FAQ : SSS Enumeration

In[23]:=
```
SetDirectory[ParentDirectory[NotebookDirectory[]]];
Needs["SSSiCv100`"];
```

Original Sessie Enumeration

The first attempt to enumerate sessies was published here This has now been replaced by enumerations allowing better predictions based on simpler mathematical relationships.

Updated Sessie Enumerations

Generalized Sessie Enumeration

Since then we have simplified and generalized the enumeration algorithm (the subject of a paper by Caviness, Case, Morrow & Kratzke)(2020). A new fully general algorithm based on 1 base-2 digit (starting with an empty string or non empty string) and further base-5 operation codes (which progressively modify the rules) were used in the code everywhere for several years.

```
1 : 0₂₅ :     {"" -> "A"}
2 : 1₂₅ :     {"A" -> ""}
3 : 0₂0₅ :    {"" -> "A", "" -> "", "A" -> ""}
4 : 0₂1₅ :    {"" -> "A", "" -> "A"}
5 : 0₂2₅ :    {"" -> "A", "A" -> ""}
6 : 0₂3₅ :    {"" -> "AA"}
7 : 0₂4₅ :    {"" -> "B"}
8 : 1₂0₅ :    {"A" -> "", "" -> "A"}
9 : 1₂1₅ :    {"A" -> "", "A" -> ""}
10 : 1₂2₅ :   {"A" -> "A"}
11 : 1₂3₅ :   {"AA" -> ""}
12 : 1₂4₅ :   {"B" -> ""}
13 : 0₂00₅ :  {"" -> "A", "" -> "", "A" -> "", "" -> "A"}
14 : 0₂01₅ :  {"" -> "A", "" -> "", "A" -> "", "A" -> ""}
15 : 0₂02₅ :  {"" -> "A", "" -> "", "A" -> "A"}
16 : 0₂03₅ :  {"" -> "A", "" -> "", "AA" -> ""}
17 : 0₂04₅ :  {"" -> "A", "" -> "", "B" -> ""}
18 : 0₂10₅ :  {"" -> "A", "" -> "A", "" -> "", "A" -> ""}
19 : 0₂11₅ :  {"" -> "A", "" -> "A", "" -> "A"}
20 : 0₂12₅ :  {"" -> "A", "" -> "A", "A" -> ""}
21 : 0₂13₅ :  {"" -> "A", "" -> "AA"}
22 : 0₂14₅ :  {"" -> "A", "" -> "B"}
23 : 0₂20₅ :  {"" -> "A", "A" -> "", "" -> "A"}
24 : 0₂21₅ :  {"" -> "A", "A" -> "", "A" -> ""}
25 : 0₂22₅ :  {"" -> "A", "A" -> "A"}
26 : 0₂23₅ :  {"" -> "A", "AA" -> ""}
27 : 0₂24₅ :  {"" -> "A", "B" -> ""}
28 : 0₂30₅ :  {"" -> "AA", "" -> "", "A" -> ""}
29 : 0₂31₅ :  {"" -> "AA", "" -> "A"}
```

```
30 : 0232₅ :  {"" -> "AA", "A" -> ""}
31 : 0233₅ :  {"" -> "AAA"}
32 : 0234₅ :  {"" -> "AB"}
33 : 0240₅ :  {"" -> "B", "" -> "", "A" -> ""}
34 : 0241₅ :  {"" -> "B", "" -> "A"}
35 : 0242₅ :  {"" -> "B", "A" -> ""}
36 : 0243₅ :  {"" -> "BA"}
37 : 0244₅ :  {"" -> "C"}
38 : 1200₅ :  {"A" -> "", "" -> "A", "" -> "", "A" -> ""}
39 : 1201₅ :  {"A" -> "", "" -> "A", "" -> "A"}
40 : 1202₅ :  {"A" -> "", "" -> "A", "A" -> ""}
41 : 1203₅ :  {"A" -> "", "" -> "AA"}
42 : 1204₅ :  {"A" -> "", "" -> "B"}
43 : 1210₅ :  {"A" -> "", "A" -> "", "" -> "A"}
44 : 1211₅ :  {"A" -> "", "A" -> "", "A" -> ""}
45 : 1212₅ :  {"A" -> "", "A" -> "A"}
46 : 1213₅ :  {"A" -> "", "AA" -> ""}
47 : 1214₅ :  {"A" -> "", "B" -> ""}
48 : 1220₅ :  {"A" -> "A", "" -> "", "A" -> ""}
49 : 1221₅ :  {"A" -> "A", "" -> "A"}
50 : 1222₅ :  {"A" -> "A", "A" -> ""}
51 : 1223₅ :  {"A" -> "AA"}
52 : 1224₅ :  {"A" -> "B"}
53 : 1230₅ :  {"AA" -> "", "" -> "A"}
54 : 1231₅ :  {"AA" -> "", "A" -> ""}
55 : 1232₅ :  {"AA" -> "A"}
56 : 1233₅ :  {"AAA" -> ""}
57 : 1234₅ :  {"AB" -> ""}
58 : 1240₅ :  {"B" -> "", "" -> "A"}
59 : 1241₅ :  {"B" -> "", "A" -> ""}
60 : 1242₅ :  {"B" -> "A"}
61 : 1243₅ :  {"BA" -> ""}
62 : 1244₅ :  {"C" -> ""}
```

