1. **Interfacing representations:** Working with natural language is all about *mapping between levels of representation*. Example:

\[
\begin{align*}
\text{FLY} + \text{PLURAL} & \quad \text{Abstract morphemic representation} \\
\downarrow & \\
\text{fly} + \text{s} & \quad \text{Orthographic representation, split morpheme by morpheme} \\
\downarrow & \\
\text{f}+\text{i}+\text{y}+\text{s} & \quad \text{Orthographic representation, letter-by-letter} \\
\downarrow & \\
\text{f}+\text{l}+\text{i}+\text{e}+\text{s} & \quad \text{Orthographic representation after spelling rule has been applied}
\end{align*}
\]

Each adjacent pair of levels has an **interface** that we need to characterize. More generally, we’ll ultimately want to specify interfaces to other levels of representation, too: phonological, phonetic, syntactic, semantic...

2. **Goal:** a general computational architecture that can automatically map back and forth among these levels of representation. That is a central part of computational linguistics!

3. For a wide variety of linguistic levels, we can use **finite-state transducers (FSTs)** to specify the mapping at the interface between levels. Examples

- Abstract morphemic to orthographic morphemic:

  ![](image1.png)

- Orthographic morphemic to orthographic letter-by-letter:
• Orthographic letter-by-letter to post-spelling-rule orthographic letter-by-letter:

This FST looks scary, but it’s really just a fully explicit version of this FST from last week!
4. We could run an input string through each level of representation sequentially.

5. But, beautifully and usefully, it is possible to compose FSTs. Suppose FST $A$ has input alphabet $\Sigma$ and output alphabet $\Gamma$, and FST $B$ has input alphabet $\Gamma$ and output alphabet $\Delta$. Then there is an FST $A \circ B$ with input alphabet $\Sigma$ and output alphabet $\Delta$ such that if $A$ accepts $\langle \alpha, \gamma \rangle$ and $B$ accepts $\langle \gamma, \beta \rangle$, then $A \circ B$ accepts $\langle \alpha, \beta \rangle$.

6. **Constructing the composition of two FSTs.** Here’s the general method for FSTs $A$ and $B$:

   (a) For every possible state pair $q_i$ in $A$, $q_j$ in $B$, put a state $q_{ij}$ in $A \circ B$.

   (b) $q_{00}$ is the start state of $A \circ B$.

   (c) $q_{ij}$ is a final state in $A \circ B$ if $q_i$ is final in $A$ and $q_j$ is final in $B$.

   (d) If $A$ has an $q_i \to q_i'$ transition labeled $x : y$ and $B$ has a $q_j \to q_j'$ transition labeled $y : z$, then $A \circ B$ has a $q_{ij} \to q_{ij}'$ transition labeled $x : z$.

   (e) If $A$ has an $q_i \to q_i'$ transition labeled $x : \epsilon$, then for every state $j$ in $B$, $A \circ B$ has a $q_{ij} \to q_{ij}'$ transition labeled $x : \epsilon$.

   (f) If $B$ has a $q_j \to q_j'$ transition labeled $\epsilon : z$, then for every state $i$ in $A$, $A \circ B$ has a $q_{ij} \to q_{ij}'$ transition labeled $\epsilon : z$.

7. **Example of FST composition.** Let’s consider composing the abstract morphemic to orthographic morphemic FST with the Orthographic morphemic to orthographic letter-by-letter FST—call them $A$ and $B$ respectively:
Convince yourself that this will accept fly +PL as input and return flys#.

8. Making FSTs more compact. As you can no doubt see, this composed FST is not optimal—many states can never be reached from the start state, and thus might as well not be in the automaton. Fortunately there are some additional fundamental operations at our disposal:

- Composition can create pure $\epsilon$-transitions—cases where the label of an arc on the FST is $\epsilon : \epsilon$. The operation of $\epsilon$-removal gets rid of those.
- It’s possible to convert any FST to an equivalent FST with a minimal number of states. Doing this is the minimization operation.

So we can automatically compose the two FSTs above and then minimize the result, giving us:

9. Complete composition. Here’s the result of composing all three levels:

10. More complex inputs. Here’s a richer morphemic input—composing will automatically do the right thing!