FAQ: “Sessie” Introduction

In[1]:=
```
SetDirectory[ParentDirectory[NotebookDirectory[]]];
Needs["SSSiCv100`"];
```

Sequential Substitution Systems

You can experiment with these functions. Download the “SSSiCv100" header file, containing the sessie code, save it locally on your computer or in your Wolfram Cloud, in your sessie research folder.

https://www.wolframcloud.com/obj/90ec721d-78e2-4836-83ce-7c844556144d

Download the "sandbox" file, save it rename it if you want to try something new and keep a record of it.

https://www.wolframcloud.com/obj/256e508b-d3b5-4d91-80bf-0f78a6ebe85d

The toy “universe” is a string of characters, whose initial state can have any length and can contain any characters. (There are two infinities here: the length of the state string and the size of the alphabet from which the characters are taken: both have no imposed bound.) The “laws of nature” for the SSS are codified by a ruleset: a set of replacement instructions, each giving a string to search for and the string to replace it with. Ex: {“AB”→“BA”}. (Now we have more infinities: there is no limit on the number of rules in the ruleset, nor on the size of any of the strings, nor on the characters in the strings.)

In a *sequential* substitution system, rules must be applied sequentially in order (use the first rule if possible, the second only if the first fails, etc.) from left-to-right: if the first usable rule can be used at multiple locations in the state string, we apply it in the first possible (left-most) position.

Each application of any rule is an event, and each event destroys and/or creates cells (or letters) in the state string. Two events are considered to be causally connected if a cell created by one is destroyed by the other.

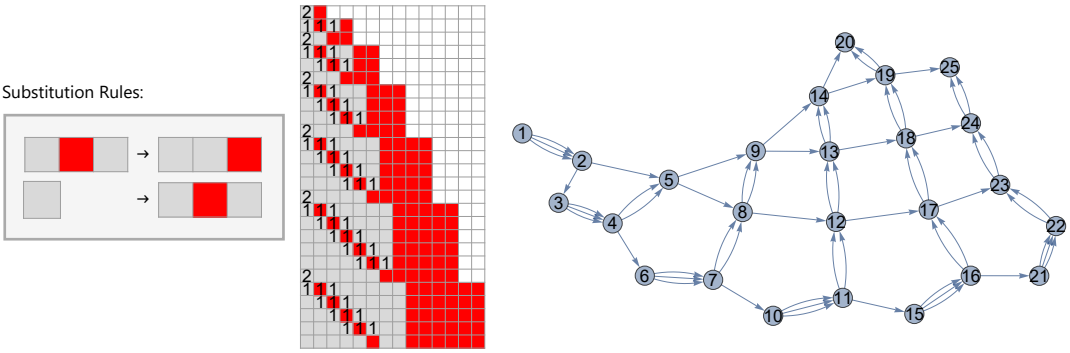
The causal network of an SSS can be visualized by treating all events as graph nodes and the causal connections between them as graph edges.

The basic command to display a Sessie is SSS. The template for usage is:

SSS[ruleset, initial state, number of steps, options]

Example:

In[•]:=
```
sss1 = SSS[{"ABA" → "AAB", "A" → "ABA"}, "AB",
    25, RulePlacement → Left, SSSSize → 100, NetSize → 300];
```



(1) By default, the display includes both the sessie (on the left) and its causal network (on the right). The colored cells in the sessie display represent the changing state string, one line per state, with (micro-)time advancing downward. (Here the gray cells are As and the red cells Bs). Events correspond to the change from one state to the next.

(2) We can dispense with the arrows indicating the direction of the causal connection in the network if the node numbers are given, since the arrow will always point from smaller to larger node number.

(3) Although the command above performed 25 steps and so the causal network has 25 events, the SSSMax option can limit the display of the actual sessie to whatever we want, showing only the beginning of the continuing pattern.