Reduced Sessie Enumeration

Reduced Sessie Enumeration uses only base-5 codes, meaning that you never start with an empty string in the first rule. *fromReducedRank[n]* gives the n-th ruleset in the reduced quinary (base-5) enumeration. The first 62 rulesets are listed below. If we define the weight of a ruleset as the sum of the individual letters, with A=1, B=2, C=3, …, it is clear that in this list, weight 1 rulesets (containing only 1 letter, an A) all appear before weight 2 rulesets (containing either 2 As or 1 B), and weight 2 rulesets before those of weight 3, etc.

```
1 : ₅    : {"A" → ""}
2 : 0₅   : {"A" → "", "" → "A"}
3 : 1₅   : {"A" → "", "A" → ""}
4 : 2₅   : {"A" → "A"}
5 : 3₅   : {"AA" → ""}
6 : 4₅   : {"B" → ""}
7 : 00₅  : {"A" → "", "" → "A", "" → "", "A" → ""}
8 : 01₅  : {"A" → "", "" → "A", "" → "A"}
9 : 02₅  : {"A" → "", "" → "A", "A" → ""}
10: 03₅  : {"A" → "", "" → "AA"}
```

Out[•]=

```
11: 04₅ : {"A" → "", "" → "B"}
12: 10₅ : {"A" → "", "A" → "", "" → "A"}
13: 11₅ : {"A" → "", "A" → "", "A" → ""}
14: 12₅ : {"A" → "", "A" → "A"}
15: 13₅ : {"A" → "", "AA" → ""}
16: 14₅ : {"A" → "", "B" → ""}
17: 20₅ : {"A" → "A", "" → "", "A" → ""}
18: 21₅ : {"A" → "A", "" → "A"}
19: 22₅ : {"A" → "A", "A" → ""}
20: 23₅ : {"A" → "AA"}
21: 24₅ : {"A" → "B"}
22: 30₅ : {"AA" → "", "" → "A"}
23: 31₅ : {"AA" → "", "A" → ""}
24: 32₅ : {"AA" → "A"}
25: 33₅ : {"AAA" → ""}
26: 34₅ : {"AB" → ""}
27: 40₅ : {"B" → "", "" → "A"}
28: 41₅ : {"B" → "", "A" → ""}
29: 42₅ : {"B" → "A"}
30: 43₅ : {"BA" → ""}
31: 44₅ : {"C" → ""}
32: 000₅: {"A" → "", "" → "A", "" → "", "A" → "", "" → "A"}
33: 001₅: {"A" → "", "" → "A", "" → "", "A" → "", "A" → ""}
34: 002₅: {"A" → "", "" → "A", "" → "", "A" → "A"}
35: 003₅: {"A" → "", "" → "A", "" → "", "AA" → ""}
36: 004₅: {"A" → "", "" → "A", "" → "", "B" → ""}
37: 010₅: {"A" → "", "" → "A", "" → "A", "" → "", "A" → ""}
38: 011₅: {"A" → "", "" → "A", "" → "A", "" → "A"}
39: 012₅: {"A" → "", "" → "A", "" → "A", "A" → ""}
40: 013₅: {"A" → "", "" → "A", "" → "AA"}
41: 014₅: {"A" → "", "" → "A", "" → "B"}
42: 020₅: {"A" → "", "" → "A", "A" → "", "" → "A"}
43: 021₅: {"A" → "", "" → "A", "A" → "", "A" → ""}
44: 022₅: {"A" → "", "" → "A", "A" → "A"}
45: 023₅: {"A" → "", "" → "A", "AA" → ""}
46: 024₅: {"A" → "", "" → "A", "B" → ""}
47: 030₅: {"A" → "", "" → "AA", "" → "", "A" → ""}
48: 031₅: {"A" → "", "" → "AA", "" → "A"}
49: 032₅: {"A" → "", "" → "AA", "A" → ""}
50: 033₅: {"A" → "", "" → "AAA"}
51: 034₅: {"A" → "", "" → "AB"}
52: 040₅: {"A" → "", "" → "B", "" → "", "A" → ""}
53: 041₅: {"A" → "", "" → "B", "" → "A"}
54: 042₅: {"A" → "", "" → "B", "A" → ""}
55: 043₅: {"A" → "", "" → "BA"}
56: 044₅: {"A" → "", "" → "C"}
57: 100₅: {"A" → "", "A" → "", "" → "A", "" → "", "A" → ""}
58: 101₅: {"A" → "", "A" → "", "" → "A", "" → "A"}
59: 102₅: {"A" → "", "A" → "", "" → "A", "A" → ""}
60: 103₅: {"A" → "", "A" → "", "" → "AA"}
```

```
61: 104₅: {"A" → "", "A" → "", "" → "B"}
62: 110₅: {"A" → "", "A" → "", "A" → "", "" → "A"}
```

