Generalized Swap Grammar

by Sven Nilsen, 2018

When creating a grammar for encoding sets, it is common to define swap the following way:

a(bc)d = abcd + acbd

The following generalizations can be made for encoding sub-sets of permutations:

a[bc]d = abdc + acdb
a[bc)d = abcd + acdb
a(bc]d = abdc + acbd

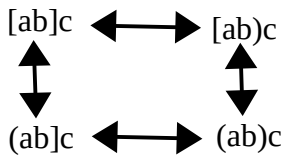
Intuitively, one can think of `[bc)` as an arrow where starting with `b` means `c` follows next.
Similarly, one can think of `(bc]` as an arrow where starting with `c` means `b` follows next.
When these arrows are combined, one gets `(bc)`.

The bracket `[bc]x` can be thought of as “jump out to `x` and then back to the other”:

[bc]x = bxc + cxb

Generalized swap grammar is used to express some sub-sets of permutations more efficiently:

[ab]c = acb + bca
[ab)c = abc + bca
(ab]c = acb + bac
(ab)c = abc + bac



Notice that all rules have two “neightbors” which share one sentence in common.

The only permutations of 3 letters that are not part of the above rules are:

cab cba

This is because `c` is before both `a` and `b`. One can obtain these by swapping `c` with `b`.
When reordering inside the bracket is permitted, there is only need for a single “arrow”:

[ab)c = (ba]c

One can also create a language that picks letters from swap rules forwards and backwards:

(si)(ed)a = sieda + sidea + iseda + isdea
[si][ed]a = seadi + sdaei + ieads + idaes
[si)(ed]a = siead + sidea + ieads + ideas

In this language, every letter occurs once and the word has fixed length, like permutations.