# Patterns in Combinator Evolution

**Eric James Parfitt**
*eparfitt84@gmail.com*

Rules for speeding up SK combinator evaluation were investigated, and experiments were performed to determine the proportion of SK combinator expressions that reach a fixed point before exceeding certain computational limits. It was found that approximately 80.3 percent of SK combinator expressions with size 100 reach a fixed point within 300 steps and also without having reached a combinator size of 200 000 sometime during evaluation.

## 1. Introduction

Combinators are constructs that can be understood as functions in a certain abstract sense [1] and that are useful as a simplified model of computation. There are two very interesting combinators called S and K, which can be used together to write any computer program (the two together are Turing complete, so can be used as a universal language).

To work with these combinators, we can look at transformations of combinator expressions. This involves starting out with an initial combinator expression, which is made by calling S and K combinators on each other in some order. Once an SK combinator expression has been generated, we apply the rules for evaluating the combinators repeatedly. The rules are to replace K[x_][y_] with y, and S[x_][y_][z_] with x[z][y[z]]. So for example, we start out with the expression S[S[S]][S][K][K], and running it step by step, we would get

```
S[S[S]][S][K][K]
S[S][K][S[K]][K]
S[S[K]][K[S[K]]][K]
S[K][K][K[S[K]][K]]
K[K[S[K]][K]][K[K[S[K]][K]]]
K[S[K]][K]
S[K]
S[K]
```

The evaluation "halts" when applying the rules no longer changes the expression, so the "output" of S[S[S]][S][K][K] is S[K].

One detail that was glossed over in the previous explanation is the order in which rules are applied to the expression. There are two standard ways in which to apply the rules, normal order and applicative

https://doi.org/10.25088/ComplexSystems.26.2.119

order. In applicative order, the rules are applied at each step by scanning once from left to right and applying the rule wherever possible without overlapping. This leads to larger parts of the combinator having the rules applied to them first. In normal order, the rules are again applied from left to right without overlapping, but the parts deepest inside brackets (the arguments of the functions) are evaluated before the outer parts.

Normal order will be used exclusively in this paper. It turns out that there are often combinator expressions that reach a fixed point in normal order but not in applicative order. This is because sometimes there will be a subpart of a combinator that would continuously expand if evaluated over and over, but if that subpart is inside a K combinator, evaluating the K function before the arguments deletes the whole expression that was going to grow before it could begin its continuous growth. For example, S[S[S]][S][S][S][S] never stops growing [2], so evaluating K[S][S[S[S]][S][S][S][S]] in applicative order will never stop growing either, whereas evaluating the same in normal order will cause the expression to become just S and to then stop changing, after just one step. In this case, the K simply deletes the second part containing the potential to grow.

In general, some combinator expressions reach a fixed point and stop changing, which is referred to as "halting," whereas some never stop growing, or begin to continuously loop through a set of states.

This paper discusses how certain patterns in combinator evaluation were discovered. Extra combinator rules based on these patterns were used in attempts to speed up the evaluation of large combinators. Also, experiments were performed to determine the proportion of SK combinator expressions that reach a fixed point before exceeding certain computational limits.

In [3], the authors created databases of combinator reductions to speed up computation. Here a slightly different approach to finding extra combinator rules for speeding up combinator reductions is used. In [3], a sort of "multiplication table" was built up of different ways combinators can evaluate, whereas in this paper, combinators up to a certain size were evaluated with different combinator parts being replaced by variables, as described in Section 2.

## 2. Patterns in Combinator Evolution

Out of all possible SK combinator patterns up to size five (of which there were 2582), just 13 are non-redundant as defined in a certain way. These 13 rules are rules in which evaluating the left side to get the right side takes more than just one step. Also, less general rules are excluded in favor of more general rules (for instance, the more general S[K][y_][x_] :> x is included but not the more specific S[K][S][x_] :> x). Also, cases where applying a shorter combinator rule has the same effect as applying a larger combinator rule are not included. The rule S[S[K][y_][x_]] :→ S[x] is not included, since the shorter rule S[K][y_][x_] :→ x applied to S[S[K][y_][x_]] will also give S[x] in one step. The 13 non-redundant rules, which are general ways in which combinators up to size five evaluate, are the following:

```
K[x_][y_] :→ x
S[x_][y_][z_] :→ x[z][y[z]]
S[K][x_][y_] :→ y
K[K[x_][y_]][z_] :→ x
K[K[x_]][y_][z_] :→ x
K[K][x_][y_][z_] :→ y
S[x_][K[y_]][z_] :→ x[z][y]
S[K[x_]][y_][z_] :→ x[y[z]]
S[S][x_][y_][z_] :→ y[z][x[y][z]]
S[S[x_]][y_][z_] :→ x[y[z]][z[y[z]]]
S[S][x_][K][y_] :→ y
S[S[K]][x_][y_] :→ x[y]
S[S][K][x_][y_] :→ x[y][x]
```

There are 22 994 possible rules of size six, and again a relatively small number, 67, are non-redundant in this way.

## 3. Applying Optimization

Next there is the matter of applying the rules that were found to combinator expressions. Combinators of various sizes were randomly generated. These combinators were then evaluated, while adding extra rules to see if this sped up the combinator evolution. It is expected that the more steps and the more often rules are applied, the more extra rules would help speed up evolution. Large combinators generally have more rule applications and run for more steps. Larger and larger combinators were tested to look for any speedups.

There is one issue with using large combinators, however, which is discussed more in Section 4. Basically, the issue is that larger combinators were more likely not to reach a fixed point within 300 steps. Generally, only combinators that reach a fixed point were of interest for testing optimizing rules, since the extra rules would likely be used primarily for evaluating halting combinators. (If we were looking at nonhalting combinators, we would presumably want to look at step-by-step evolution, and the extra rules would likely be skipping steps in unpredictable increments.)

Based on some preliminary experimenting into how large combinators evolved, combinators that seemed unlikely to halt in two ways were first analyzed. Combinators that did not halt after a certain number of steps and combinators that did not halt before growing to a certain size were selected. As elaborated more in Section 4, larger combinators do seem to either quickly begin to grow exponentially or to quickly begin evolving in a repetitive way. A small minority of combinators that did neither of these things, but instead grew non-exponentially, but in a complex way for many steps, were ignored. (These are rather interesting combinators; however, it seems especially hard to predict what they will do in the future: whether they will continue having complicated slow growing behavior, whether they will begin growing exponentially at some point, or whether they will in the end exhibit repetitive behavior after all.)