Within any given weight, advancing through the enumeration tends to compact the weight into fewer characters, moving it gradually towards the left. So the most spread out ruleset of weight 3, {“A”->””, “”->”A”,””->””,”A”->””} is the first of this weight in the list, {“A”->””,””->”B”} appears further down, and {“C”->””} is the last of this weight, with the entire weight compacted to the left.

Note: The “most spread out” rulesets have exactly 2 empty strings between single “A” strings, since we want to include the possibility of deletion rules (where the replacement string is “”) and insertion rules (where the search string is “”), and the possibility that an insertion might follow right after a deletion, thus putting 2 empty strings in sequence. E.g., {“A”->””,””->”B”}. But we will never need more than 2 consecutive empty strings, and this algorithm also produces “”->”” rules. More on that problem below.

The algorithm is guaranteed to produce all valid rulesets somewhere in the list, if one goes far enough. It can easily be seen that there is no limit on the number of rules in the ruleset, all cases eventually occurring, and all combinations of all letters will eventually appear everywhere in the ruleset.

As already mentioned, this does generate some unneeded cases, e.g., with empty strings at too many positions in the ruleset, such as “”->”” rules, or multiple “”->something rules -- only the first one could ever be used, or indeed any case where an insertion rule appears before the final position in the ruleset. But the most important thing is that all cases we want to study are included, and there is no repetition at all in the enumeration. It is also worth noting that this enumeration never produces invalid rulesets.

Explanation of Q-Codes

- 0: end this string, insert two empty strings and start a new string with an “A”
- 1: end this string, insert one empty string and start a new string with an "A"
- 2: end this string and start a new string with an "A"
- 3: end this character and start a new character (as an "A")
- 4: increment this character

*****Look at the RSSEnumerationIntro.pptx in Intro-Documents in 0intro to make animation******

Ruleset tests:

Any ruleset that contains a problem can be skipped without generating the sessie and its network, to save time. In addition, the pattern and order of the enumeration lets us skip over multiple cases at once. If such a “long-jump” is possible, the test returns the first ruleset in the enumeration that does not have that particular problem, the target of the long-jump. The main tests are:

Test that eliminate the ruleset, but don’t allow long-jumps:

Command Test Description
(* Two rules conflict if the second rule never has a chance to be used because the first rule preempts it. Ex/ {“C”->””, “A”->”B”, “AA”->”C”} : rule 3 conflicts with rule 2, so the ruleset is equivalent to {“C”->””, “A”->”B”}, omitting rule 3 *)
(* an identity rule is a rule in which a string is replaced by itself, no further rules will be used if the

identity rule ever matches *)
(or
) (* if solo identity rules are treated separately *)
(* Ex/ {“A”->”BAB”, …} If 1st rule matches, it will never fail again. *)
(* If the characters used in the strings of the ruleset are permuted or replaced by other characters, the SSS will look the same up to a permutation of colors, and the causal network will be identical. Ex/ {“BAB”→”ABA”,”A”→”B”,”B”→”AA”} is indistinguishable from {“ABA”→”BAB”,”B”→”A”,”A”→”BB”}
(* Still under construction: the idea is that if a rule is never used, it could be omitted without affecting the results. The difficulty is in being sure that the rule will never be used, rather than only rarely. *)
Tests that eliminate the ruleset, but don’t allow long-jumps