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 \( i \)th bit set to 1.
- **Clear:** The operation clear\((N,i)\) produces a tube in which every string from \( N \) has had its \( i \)th 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 \( S \) \((C_1, \ldots, C_q \subseteq S)\) 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 least 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($+(N_0, i), q + c^j_i$) //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
```
C. FORMAL MODELS

3.2 Filtering Models

\[(1,3), (1,2,3,4), (3,4), (2,3), (1,3,4), (1,2,3)\]

Separate on 1
Separate on 2
Separate on 3
Separate on 4

**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.
\[
N_{j+1} \leftarrow \text{merge}(+(N_j, i + 1), N_{j+1}) \quad \text{//move those that do to } N_{j+1}
\]
\[
N_j \leftarrow -(N_j, i + 1) \quad \text{//leave those that don’t in } N_j
\]
end for
end for

Detection: Find the minimum \(i\) such that \(N_i\) contains DNA; \(N_i\) contains the minimum covers.

The algorithm is \(O(pq)\).

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 splicing systems have been defined. The splicing operations takes two strings \(S = S_1 S_2\) and \(T = T_1 T_2\) and performs a “crossover” at a specified location, yielding \(S_1 T_2\) and \(T_1 S_2\). **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 \(S = S_1 S_2\) and \(T = T_1 T_2\). Then,

\[
Rsplice(S, T) = S_1 T_2, \quad \text{if } S_2 = T_1,
\]

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