However, even with combinators of size 1000, the extra rules only modestly sped up evaluation. With only S and K rules, Mathematica took an average of 1.20 seconds each to repeatedly evaluate 200 randomly generated size 1000 combinators (these were also selected as combinators that reached a fixed point before reaching either 10 000 iterations or a size of 10 000). With the extra 11 rules, this took 1.13 seconds, and with only one extra rule, S[K][x_][y_] :→ y, this took 1.06 seconds. All timing measurements here were performed with Mathematica 11 on a 3.40 GHz Intel Core i7-6700 processor.

Different selections and orders of the 11 optimization rules found, along with the original S and K rules, were tested to see which would speed up evaluation the most. First, each extra rule was added one at a time, and the added rule was added at almost each possible index in the rule list. Some positions were not checked. For example, putting the extra rule S[K][x_][y_] :→ y after S[x_][y_][z_] :→ x[z][y[z]], as opposed to before it, would mean the extra rule would not be used. The extra rule would be skipped over by the original and more general rule. The fastest rule permutation turned out to be {K[x_][y_] :→ x, S[K][x_][_] :→ x, S[x_][y_][z_] :→ x[z][y[z]]}, which took an average of 1.00 second to evaluate the same combinators as before.

This is interesting: since S[K][x_][y_] :→ y is equivalent to the identity or "I" combinator, (I[y_] :→ y, I is essentially the same as S[K][x_]), which can be used in practice to simplify the use of SK combinators. Extra rules were added in all possible combinations. Two, three and then four different rules were added. These are the fastest rules in each of those situations, with rules on the left and time in seconds on the right of each list:

```
RepeatedTiming[a /. {b :→ c}]
{9.8 × 10^-7, a}
RepeatedTiming[a /. {b :→ c, d :→ e}]
{1.4 × 10^-6, a}
RepeatedTiming[a /. {b :→ c, d :→ e, e :→ f}]
{2.02 × 10^-6, a}
```

The extra rules can be used for speeding up combinator evaluation. However, there is a tradeoff when using them, as more rules can help in skipping steps, but extra rules also generally slow down evaluation. This is due to the interpreter needing to scan for more rule matches.

With only S and K rules, Mathematica took an average of 1.20 seconds each to repeatedly evaluate 200 randomly generated size 1000 combinators that were also selected as combinators that reached a fixed point before reaching either 10 000 iterations or a size of 10 000. With the extra 11 rules, this took 1.13 seconds, and with only one extra rule, S[K][x_][y_] :→ y, this took 1.06 seconds.

When only one to four extra rules were used, the rule selection and order that were fastest for each were found to be the following:

```
extra1 → {K[x_][y_] :→ x, S[K][x_][y_] :→ y, S[x_][y_][z_] :→ x[z][y[z]], 1.}
extra2 → {S[K[x_]][y_][z_] :→ x[y[z]], K[x_][y_] :→ x, S[K][x_][y_] :→ y,
     S[x_][y_][z_] :→ x[z][y[z]], 0.959}
extra3 → {S[K[x_]][y_][z_] :→ x[y[z]], K[K][x_][y_][z_] :→ y, S[K][x_][y_] :→ y,
     S[x_][y_][z_] :→ x[z][y[z]], K[x_][y_] :→ x, 0.79}
extra4 → {S[K][x_][y_] :→ y, K[K][x_][y_][z_] :→ y, K[K[x_]][y_][z_] :→ x,
     K[x_][y_] :→ x, S[K[x_]][y_][z_] :→ x[y[z]], S[x_][y_][z_] :→ x[z][y[z]], 0.765625}
```

Interestingly, when looking at all possible rule sets of a given size, the fastest always has the same elements as the rule set one size smaller, but with one new rule added, and in most cases in a different order. Also, the one extra rule that is in all of these fastest rule selections is S[K][x_][y_] :→ y, which again is equivalent to the identity, or "I" combinator I[x_] :→ x.

In the graphs in Figure 1, the $x$ axis represents rules in the order that they show up using the Permutations function. The $y$ axis shows the average time in seconds for evaluating sets of large combinators.

When one of each extra rule was added to extra4, the rule with a fifth extra rule that was fastest was

```
{S[K][x1_][x2_] :→ x2, K[K][x1_][x2_][x3_] :→ x2, S[S][x1_][K][x2_] :→ x2,
K[K[x1_]][x2_][x3_] :→ x1, K[x_][y_] :→ x, S[K[x1_]][x2_][x3_] :→ x1[x2[x3]], S[x_][y_][z_] :→
x[z][y[z]]},
```

which took 0.915 seconds to go through all the combinators. This rule set was slower than extra4.

To speed up finding rules of size 7, only rules similar to the fastest rule of size 6, but with one new rule part added, were tested.

The fastest of these, however, was slower than the fastest rule set that has one less rule. The rule was

```
{S[K][x1_][x2_] :→ x2, K[K][x1_][x2_][x3_] :→ x2, S[S][x1_][K][x2_] :→ x2,
K[K[x1_]][x2_][x3_] :→ x1, K[x_][y_] :→ x, S[K[x1_]][x2_][x3_] :→ x1[x2[x3]], S[x_][y_][z_] :→
x[z][y[z]]},
```

which took 0.915 seconds to go through all the combinators. This could be due to the fact that extra rules will generally significantly slow down evolution, if the rule has no effect of skipping steps in evaluation.

