Compound Analysis in FM
Work Paper

Markus Forsberg
Språkbanken, Department of Swedish Language
University of Gothenburg
markus.forsberg@gu.se

We will describe the compound algorithm of Functional Morphology and suggest a novel improvement.

The first step in the compound analysis of Functional Morphology is $deconstruct$, which deconstructs the input string into a set of sequences of forms, where the forms are word forms in our lexicon. The lexical lookup occurs in the function $lexical\_prefix$, which divides the string into all prefixes occuring in the lexicon together with their suffixes. The suffixes are recursively deconstructed.

If we have sandhi-like phenomena in the word boundaries, e.g., the *three consonant rule*[1] in Swedish, then $lexical\_prefix$ would be responsible of generating the candidates that the phenomena predict.

$$\begin{array}{lcll} deconstruct([]) & = & \{[]\} & \\ deconstruct(s) & = & \{form : forms \mid & (form, suffix) \in lexical\_prefix(s) \wedge \\ & & & forms \in deconstruct(suffix)\} \end{array}$$

The function $deconstruct$ is typically overgenerating, so in the next step we use the function $validate$ to filter out impossible compounds. Note, if alphabetic characters are elements in the lexicon, we end up with a combinatorial explosion by using $deconstruct$. For this reason, we normally push the test $validate$ into the generator $deconstruct$, or, as is the case in Functional Morphology, use a lazy language.

$$compound(s) = \{c \mid forms \in deconstruct(s) \wedge c \in validate(forms)\}$$

It is the $validate$ function we will focus on here, where we start with describing the current approach of Functional Morphology. Every morphosyntactic description in Functional Morphology are associated with a compound attribute integer value, encoding where a form with that morphosyntactic description may occur in a compound.

[1] The rule concerns Swedish compound formation: if a compound boundary consists of three consonants that are the same, then these consonants are reduced into two consonants, e.g., glass + skål → glasskål (Eng. 'ice cream bowl'). This reduction introduces ambiguity, since we have glass+skål and glas+skål (Eng. 'glass bowl'), and $lexical\_prefix$ is responsible of generating both candidates.

Naturally, a form may have many descriptions, and consequently, many compound attribute values, since forms are homographically ambiguous.

$$validate(c) = \{c \,|\, x \in attributes(c) \wedge valid\_sequence(x)\}$$

The function $attributes$ generates all possible compound parameter sequences, and $valid\_sequence$ is a boolean function that describes which sequences are considered possible.

The algorithm works efficiently, but the description of compounds is unsatisfactory: it is highly inflexible, we do not use all information that is actually available, and the use of integers to encode compounding behaviour is simply error-prone and unaesthetic.

Let us start by looking at what information we have available in an analysis in Functional Morphology, here for the Swedish word *ankans* (Eng. *the duck's*), which only has one analysis. The eight fields have the following meaning: *wf* is the current word form, *cf* its citation form, *pos* its word class, *msd* its morphosyntactic description, *inhs* its inherent feature, *lid* its lemma id, *pid* its paradigm identifier, and *attr* its compound attribute.

wf	ankans	*cf*	anka
pos	nn	*msd*	sg def gen
inhs	u	*lid*	anka..nn.1
pid	nn_1u_flicka	*attr*	0

We can now ask us the question — do we actually need the $attr$ field, why not have compound rules based on all the other fields? This idea will be further pursued in the rest of the text.

For example, we may have a rule for Swedish noun compound looking like this, where words tagged *ci* are compound forms that may appear in initial positions and words tagged with *cm* are compound forms that may appear in medial positions.

$$\text{rule noun } [pos = nn] = \{msd = ci\}\{msd = cm\}^*\{msd \notin \{ci, cm\}\}$$

The name of the rule is noun, and the association *pos=nn* enclosed in square brackets is common for all patterns after the equal sign. The first pattern requires that the first word form has *msd=ci*, the second pattern requires zero or more word forms with *msd=cm*, and the third and last pattern requires a word form that has a *msd* which is not *ci* or *cm*.