Sub-Permutation Grammar

by Sven Nilsen, 2018

In this paper I develop a sub-permutation grammar from the notation of generalized swap grammar. This has applications in permutative path semantics, which reasons about sub-types of permutations.

In grammar for sub-sets of permutation, it is common to write the following for sub-permutations:

a(bcd)e = abcde + abdce + acbde + acdbe + adbce + adcbe

In “Generalized Swap Grammar”, I extended this notation of 2 letters with `[]`, `[)` and `(]`.
In this paper I will build on this further to more than 2 letters, a sub-permutation grammar.

The first thing I will do, is change the interpretation of `()` for more than 2 letters.
In this sub-permutation grammar, I will use a `!` after the rule for sub-permutations.
This is consistent with the factorial notation, which counts the members of sub-permutations:

a(bcd)!e = abcde + abdce + acbde + acdbe + adbce + adcbe

|(bcd)!| = |(bcd)|! = 3! = 3 · 2 · 1 = 6

Instead, `()` without the `!` is interpreted as “forward then backward”:

a(bcd)e = abcde + adcbe

For `()` with 2 letters, the these two interpretations are the same:

(ab) = (ab)! = ab + ba

The motivation for this is, when defining `[]` and `[)` for more than 2 letters, the following holds:

|[)!| >= |(]!|
|[)!| >= |[]| = |[]!| = |()!| = |[]| <= |(]!|
|[]| >= |[)| |(]| <= |[]|
|[)| >= |()| |()| = 2 |()| <= |(]|

This set of rules is called “two in the house” and makes it easy to remember the relative number of members: At the floor, one compares the smallest sub-grammars. At the roof, one compares the sub-grammars holding at least `N!`, but the pipe gets even higher than the roof.

The “two in the house” rule holds for sub-grammars of same number of letters. There is a “house” for every number of letters, which can be thought of a city where 2 people live in every house.
With other words, a “sub-permutation city”.

The exception of course, is rules with 1 or 0 letters, which has 1 and 0 members respectively, regardless of how one uses `[a]`, `[a)`, `(a]`, `(a)` or `[a]!`, `[a)!`, `(a]!`, `(a)!`.

In nested grammars of generalized swap grammar, such as `[ab][cd]`, one can imagine it as going forward through the rules and then backward, picking up letters that has not been picked up before.

For the more general grammar of sub-permutations, one can think of it as swiping forward and backward until all letters are picked up, or transforming the grammar into sub-grammars:

a[bcd]e =
abe[cd] + ace[bd] + ade[bc] =
abecd + abedc + acebd + acedb + adebc + adecb

Swiping forward and backward is the same as placing the unpicked sub-grammars at the end.

Notice that in this case, adding `!` after the rule produces the same result:

a[bcd]!e = a[bcd]e

The interpretation of the arrow `[)` is similar, by removing the picked letters and appending the rest:

a[bcd)e =
abcde + acde[b) + ade[bc) =
abcde + acdeb + adebc + adecb

By adding `!` after the bracket, one permutates all the sub-grammars:

a[bcd)!e = a[bcd)e + a[bdc)e + a[cbd)e + a[cdb)e + a[dbc)e + a[dcb)e
a[bcd)e = abcde + acde[b) + ade[bc) = abcde + acdeb + adebc + adecb +
a[bdc)e = abdce + adce[b) + ace[bd) = abdce + adceb + acebd + acedb +
a[cbd)e = acbde + abde[c) + ade[cb) = acbde + abdec + adecb + adebc +
a[cdb)e = acdbe + adbe[c) + abe[cd) = acdbe + adbec + abecd + abedc +
a[dbc)e = adbce + abce[d) + ace[db) = adbce + abced + acedb + acebd +
a[dcb)e = adcbe + acbe[d) + abe[dc) = adcbe + acbed + abedc + abecd

Notice that this gives fewer members than permutation of 4 letters,
because some members are redundant, but more members than permutation of 3 letters.
It also turns out that `a(bcd]!e` is a sub-set of `a[cbd)!e`:

a(bcd]!e ⊊ a[cbd)!e

a(bcd]e =
abe(cd] + ace(bd] + adcbe =
abecd + abedc + acebd + acedb + adcbe

a(bcd]!e = a(bcd]e + a(bdc]e + a(cbd]e + a(cdb]e + a(dbc]e + a(dcb]e
a(bcd]e = abe(cd] + ace(bd] + adcbe = abecd + abedc + acebd + acedb + adcbe
a(bdc]e = abe(dc] + ade(bc] + acdbe = abedc + abecd + adebc + adecb + acdbe
a(cbd]e = ace(bd] + abe(cd] + adbce = acebd + acedb + abecd + abedc + adbce
a(cdb]e = ace(db] + ade(cb] + abdce = acedb + acebd + adecb + adebc + abdce
a(dbc]e = ade(bc] + abe(dc] + acbde = adebc + adecb + abedc + abecd + acbde
a(dcb]e = ade(cb] + ace(db] + abcde = adecb + adebc + acedb + acebd + abcde