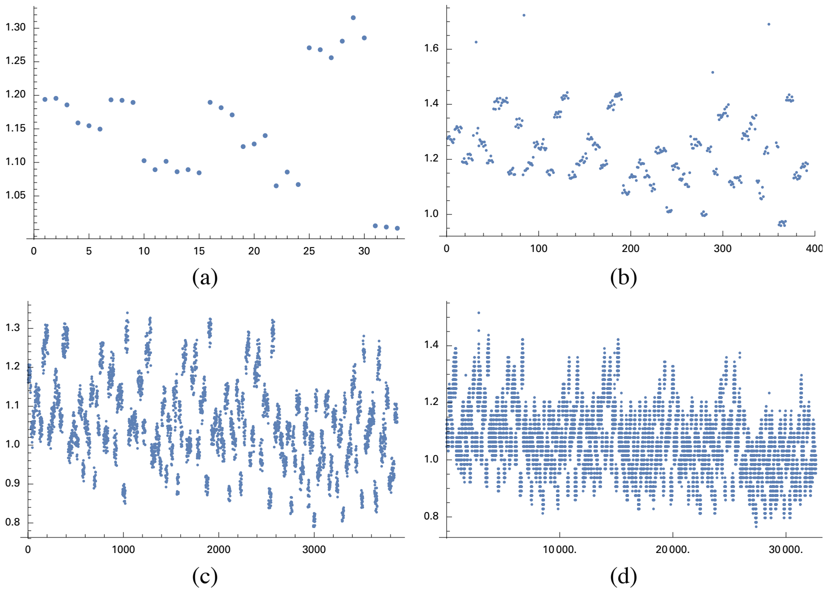


**Figure 1.** Timing versus permutation index of implementing combinator rules. This shows how different rules speed up or slow down the evaluation of large combinators. The $x$ axis represents rules in the order that they show up using the Permutations function. The $y$ axis shows the average time in seconds for evaluating sets of large combinators. (a) An extra optimization rule is added in different ways to the standard S and K rules. (b) Two extra rules are added. (c) Three extra rules are added. (d) Four extra rules are added.

Thus, extra rules can only help in the case of large combinators, and it appears that only up to a certain point do extra rules continue to speed up evolution. Taking into consideration that extra rules had a modest effect on speeding up evolution of size 100 combinators, and that extra rules would slow down the evolution of smaller combinators, in the end only the combinator rule that corresponds to the "I" combinator was added to the actual EvaluateCombinator function.

## 4. Random Long-Running Combinators

The way in which large combinators evaluated was investigated in order to learn more about why the rules found only had a modest effect on the speeding up of combinator evolution.

First, the evolution of 400 randomly generated size 100 combinators was run, and most of these combinators quickly did one of two things. They either began growing exponentially or began having repetitive behavior. Only one, in fact, appeared to be doing neither.

First of all, it is known that two of the 16 896 possible size 7 combinators do not reach fixed points, and these combinators grow exponentially [2]. Also, "At size 8, out of all 109 824 combinator expressions it appears that 49 show exponential growth. And many more show roughly linear growth." [2] It seems that this becomes a trend, with even larger combinators being more and more likely to grow at an exponential rate, at least for a large number of steps. This could be partially due to a larger combinator being more likely to contain a smaller combinator part that grows exponentially.

The behavior of larger combinators that do not halt before reaching certain computational limits was investigated. Four hundred random size 100 SK combinators were generated, using `SeedRandom[1]` as a starting seed for the random combinator generator. These combinators were then evaluated, with evolution pausing when one of three outcomes happened: (1) the combinator evolution reached a fixed point (no longer changed with replacement rules applied); (2) the combinator evolution reached 300 steps; or (3) the size of the combinator went above 2000.

Combinators that led to criterion 1 (reached a fixed point) were filtered out, leaving only combinators that took many steps and/or became very large without halting. Only 66 of the original 400 randomly generated combinators had not halted or had grown too large by this number of steps. These 66 combinators were then evaluated a second time, again for a maximum of 300 steps, but for a maximum combinator size of 200 000, to see if they would halt after reaching this size. Figure 2 shows combinators that did not halt after 300 steps or before reaching size 200 000.