(a couple other common options: NetSize→200,HighlightMethod→Dot)

In[•]:=
```
? SSS
```

Out[•]=

Symbol

SSS[*ruleset*, *init*, *n*, *opts*] creates and displays a sequential substitution system (SSS) and its causal network, using *ruleset* (as a list of rules, or a RSS index), starting with the state *init* (using string notation), allowing the SSS to evolve for *n* steps. If the initial state string is omitted, SSSInitialState is called to provide a sufficiently complex string. Use the option EarlyReturn to give/deny permission to quit early if the SSS can be identified as dead or repeating.) Use option Mode → Silent to suppress display of the sessie. Any other options given are passed on to SSSDisplay.

(Returns a copy of the SSS that can then be displayed or manipulated without rebuilding, using SSSDisplay, SSSAnimate, or directly, looking at its keys, "Evolution" and "Net", etc.)

Once the sessie has been created you can display it (using SSSDisplay), change options, look at the network, etc., without reconstructing it.

In[3]:= **? SSSDisplay**

Out[3]=

Symbol

SSSDisplay[*sss*, *opts*] displays the sequential substitution system *sss*
and/or its causal network. Use SSS (or SSSInitialize and SSSEvolve) to construct it first.

Options:

Min → *n* cuts off the display before the first *n*
steps of the system. (Separate values can be specified for SSSMin and NetMin.)

Max → *n* cuts off the display after the first *n*
steps of the system. (Separate values can be specified for SSSMax and NetMax.)

VertexLabels → Automatic (or "Name") |
"VertexWeight" | … labels vertices by node number or distance from origin, etc.

HighlightMethod → Dot | Frame | Number (or
True) | None (or False) specifies how the matches in the SSS are highlighted.

RulePlacement → Bottom | Top | Left | Right | None (or False)
specifies where to place the rulelist icon relative to the SSS visual display (if shown).

Sizes of display components are specified by the options NetSize, SSSSize,
IconSize and ImageSize (which refers to the pane containing the SSS display and icon).

NetMethod → GraphPlot | LayeredGraphPlot | TreePlot | GraphPlot3D | All | NoSSS | list of methods,
where NoSSS generates no SSS display (causal
network only) and the other choices specify how the causal network is to be shown.

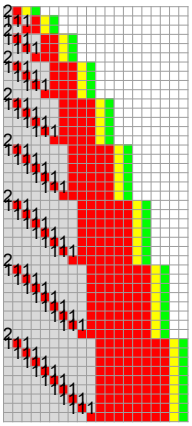
You may choose options interactively using SSSInteractiveDisplay, then click one of the “use” buttons for one time or default use.

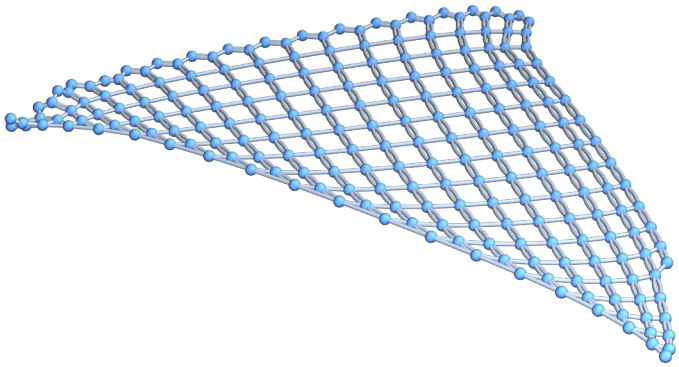
In[ ]:= **SSSInteractiveDisplay[sss2]**

We might chose the GraphPlot3D display for the network. (Clicking and dragging with the mouse shows different view points.)

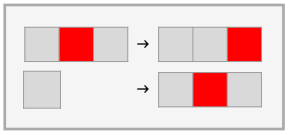
In[•]:=
```
SSSDisplay[sss2, Min → 1, Max → 500, SSSMin → Automatic, SSSMax → 45, NetMin → Automatic,
 NetMax → 224, HighlightMethod → Number, RulePlacement → Bottom, NetMethod → {GraphPlot3D},
 ImageSize → 170.`, NetSize → 350, SSSSize → {Automatic, 220}, IconSize → {Automatic, 20},
 VertexSize → Automatic, VertexLabels → Placed[Automatic, Tooltip], DirectedEdges → False]
```

Out[•]=





Substitution Rules:



SSSAnimate shows the order the network is created.

