Tool Set for Morphic Cellular Automata Systems

Dr. phil Rudolf Kaehr

ISSN 2041-4358
(work in progress, vs. 0.1, July 2014)

Functional and morphogrammatic definitions of ECA

Again, it easily happens to confuse the morpgram-based approach to elementary CAs with the original function-based approach as we know it. That happens naturally because of the coincidence of the objectified results.

What differs is *how* the results are achieved (produced) and not so much *what* is reached (produced).

Functional representations of the morphoCAs are in fact *simulations* in the realm of functions, and not morphic *realizations*.

Again, there is no 'natural' method to extend the classical ECA concept to a 'trans-classical' theory on the base of set-theoretical functions.

An extension of the function-based approach is easily achieved with an extension of the value-set from 2 elements to n>2. But such a concept of extension is abstract and there are no systematic criteria to chose the elements. The value-set might be arbitrarily extended to any size.

The analogous situation happened and still happens with the transition from 2-valued to multiple-valued logics.

Hence, the morphogrammatic approach to ECAs is an interpretation of the basic morphograms of morphogrammatics.

In the case of complexity/complication of 4, i.e. for $MG^{(4,4)}$, there are just 15 basic morphograms. To define the classical ECAs, not more than 8 morphograms are necessary as a base for interpretation.

This way to interpret morphograms by values and relabeling is not yet taking the genuine morphogrammatic level into account. Morphograms are introduced by *differences* and not by *values* of a function. The difference-oriented approach to morphogrammatic CAs is ruled by the ϵ/ν-structuration of the domain of computation.

The first presentation of morphogrammatic based CAs had been restricted on a combination of just 4 to 5 morphograms per automaton.

The morphogrammatic approach enables an easy method of combining basic cellular automata to compound structures of well defined complex morphic automata.

The tool set contains the 15 basic morphograms and the two rules of composition: *ruleCl* for 'classical' and *ruleM* for 'trans-classical structurations.

The tool set allows to define morphogrammatic compounds of basic morphograms for more complex CAs.

In this sense, a morphogram $MG^{(9,3)}$ is defined as a *mediation* of 3 basic morphograms: e.g.

$MG^{(9,3)} = MG^{(4,3)}$ ⊔ $MG^{(4,3)}$ ⊔ $MG^{(4,3)}$.

Morphic CAs are therefore defined as interpretations of *additive* and *mediative* compositions of morphogrms.

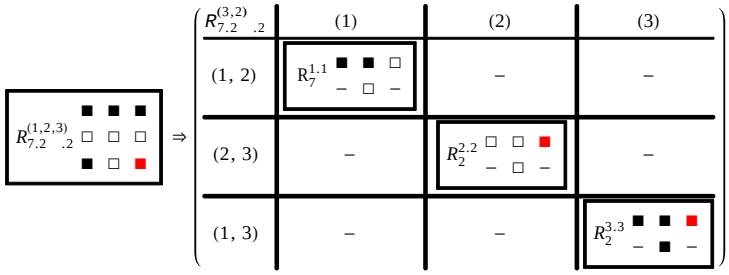
Additive compositions have the form:

$$\left[MG_1^{(4,n)}, MG_2^{(4,n)}, \ldots, MG_m^{(4,n)}\right], \text{ with } \begin{pmatrix} m = 4 : \text{classical} \\ m = 5 : \text{transclassical} \end{pmatrix} \text{ for CAs.}$$

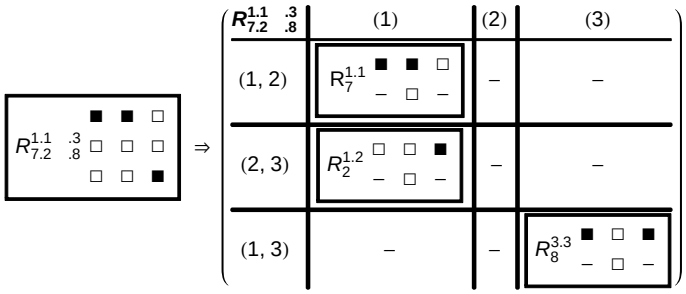
While *mediative* compositions have the reflectional and interactional distribution form:
$\left[MG_1^{(4,n)} \sqcup MG_2^{(4,n)} \sqcup, \ldots, \sqcup MG_m^{(4,n)}\right]$.

Mediative composition examples for m=3:

$\mathbf{CA^{(9,3)}}$: $R_{7.2}^{1.1}{}_{.2}^{.3} = \left(R_7^1 \sqcup R_2^2\right) \sqcup R_2^3$, "⊔" : mediation



CA (9,2): $R_{7.2}^{1.1\ .3}_{\ .8} = \left(R_7^1 \square R_2^1\right) \amalg R_8^3$, "□": replication, "Ш" : mediation



http : // memristors.memristics.com/Notes on Polycontextural Logics/Notes on Polycontextural Logics.html

Other definitions and derivations from the morphogrammatic approach are naturally possible as different and non-standard interpretations of morphograms.