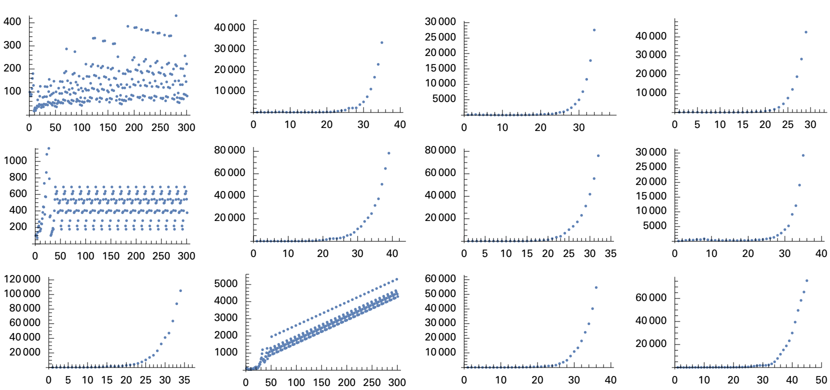


**Figure 2.** (*continues*)

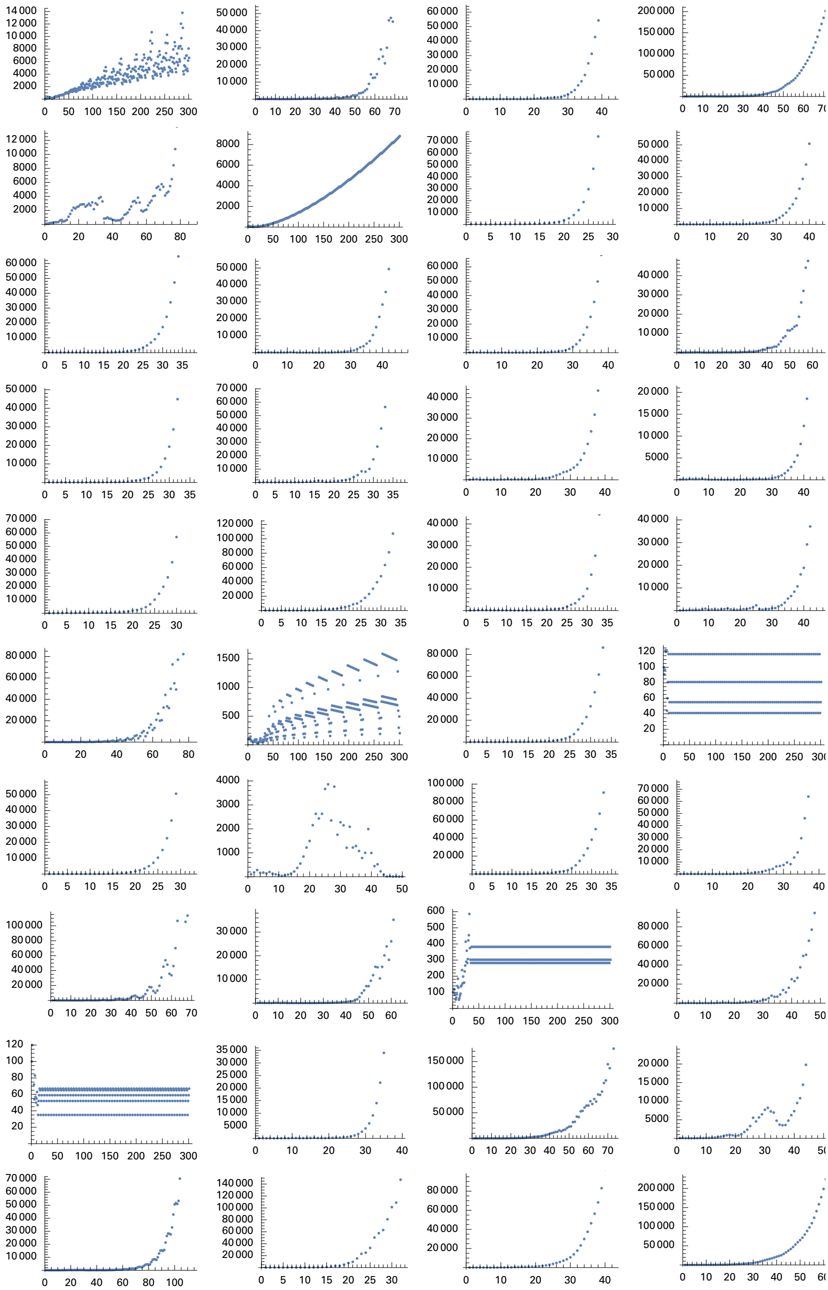

**Figure 2.** (*continues*)

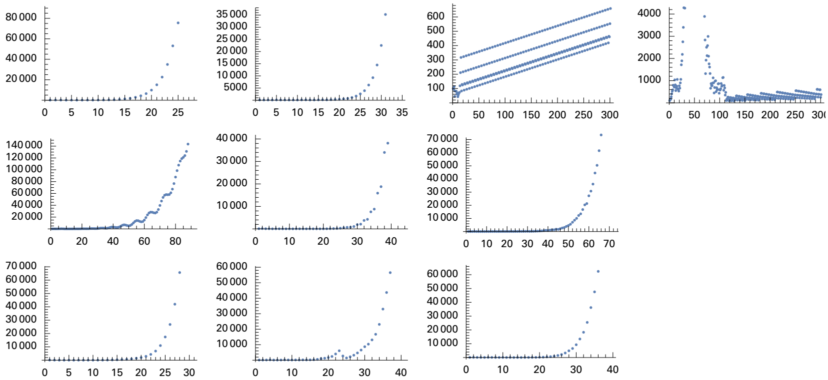

**Figure 2.** Randomly selected non-terminating combinator growth. This shows the growth of randomly selected combinators that were also selected under the criterion that they not terminate after many steps.

Figure 3 shows the same plots, but with the natural logarithm taken of each size point, and then with the differences taken between those. It shows that most of the combinators that do grow exponentially still appear to be exhibiting complex behavior.