In[•]:=
```
SSSAnimate[sss1]
```

Other components of the Sessie object can also be studied individually.

In[•]:=
```
sss1[["RuleSet"]]
```

Out[•]=
```
{ABA → AAB, A → ABA}
```

In[•]:=
```
sss1[["Evolution"]]
```

Out[•]=
```
{AB, ABAB, AABB, ABAABB, AABABB, AAABBB, ABAAABBB, AABAABBB, AAABABBB, AAAABBBB,
 ABAAAABBBB, AABAAABBBB, AAABAABBBB, AAAABABBBB, AAAAABBBBB, ABAAAAABBBBB,
 AABAAAABBBBB, AAABAAABBBBB, AAAABAABBBBB, AAAAABABBBBB, AAAAAABBBBBB,
 ABAAAAAABBBBBB, AABAAAAABBBBBB, AAABAAAABBBBBB, AAAABAAABBBBBB, AAAAABAABBBBBB}
```

In[•]:=
```
sss1[["RulesUsed"]]
```

Out[•]=
```
{2, 1, 2, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1}
```

The RuleSet contains the “laws of nature” for this toy universe, rules which are always invoked sequentially: The first rule is always used before the second, if possible, and any rule is always used in the leftmost position possible in the state string. The Evolution gives the list of all state strings from the initial state on. For instance, sss1 only contains the first 25 time steps. The colored box picture is a visual

representation of these state strings, starting with the initial state string at the top, with time flowing downward in the graphic. The component RulesUsed gives the list of which rules were used at each time step.

Context: Sessies and Simple Programs:

Sequential Substitution Systems are treated in one chapter of Stephen Wolfram's book A New Kind of Science, and an initial (not directly extensible) treatment has been published in the journal Complex Systems:

Exploring the Space of Substitution Systems. http://www.complex-systems.com/pdf/22-1-1.pdf

SSSs ("Sessies") serve as one example of the larger class of systems that although defined by simple rules can result in complex behavior. Perhaps the simplest such system is the Elementary Cellular Automaton (see FAQ: ECA), a 1-d finite length binary string that is modified as specified to be a finite fixed length set of rules, depending on the bit under consideration and its neighbors. John Conway's famous "Game of Life" is a 2-d version of this.

Turing Machines add an internal state of the "head" to the bits recorded on the "tape", thus extending the concept of cellular automaton, which only consists of the "tape". (Possibly explain more on what you mean by "head" and "tape" in this context)

Substitution systems add the possibility of increasing or decreasing the length of string, by replacing a substring by another of different length. Our treatment extends this variation by allowing either target or replacement to be the empty string, forming insertion and deletion rules. Substitution systems also allow multiple rules and an unbounded set of characters making up the strings the rules refer to. In a sequential substitution system, application of these rules is attempted successively and sequentially, as has been described above. In a multi-way substitution system, all rules are applied at all positions simultaneously, possibly forming a large (but not infinite) number of outcomes. The Generalized Substitution System enumeration includes all multi-way SS rulesets, as well as all Sessie rulesets.

Added complications: Could the SSS 1-d state string be extended to 2-d or higher dimensions, given the possibility of inserting/deleting characters? Would the fabric of the state array (2-d analog of the state string) bunch up due to these changes? Could a n x m target block be searched for, and deleted, shrinking each of the n rows by m letters, and then a n' x m' block inserted at the same place, inserting m' characters into each of n' rows? Possible, but complicated!

One idea, proposed by Charles Sarr in a discussion on 2016.07.10, is that the state string be treated as wrapping around, much as the coils of a spring. Rules could then seek for 2-d patterns and insert 2-d patterns in the target locations. This would presumably not change the radius of curvature of the spring, but might result in shifting of the line-up of the coils, as can easily be visualized by considering how a physical spring would be affected by inserting or deleting segments of one or more coils: