamar Paz-Elizur, Rivka Adar, Ehud Keinan, Zvi Livneh, and Ehud Shapiro. Programmable and autonomous computing machine made of biomolecules. *Nature*, 414:430–434, 2001.
(2) Yaakov Benenson, Rivka Adam, Tamar Paz-Livneh, and Ehud Shapiro. DNA molecule provides a computing machine with both data and fuel. *Proceedings of the National Academies of Science*, 100(5):2191–2196, 2003.

protein is bound to duplex DNA via its DNA-binding domain at the 5′-GGATG-3′ : 5′-CATCC-3′ recognition site, the DNA cleavage domain is activated and cleaves, without further sequence specificity, the first strand 9 nucleotides downstream and the second strand 13 nucleotides upstream of the nearest nucleotide of the recognition site."[12] It leaves 4-nucleotide sticky ends. That is, the restriction enzyme cuts the DNA as follows:

```
GGATGNNNNNNNNN NNNNNNNNNN
CCTACNNNNNNNNNNNNN NNNNNN
```

The 'N's can be any nucleotides (respecting Watson-Crick complementarity, of course).

Both the current state of the FSM and the input string are represented by a double DNA strand. *fok*I operates at the beginning of this string and leaves a sticky end that encodes both the current state and the next input symbol (see p. 229 below). The state transitions of the FSM are encoded in *transition molecules*, which have sticky ends complementary to the state-symbol code at the beginning of the string. The rest of a transition molecule ensures that the string properly encodes the new state, including adding a new recognition site for the enzyme. A matching transition molecule binds to the string's sticky end, providing a new opportunity for *fok*I to operate, and so the process continues.

A state transition $(q, s_1) \to q'$ can be represented:

$$[q, s_1]s_2s_3 \cdots s_n t \Longrightarrow [q', s_2]s_3 \cdots s_n t$$

where $[q, s]$ represents a DNA sequence encoding both state q and symbol s, and t is a *terminator* for the string. The *fok*I enzyme cleaves off $[q, s_1]$ in such a way that a transition molecule can bind to the sticky end in a way that encodes $[q', s_2]$. A special *terminator* symbol marks the end of the input string.

As an example we will consider a two-state FSM on {a, b} that accepts strings with an even number of 'b's. Ignoring the terminator, DNA codes are assigned to the two symbols 'a' and 'b' as follows:

$$\begin{aligned} \text{a} &\mapsto AA\alpha\alpha aa \\ \text{b} &\mapsto BB\beta\beta bb \end{aligned}$$

[12] wikipedia, s.v. fokI.

where $A, \alpha, a, B, \beta, b$ are unspecified (by me) bases.[13] The bases are selected in such a way that either the first four bases ($AA\alpha\alpha$, $BB\beta\beta$) or the last four bases ($\alpha\alpha aa$, $\beta\beta bb$) encode the symbol. These alternatives represent the two machine states.

The transition molecules are constructed so that the distance between the recognition site (for *fok*I) and the next symbol depends on new state. As a consequence, when *fok*I operates it cleaves the next symbol code at a place that depends on the state. Therefore the sticky end encodes the state in the way that it represents the next symbol:

$$\begin{aligned} [q_0, \mathrm{a}] &\mapsto \alpha\alpha aa \\ [q_1, \mathrm{a}] &\mapsto AA\alpha\alpha \\ [q_0, \mathrm{b}] &\mapsto \beta\beta bb \\ [q_1, \mathrm{b}] &\mapsto BB\beta\beta \end{aligned}$$

The transition molecules are:

$(q_0, \mathrm{a}) \to q_0$ GGATGNNN
CCTAC$\overline{NNN\alpha\alpha aa}$

$(q_1, \mathrm{a}) \to q_1$ GGATGNNN
CCTAC$\overline{NNNAA\alpha\alpha}$

$(q_0, \mathrm{b}) \to q_1$ GGATG$NNNNN$
CCTAC$\overline{NNNNN\beta\beta bb}$

$(q_1, \mathrm{b}) \to q_0$ GGATGN
CCTAC$\overline{NBB\beta\beta}$

The Ns represent any bases as before. They are used as *spacers* to adjust the restriction site to represent the new state.

After transition to the new state the sense strand will look like this (for convenience assuming the next symbol is 'a'):

q_0 GGATG$NXXYYyyAA.\alpha\alpha aa$
q_1 GGATG$NNNXXYYyy.AA\alpha\alpha aa$

This is *after* attachment of the transition molecule but before restriction. Here XX represents either spacers or the first two bases of the previous first symbol, and $YYyy$ represents the last four bases of this symbol. The cleavage site is indicated by a period.

[13] Note that repeated letters might refer to different bases.

The longest strings processed in the PNAS experiments were 12.[14] Operation required about 20 seconds per step. However, the parallel speed was about 6.6×10^{10} ops/s/μl. Energy consumption was about $34kT$ per transition, which is only about $50\times$ the von Neumann-Landauer limit ($kT \ln 2$). The authors note, "Reaction rates were surprisingly insensitive to temperature and remained similar over the range of 2–20°C." This implementation also handles nondeterministic FSMs (just put in all the transition molecules), but the yield decreases exponentially (due to following out all the nondeterministic paths, breadth-first). Therefore it doesn't seem to be practical for nondeterministic machines.

[14]Benenson et al., PNAS **100** (5), March 4, 2003.