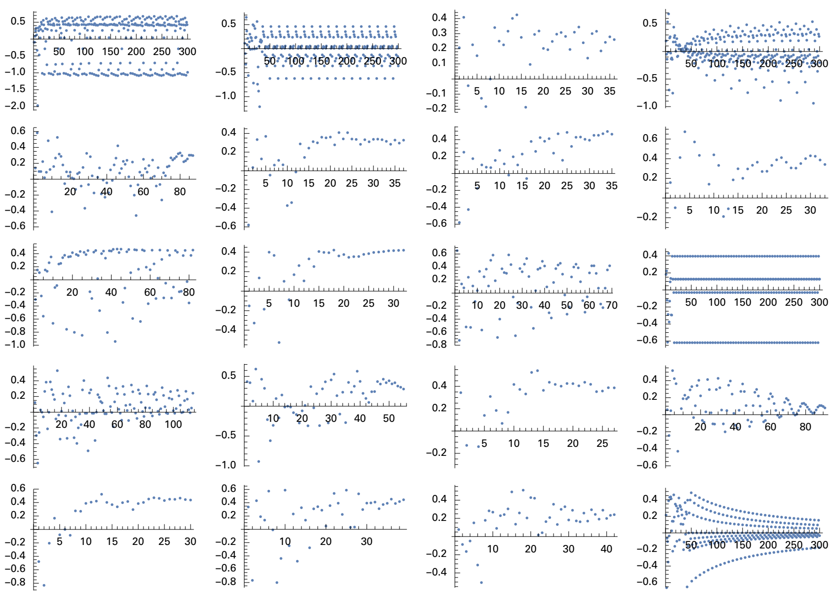

**Figure 3.** (*continues*)

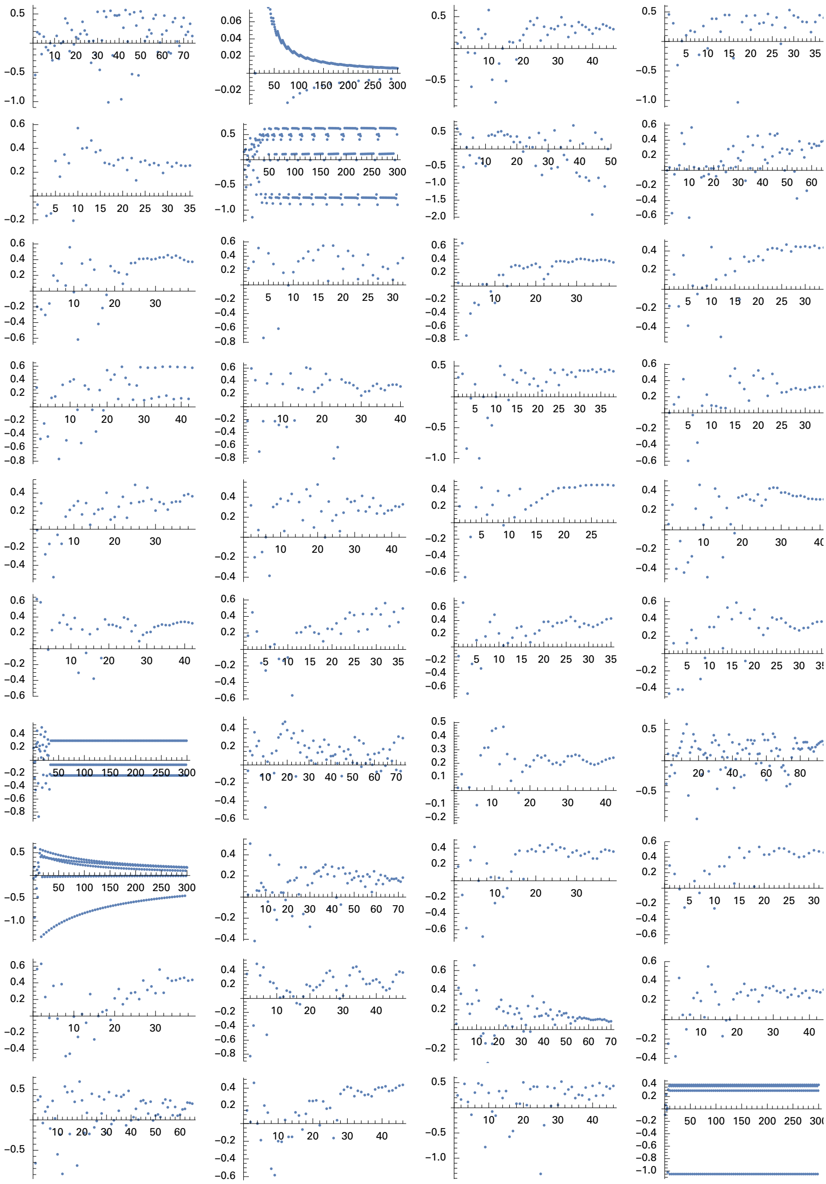

**Figure 3.** (*continues*)

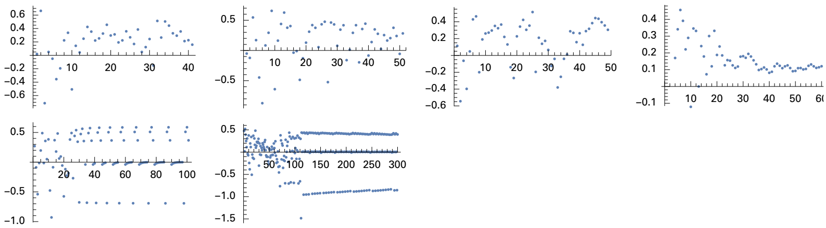

**Figure 3.** Logarithmic differences of randomly selected non-terminating combinator growth. This shows the growth of randomly selected combinators that do not terminate after many steps. The *y* axis shows the logarithms of the differences between combinator lengths from step to step.

Of the 66 combinators, 53 appear to have been selected by going above length 2000 before they reached step 200. One of them got above size 2000 but below 200 000 (it reached a maximum size of 3861), but reached a fixed point anyway before step 300. Four of them got to step 300 by beginning to loop through the same few values repeatedly. Six of them began to grow in what appears to be a repetitive pattern.

Figure 4 shows the two combinators that appeared to be showing complex behavior without growing exponentially, this time with a cutoff of 2000 steps. They both appeared to show complicated behavior without becoming repetitive or growing exponentially, although the behavior may be nested.

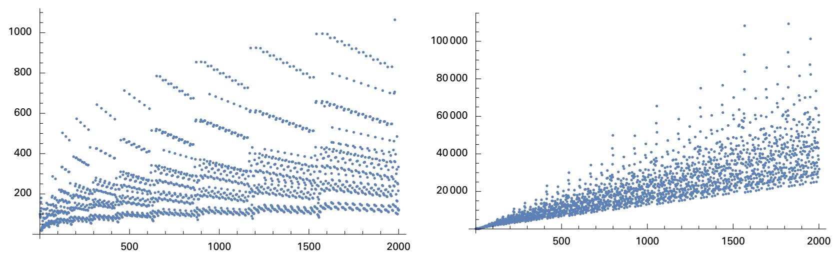

**Figure 4.** Two large combinators that appear to show complicated behavior without becoming repetitive or growing exponentially.

It should be noted, however, that the combinators that appeared to be growing at an exponential rate or that seemed to be exhibiting complex behavior even after many steps or reaching a large size could still potentially reach a fixed point if run for more steps or to larger sizes. However, with the limited computational resources given, this was not observed in most of the large combinators.

## 5. Random Terminating Combinators

Figure 5 shows 62 combinators selected from 200 randomly chosen size 1000 combinators that did terminate. All of the ones that reached a fixed point did so within 100 steps (one ran for exactly 100 steps). The largest size any of them reached at any time during its evaluation was 154 820 (it ran for a total of 44 steps).

