CONSTRUCTING A 
DETERMINISTIC FINITE 
AUTOMATA FOR THE 

KMP - Algorithm

. skeleton machine
. Mp machine

Mp is not exactly a deterministic finite automaton: because inserting a state will reset the input tape.
on an input symbol it can jump to different states, several times in a row.

We are going to make the construction a deterministic finite automata.

This new DFA makes exactly one state transition per input symbol.
A finite automata accepts a set of strings over the input alphabet $\Sigma$.

$\Sigma^*p$: we want a finite automata $M$ such that

$L(M) = \Sigma^*p \in \{\text{yes}, \text{no}\}$.

Set of patterns $\equiv \Sigma^*p$

$\text{yes = accepted} = \equiv p \text{ occurs exactly in the input string}$
Algorithm for constructing a DFA \( M \) for \( \Sigma^* \):

\[ L(M) = \Sigma^* \]  

**Input**: a pattern

\[ P = p_1 \cdots p_c, \ p_i \in \Sigma \]

**Output**: a DFA \( M \) such that \( L(M) = \Sigma^* \)

**Algorithm**

1. Use the Failure function algorithm to construct
the failure function $f$ for $p$.

2) Let $M = (S, \Sigma, \delta, 0, \epsilon \in \Sigma)$

\[ S = \xi_0, \xi_1, \ldots, \xi_l \]

compute $\delta$ as follows:

BEGIN

\[ \text{for } j = 1 \text{ to } l \text{ do } \]

END
\begin{equation}
\delta(j-1, p_j) = j;
\end{equation}
\begin{theorem}
\text{For each } r \in \Sigma, r \neq p_j, \text{ do}
\delta(0, r) = 0
\end{theorem}
\begin{theorem}
\text{For } j = 1 \text{ to } e \text{ do}
\text{for each } r \in \Sigma \text{ do}
\text{if } r \neq p_{j+1} \text{ then}
\delta(j, r) = \delta(k(j), r)
\end{theorem}
\text{END}

\text{EXAMPLE}
\begin{equation}
p = aabbbbaaab
\end{equation}
Construct DFA that accepts \( \{ a, b \}^* \).

\[ t = ab aab aa bbaab \]

<table>
<thead>
<tr>
<th>Input</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>0</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>a</td>
</tr>
</tbody>
</table>

The states and transitions are represented in the diagram above.
M makes exactly one state transition on each input symbol.

**Theorem**

In time $O(|p| + |t|)$
we can determine whether $p$ is a substring of $t$.

**Significance of Alignment**

Alignment & Homology
An alignment is a "hypothesis."

Two sequences that are derived from a common ancestor are called **homologous**.

"Similarity" vs "homology"

**Similarity**: matches, substitutions of similar amino acids
Gene 1
Gene 2
Gene 3
Gene 4
Gene 5
Gene 6
Gene 7
Gene 8
Gene 9
Gene 10

Homology: it is about Biology and Evolution

Protein sequences,
common ancestral letters

One into the
other: similar
Homologous genes

Homology ≠ common ancestor

Homology ⊢ common function

common structure

Detective work:

If one protein/gene has a known function in a model organism and the other gene...
is a human gene
then we infer that
gene 2 in humans
has a similar
function

---

**Natural Selection:**

Mutations happen "randomly" involved in evolution

Natural Selection = mutations

that preserve function

and folding are accepted of proteins
The mutations that destroy function are eliminated.

Mutation & selection can create new proteins with new function with little change!

Similarity & homology not always imply
There are sequences (genes) with very little similarity which have the same fold and therefore function.

Low seq. similarity → different function or no homology

↑ convergent evolution
Example of convergent evolution

Wings of insects
Wings of bats

Conclusion:

An alignment of two sequences is a start, a "hypothesis" about which pairs of amino acids evolved from a common ancestor.
there is a common ancestor letter \( c_k \)

Determine work follows to see whether the alignment is random
or it implies homology between the two seqs.

EASIER TO DETECT
Homology when comparing protein sequences than DNA seq.

3 reasons:

1. 20 vs. 4 letter

2. Genetic code

3. Protein structure
Comparing DNA sequences is also very important:
- search for regulatory regions of proteins
- whole-genome comparison
- gene identification in the genome