# C Formal models

## C.1 Sticker systems

### C.1.a Basic operations

¶1. The *sticker model* was developed by Rosweis et al. in the mid-1990s.

¶2. It depends primarily on separation by means of hybridization and makes no use of strand extension and enzymes.

¶3. It implements a sort of random-access binary memory.

¶4. **Substrands:** Each bit position is represented by a substrand of length $m$.

¶5. **Memory strands:** A *memory strand* comprises $k$ contiguous substrands, and so has length $n = km$ and can store $k$ bits.

¶6. **Sticker strands:** *Sticker strands* or *stickers* are strands that are complementary to substrands representing bits.
When a sticker is bound to a bit, it represents 1.
If no sticker is bound, the bit is 0.

¶7. **Complex:** Such a strand, which is partly double and partly single, is called a *complex*.

¶8. **Library:** 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, that last $k - l$ are initially 0.

¶9. **Operations:** There are four basic operations, which act on multi-set of binary strings:

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

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

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

¶13. **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

¶1. **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 $S$.
E.g., finding the minimum number of people who can do all the tasks/.

¶2. **Example:** $S = \{1, 2, 3, 4, 5\}$.
$C_1 = \{3, 4, 5\}, C_2 = \{1, 3, 4\}, C_3 = \{1, 2, 5\}, C_4 = \{3, 4\}$.
In this case there are three minimal solutions: $\{C_1, C_3\}, \{C_3, C_4\}, \{C_2, C_3\}$.

¶3. **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*.

¶4. For example 1011 represents $\{C_1, C_3, C_4\}$ and 0010 represents $\{C_3\}$.

¶5. 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 $\{C_2, C_4\}$ $\{1, 3, 4\}$.

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

¶7. **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 $N_0$.

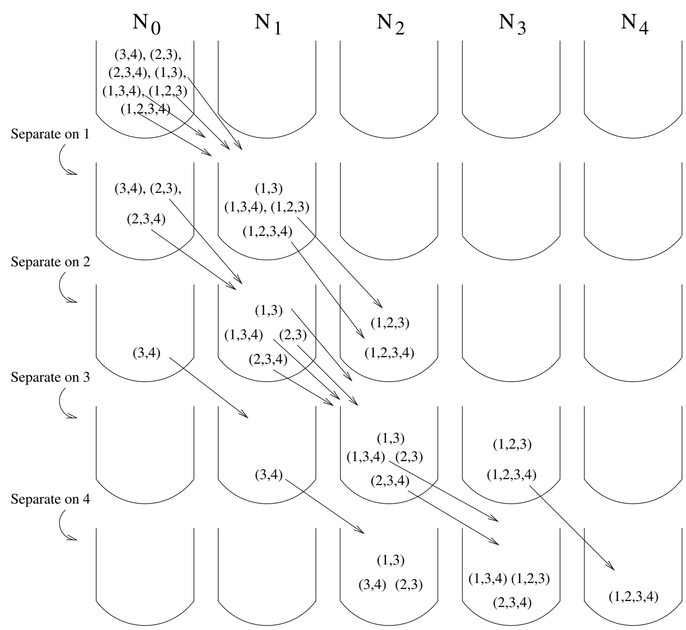


Figure IV.9: Sorting of covers by repeated separations. [source: Amos, Fig. 3.4]

¶8. 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 $j$th element of set $i$
  **end for**
  $N_0 \leftarrow$ merge$(+(N_0, i), -(N_0, i))$ //recombine
**end for**

¶9. **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.

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

¶11. **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 it that $N_i$ contains the covers of size $i$; then we just have to find the first tube with some DNA in it.

¶12. **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.9.

¶13. This is accomplished by the following code:

**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 $i$
    $N_{j+1} \leftarrow$ 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**

## C.2 Splicing systems

¶1. It has been argued that the full power of a TM requires some sort of string editing operation.

¶2. **Splicing systems:** Therefore, beginning with Tom Head (1987), a number of *splcing systems* have been defined.

¶3. **Splicing operation:** 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 $T_1 S_2$.

¶4. *Finite extended splicing systems* have been show to be computationally universal (1996).

¶5. **Parallel Associative Memory Model:** The *Parallel Associative Memory (PAM) Model* was defined by Reif in 1995.

¶6. It is based on a restricted splicing operation called *parallel associative matching* (PA-Match) operation called *Rsplice*.

¶7. Suppose $S = S_1 S_2$ and $T = T_1 T_2$.

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

and is undefined otherwise.

¶8. The PAM model can simulate nondeterministic TMs and parallel random access machines.