Introduction

How can a computer process words? It must take a word and be able to divide it into its meaning-giving parts. What is a meaning-giving part? When looking at the word “houses” provides us with two types of information. First, there is the obvious part “house”. This gives the word its main meaning, denoting a building people live in. The second part is the plural “s”. When reading the word “houses” the reader instantly knows that this is about more than one house. How can we make the computer understand this? It must undertake a morphological analysis of the word.

The computer first needs a lexicon where all the base words, called “stems”, are stored in. These stems can be combined with “affixes”, word-pieces that can change the stem’s meaning and grammatical function. They can be put in before the stem (“prefixes”) or behind it (“suffixes”). Naturally, we also need a lexicon storing the affixes.

When we have a closer look at the combinations of stems and affixes we notice that not all stems can be combined with all affixes. For example, we can combine the stem “pain” with the suffix “ful”, but we cannot combine “house” with it, since “houseful” is not a word. Thus, we also need a set of rules concerning which affixes can be combined with which word.

How can we implement this in an algorithm? We use Finite State Transducers (FSTs). They allow us to go through the word and give an output for each letter input. Formally, a finite transducer *T* is a 6-tuple (*Q*, Σ, Γ, *I*, *F*, δ) such that:

* *Q* is a [finite set](https://en.0wikipedia.org/index.php?q=aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnL3dpa2kvRmluaXRlX3NldA), the set of *states*;
* Σ is a finite set, called the *input alphabet*;
* Γ is a finite set, called the *output alphabet*;
* *I* is a [subset](https://en.0wikipedia.org/index.php?q=aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnL3dpa2kvU3Vic2V0) of *Q*, the set of *initial states*;
* *F* is a subset of *Q*, the set of *final states*; and
* {\displaystyle \delta \subseteq Q\times (\Sigma \cup \{\epsilon \})\times (\Gamma \cup \{\epsilon \})\times Q}TODO FORMULA
* (where ε is the [empty string](https://en.0wikipedia.org/index.php?q=aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnL3dpa2kvRW1wdHlfc3RyaW5n)) is the *transition relation*.

Using these transducers we implement three levels in order to be able to analyze the morphological composition of a word.

First, the **lexicon** level. This is where all the stems are being represented as an FST. Its end state connects to the **morphotactical** level. This level defines all the rules about how stems can be combined with affixes. It can be very complex, since a word can take more than one affix, and the order of the affixes is important. The third level concerns itself with *regular spelling changes*. These are rules about how a word’s spelling changes when combined with affixes. For example, the word “try” in third person singular is “tries”. You cannot simply add an “s” to the stem, but have to replace the “y” with “ie”. The transducer can go in two directions. Starting with a real word like “houses” it should give us its morphological parts: “house+N+Pl”. On the other hand, given the input “house+N+Pl”, we can also get “houses” as output. Implementing such a morphological analyzer is our goal. Our morphological analyzer still has some weaknesses though. Especially starting with a real word and breaking it down into its individual morphological pieces was problematic for us. Since we wrote our own FST class we could not implement the whole range of functions a real FST should be able to do, making our FST incapable of going the reversed direction easily.