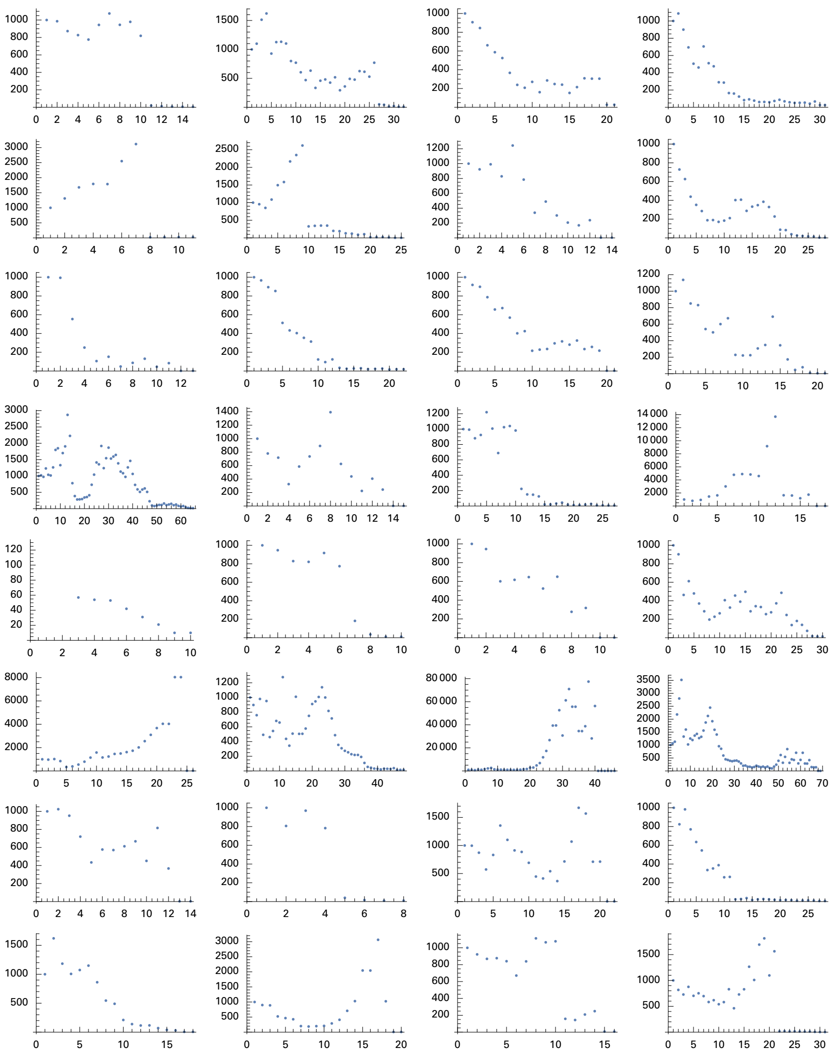

**Figure 5.** (*continues*)

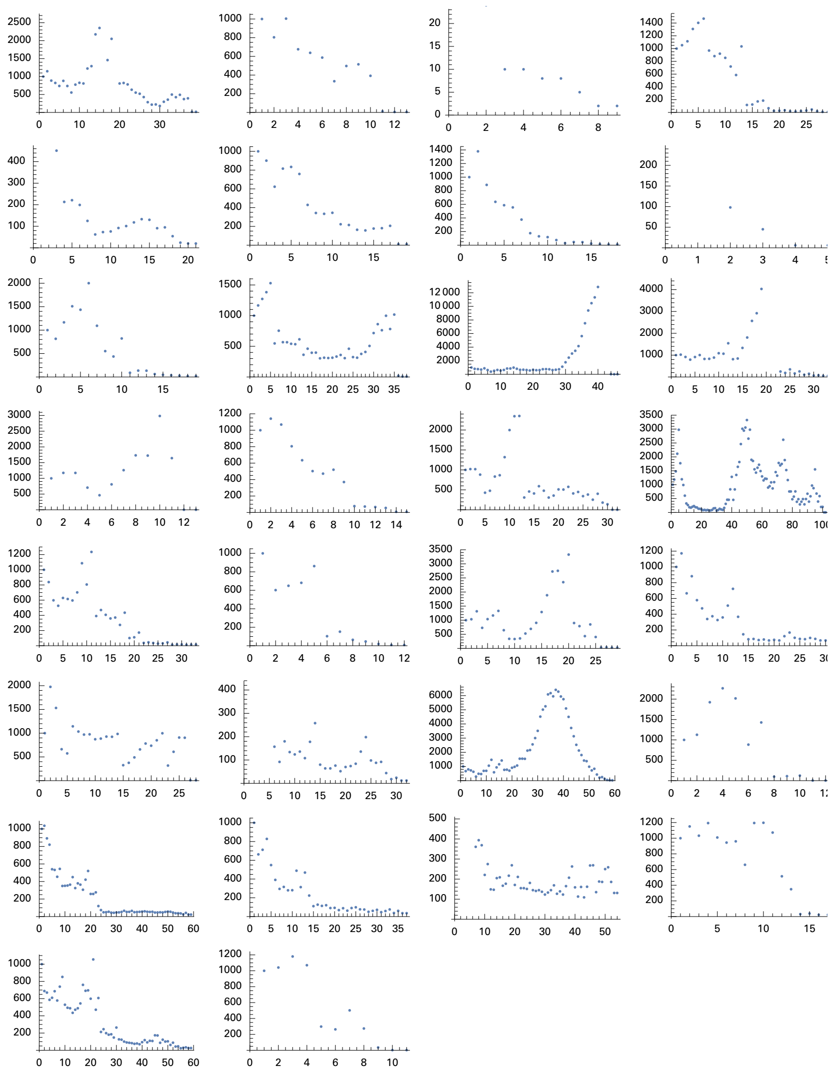

**Figure 5.** Randomly selected terminating combinator growth. This shows the growth of randomly selected combinators that terminate before a given large number of steps.

## 6. Frequency of Optimization Rule Usages

This shows what fraction of the time an extra optimization rule (besides the usual S and K rules) was used during the attempt at optimized evolution. The combinators are sorted by how often the rules were used. Each letter corresponds to a different size 1000 combinator.

{A → 0.0257056, B → 0.024629, C → 0.0246223, D → 0.024045, E → 0.0235916,
F → 0.0225295, G → 0.0222673, H → 0.021963, I → 0.0212766, J → 0.0210058,
K → 0.0209823, L → 0.0209536, M → 0.0209166, N → 0.0208107, O → 0.0206622,
P → 0.0205735, Q → 0.0205672, R → 0.0204877, S → 0.0204736, T → 0.0204562,
U → 0.0202132, V → 0.0200748, W → 0.0200241, X → 0.0199806, Y → 0.0196201,
Z → 0.0193757, A1 → 0.0192907, B1 → 0.0192356, C1 → 0.0191463,
D1 → 0.0190822, E1 → 0.0189619, F1 → 0.0188708, G1 → 0.0188185,
H1 → 0.0185733, I1 → 0.0185015, J1 → 0.0184146, K1 → 0.0183781,
L1 → 0.0183239, M1 → 0.0182556, N1 → 0.018183, O1 → 0.0181257,
P1 → 0.0180765, Q1 → 0.0179722, R1 → 0.0178833, S1 → 0.0173198,
T1 → 0.0171956, U1 → 0.0161381, V1 → 0.0160847, W1 → 0.0159823,
X1 → 0.015799, Y1 → 0.0156495, Z1 → 0.0155909, A2 → 0.0155469, B2 → 0.015438,
C2 → 0.0148621, D2 → 0.0146095, E2 → 0.0141228, F2 → 0.0138126,
G2 → 0.0107118, H2 → 0.00715231, I2 → 0.00564804, J2 → 0.0050268}

Figure 6 labels the combinator optimization rules. Figure 7 breaks down how often each of these labeled rules is used individually in evaluation out of 100 000 and also shows the number of times out of 100 000 when a part of the combinator evolution does not match any rule.

```
a → S[S][x1_][K][x2_] ⧴ x2
b → S[S[K]][x1_][x2_] ⧴ x1[x2]
c → S[S][K][x1_][x2_] ⧴ x1[x2][x1]
d → K[K][x1_][x2_][x3_] ⧴ x2
e → K[K[x1_]][x2_][x3_] ⧴ x1
f → K[K[x1_][x2_]][x3_] ⧴ x1
g → S[x1_][K[x2_]][x3_] ⧴ x1[x3][x2]
h → S[K[x1_]][x2_][x3_] ⧴ x1[x2[x3]]
i → S[S][x1_][x2_][x3_] ⧴ x2[x3][x1[x2][x3]]
j → S[S[x1_]][x2_][x3_] ⧴ x1[x2[x3]][x3[x2[x3]]]
k → S[K][x1_][x2_] ⧴ x2
l → K[x_][y_] ⧴ x
m → S[x_][y_][z_] ⧴ x[z][y[z]]
n → x_ ⧴ x
```

**Figure 6.** Labeling optimization rules for the table in Figure 7.

| | a | b | c | d | e | f | g | h | i | j | k | l | m | n |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | 45 | 93 | 405 | 546 | 45 | 11 | 270 | 138 | 546 | 186 | 286 | 1823 | 808 | 94 798 |
| B | 213 | 225 | 6 | 599 | 33 | 18 | 198 | 189 | 362 | 295 | 325 | 2484 | 888 | 94 165 |
| C | 51 | 87 | 29 | 298 | 414 | 15 | 327 | 247 | 283 | 189 | 523 | 1816 | 1148 | 94 574 |
| D | 12 | 257 | 17 | 82 | 29 | 48 | 149 | 596 | 125 | 580 | 510 | 1485 | 1489 | 94 621 |
| E | 76 | 76 | 84 | 353 | 260 | 126 | 294 | 134 | 269 | 118 | 571 | 2359 | 1192 | 94 089 |
| F | 24 | 48 | 83 | 238 | 191 | 83 | 274 | 215 | 215 | 453 | 429 | 2134 | 1359 | 94 254 |
| G | 78 | 78 | 90 | 90 | 66 | 84 | 394 | 149 | 310 | 173 | 716 | 2298 | 1128 | 94 347 |
| H | 30 | 76 | 45 | 197 | 121 | 76 | 288 | 257 | 151 | 273 | 682 | 2196 | 1621 | 93 987 |
| I | 89 | 113 | 122 | 215 | 85 | 122 | 300 | 178 | 349 | 150 | 405 | 2610 | 1325 | 93 937 |
| J | 21 | 106 | 38 | 458 | 168 | 14 | 322 | 147 | 192 | 144 | 493 | 1796 | 1020 | 95 084 |
| K | 5 | 21 | 72 | 257 | 370 | 26 | 257 | 77 | 216 | 108 | 689 | 2294 | 1353 | 94 256 |
| L | 64 | 64 | 86 | 278 | 107 | 21 | 257 | 150 | 214 | 214 | 641 | 2651 | 1219 | 94 035 |
| M | 7 | 145 | 51 | 232 | 174 | 138 | 254 | 240 | 203 | 196 | 450 | 2142 | 1118 | 94 647 |
| N | 22 | 185 | 115 | 393 | 153 | 70 | 118 | 80 | 221 | 211 | 511 | 2986 | 1132 | 93 802 |
| O | 85 | 109 | 193 | 169 | 97 | 133 | 169 | 230 | 290 | 205 | 387 | 2127 | 1112 | 94 696 |

**Figure 7.** (*continues*)

| | a | b | c | d | e | f | g | h | i | j | k | l | m | n |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P | 16 | 43 | 155 | 187 | 224 | 32 | 197 | 155 | 352 | 165 | 533 | 2473 | 911 | 94558 |
| Q | 27 | 127 | 63 | 136 | 399 | 136 | 263 | 109 | 181 | 190 | 426 | 2600 | 870 | 94473 |
| R | 31 | 13 | 82 | 308 | 195 | 258 | 145 | 176 | 201 | 44 | 597 | 2709 | 1276 | 93967 |
| S | 25 | 99 | 49 | 345 | 173 | 74 | 247 | 222 | 123 | 271 | 419 | 2935 | 1184 | 93833 |
| T | 26 | 22 | 26 | 232 | 318 | 45 | 310 | 90 | 303 | 131 | 542 | 1421 | 1144 | 95389 |
| U | 61 | 104 | 141 | 202 | 37 | 37 | 147 | 202 | 392 | 233 | 466 | 3001 | 1133 | 93844 |
| V | 25 | 157 | 96 | 298 | 253 | 40 | 243 | 162 | 233 | 172 | 329 | 2559 | 895 | 94539 |
| W | 28 | 51 | 79 | 222 | 88 | 97 | 357 | 121 | 334 | 167 | 459 | 2647 | 973 | 94377 |
| X | 6 | 11 | 91 | 194 | 194 | 11 | 360 | 337 | 280 | 6 | 508 | 2546 | 1222 | 94234 |
| Y | 37 | 31 | 112 | 187 | 156 | 93 | 187 | 212 | 336 | 137 | 473 | 1682 | 1046 | 95310 |
| Z | 20 | 20 | 173 | 81 | 243 | 66 | 173 | 234 | 228 | 141 | 557 | 1071 | 1135 | 95856 |
| A1 | 12 | 47 | 39 | 304 | 126 | 20 | 201 | 158 | 205 | 233 | 584 | 1333 | 1325 | 95412 |
| B1 | 25 | 25 | 50 | 225 | 250 | 75 | 200 | 200 | 250 | 125 | 500 | 2698 | 999 | 94379 |
| C1 | 10 | 105 | 33 | 119 | 95 | 43 | 282 | 234 | 115 | 363 | 516 | 1289 | 1289 | 95507 |
| D1 | 47 | 102 | 37 | 168 | 140 | 84 | 279 | 112 | 205 | 307 | 428 | 2523 | 1378 | 94192 |
| E1 | 28 | 42 | 83 | 249 | 194 | 69 | 180 | 125 | 277 | 138 | 512 | 2920 | 955 | 94228 |
| F1 | 5 | 118 | 15 | 159 | 67 | 62 | 257 | 293 | 129 | 221 | 560 | 1568 | 1203 | 95341 |
| G1 | 39 | 165 | 24 | 194 | 204 | 53 | 179 | 58 | 189 | 247 | 529 | 2454 | 1174 | 94490 |
| H1 | 0 | 6 | 0 | 296 | 661 | 3 | 340 | 29 | 7 | 498 | 17 | 606 | 890 | 96647 |
| I1 | 85 | 23 | 85 | 128 | 124 | 43 | 113 | 346 | 283 | 274 | 345 | 2323 | 1444 | 94382 |
| J1 | 46 | 63 | 11 | 193 | 151 | 7 | 319 | 260 | 200 | 119 | 474 | 2483 | 1105 | 94570 |
| K1 | 5 | 49 | 16 | 65 | 139 | 84 | 272 | 383 | 92 | 117 | 614 | 1283 | 1188 | 95691 |
| L1 | 43 | 36 | 25 | 140 | 36 | 36 | 428 | 122 | 263 | 324 | 378 | 1120 | 1688 | 95360 |
| M1 | 16 | 54 | 13 | 432 | 115 | 51 | 192 | 67 | 192 | 144 | 550 | 2513 | 1148 | 94514 |
| N1 | 86 | 33 | 7 | 178 | 119 | 26 | 198 | 224 | 250 | 79 | 619 | 1779 | 1252 | 95151 |
| O1 | 45 | 58 | 58 | 135 | 103 | 122 | 231 | 180 | 212 | 96 | 572 | 2410 | 1202 | 94575 |
| P1 | 7 | 31 | 50 | 474 | 67 | 22 | 87 | 50 | 87 | 132 | 800 | 1250 | 1382 | 95560 |
| Q1 | 54 | 34 | 41 | 183 | 332 | 27 | 176 | 122 | 264 | 109 | 454 | 2197 | 943 | 95063 |
| R1 | 9 | 48 | 18 | 254 | 169 | 16 | 229 | 338 | 64 | 105 | 537 | 1301 | 1249 | 95662 |
| S1 | 127 | 80 | 27 | 107 | 67 | 160 | 74 | 134 | 201 | 154 | 602 | 2213 | 1237 | 94817 |
| T1 | 6 | 18 | 54 | 223 | 235 | 127 | 97 | 157 | 133 | 121 | 549 | 2552 | 887 | 94841 |
| U1 | 7 | 22 | 31 | 109 | 64 | 22 | 107 | 97 | 191 | 164 | 800 | 814 | 1113 | 96459 |
| V1 | 75 | 0 | 19 | 115 | 71 | 14 | 105 | 104 | 441 | 113 | 551 | 1398 | 909 | 96084 |
| W1 | 15 | 29 | 18 | 147 | 50 | 24 | 144 | 162 | 197 | 303 | 509 | 2028 | 1266 | 95108 |
| X1 | 5 | 25 | 5 | 231 | 85 | 125 | 231 | 181 | 201 | 85 | 406 | 2112 | 828 | 95481 |
| Y1 | 41 | 36 | 51 | 87 | 31 | 5 | 273 | 264 | 351 | 62 | 364 | 1716 | 869 | 95851 |
| Z1 | 4 | 31 | 13 | 297 | 66 | 17 | 301 | 201 | 231 | 109 | 288 | 2328 | 926 | 95187 |
| A2 | 20 | 55 | 25 | 325 | 115 | 90 | 170 | 195 | 115 | 200 | 245 | 2050 | 915 | 95481 |
| B2 | 6 | 9 | 209 | 91 | 49 | 4 | 82 | 147 | 220 | 483 | 242 | 743 | 911 | 96802 |
| C2 | 63 | 33 | 37 | 187 | 53 | 57 | 240 | 224 | 180 | 83 | 327 | 1383 | 698 | 96433 |
| D2 | 24 | 77 | 19 | 89 | 228 | 56 | 100 | 81 | 212 | 121 | 453 | 1664 | 873 | 96002 |
| E2 | 5 | 103 | 18 | 166 | 27 | 10 | 161 | 203 | 206 | 127 | 388 | 910 | 1153 | 96524 |
| F2 | 15 | 15 | 10 | 175 | 113 | 87 | 277 | 252 | 67 | 62 | 308 | 1951 | 703 | 95964 |
| G2 | 1 | 56 | 33 | 151 | 82 | 68 | 173 | 79 | 70 | 91 | 268 | 935 | 653 | 97341 |
| H2 | 11 | 21 | 35 | 80 | 17 | 2 | 106 | 125 | 95 | 37 | 186 | 544 | 403 | 98338 |
| I2 | 1 | 23 | 4 | 14 | 9 | 4 | 32 | 38 | 15 | 38 | 387 | 291 | 674 | 98470 |
| J2 | 20 | 18 | 14 | 28 | 18 | 1 | 25 | 72 | 59 | 57 | 191 | 374 | 606 | 98518 |

**Figure 7.** Shows the number of times a rule was used during the evolution of different large combinators. Rules are column labels and combinators are row labels. They are ordered by how many times total a rule (besides the S and K rules) matched, from most to least. S and K rules are third and fourth from the right, and the rightmost column shows the number of subpatterns of combinators that did not match any combinator rule.

These are the rounded number of times a rule was used on average out of 100 000.

{a → 34, b → 65, c → 60, d → 215, e → 146, f → 58, g → 215,
h → 178, i → 216, j → 181, k → 467, l → 1918, m → 1083, n → 95 163}

## 7. Conclusion

Out of the numerous possible rules up to size five that could be made to skip combinator evolution, only 11 were not redundant in some way. This made it easier to look through different possible ways to use them for speeding up combinator evolution. In the end, adding too many extra rules also generally slows down evolution, as each rule needs to be compared with all parts and subparts of a combinator expression.

Optimization rules are generally most helpful for combinators that terminate. As for non-terminating combinators, we would most likely want to look at the evolution step by step, and optimization rules would skip steps in a generally hard-to-predict fashion. The majority of large combinators, say of size 100, appear to not terminate. Most begin to grow exponentially rather quickly, and some start exhibiting repetitive behavior. A couple of interesting cases were found where there was no exponential growth, but there also seemed to be less repetitive behavior, and it is unclear whether these will in fact terminate at some point.

## Acknowledgments

The author would like to thank Todd Rowland for his help and advice in collecting the data presented in this paper.

## References

[1] A. Church, *The Calculi of Lambda Conversion,* Princeton, NJ: Princeton University Press, 1985.

[2] S. Wolfram, *A New Kind of Science*, Champaign, IL: Wolfram Media, Inc., 2002 pp. 1122–1123.

[3] F. Obermeyer, "Automated Equational Reasoning in Nondeterministic λ-Calcluli Modulo Theories H*," thesis, Department of Mathematics, Carnegie Mellon University, Pittsburgh, PA, 2009. (Apr 4, 2017) fritzo.org/thesis.pdf.