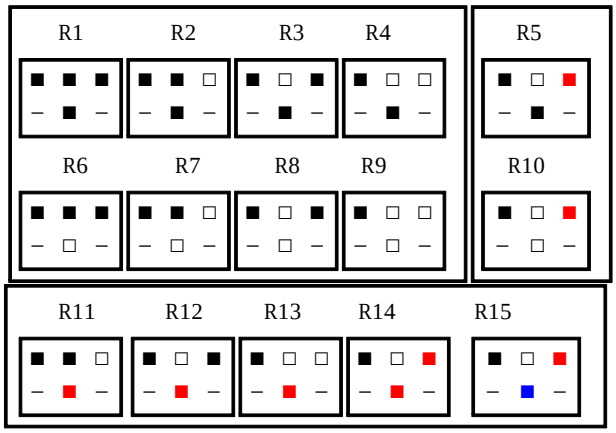
It is obviously the advantage of the morphogrammatic approach to enable such systematic automata-theoretical constructions and structurations.

"It might be thought that CA with greater values of κ have also greater computational power, however this is not true.It is true that rules with κ = 1 can be easily characterized because they describe linearly separable CA, but even rules with κ = 2 can have extremely complex behaviors as in the case of rule 110, which is known to be equivalent to a Universal Turing Machine.
This phenomenon, already hypothesized by Wolfram, is called threshold of complexity. It is noteworthy to mention that not all rules with high complexity index have complex behaviors, [...]."
Giovanni E. Pazienza, Aspects of algorithms and dynamics of cellular paradigms

http://www.tdx.cat/bitstream/handle/10803/9151/gpazienza_thesis.pdf

System of elementary morphic cellular automata rules

$$\text{rules}-\text{CA}^{(4,4)} = \sum_{k=1}^{4} \text{Sn2}(4, k) = 1+6+7+1 = 15$$



There might be a difference between the *pheno* - and the *geno* - type of CAs. In other words, the surface - structure as an interpretation of morphograms, i.e. as a function- and set - oriented approach has to be distin-

guished from a morphogrammatic understanding of CAs that is pre - ordered as a pattern-oriented deep-structure of mappings and sets.

Structurally, the system of $CA^{(2,2)}$ with its 256 elements or mappings is inherently symmetric. That is not excluding that the behavior of some functions are asymmetric.

It turns out that not even dually defined CAs are behaving dually. That is, e.g, that the dual of rule 30 has not a dual representation of the rule 30.

In sharp contrast, the morphogrammatic system is with its 15 basic morphograms for $CA^{(4,4)}$ is inherently asymmetric. That obviously includes the symmetric subsystem of $CA^{(2,2)}$ too.

The morphogrammatic approach enables a simple *modular* construction of the automata.

Rule space for $CA^{(m,n)}$

$CA^{(4,2)}$: 16

```
rule1.2  .3  .4
rule1.2  .3  .9
rule1.2  .8  .4
rule1.2  .8  .9

rule1.7  .3  .4
rule1.7  .3  .9
rule1.7  .8  .4
rule1.7  .8  .9
```

```
rule6.2  .3  .4
rule6.2  .3  .9
rule6.2  .8  .4
rule6.2  .8  .9

rule6.7  .3  .4
rule6.7  .3  .9
rule6.7  .8  .4
rule6.7  .8  .9
```

$CA^{(4,3)}$: 32

```
appendix =
  .5; 10; 14

rule1.2  .3  .4
rule1.2  .3  .9
rule1.2  .8  .4
rule1.2  .8  .9

rule1.7  .3  .4
rule1.7  .3  .9
rule1.7  .8  .4
rule1.7  .8  .9
```

```
appendix =
  .5; 10; 14

rule6.2  .3  .4
rule6.2  .3  .9
rule6.2  .8  .4
rule6.2  .8  .9

rule6.7  .3  .4
rule6.7  .3  .9
rule6.7  .8  .4
rule6.7  .8  .9
```

$CA^{(4,4)}$: (not complete)

```
rule1.2  .3  .4  .5.–10–14–15
rule1.2  .8  .4  .5.–10–14–15
rule1.2  .12  .4  .5.–10–14–15

rule1.2  .3  .9  .5.–10–14–15
rule1.2  .8  .9  .5.–10–14–15
rule1.2  .12  .9  .5.–10–14–15

rule1.2  .3  .13  .5.–10–14–15
rule1.2  .8  .13  .5.–10–14–15
rule1.2  .12  .13  .5.–10–14–15
```

```
rule1.7  .3  .4  .5 –10–14–15
rule1.7  .3  .9  .5 –10–14–15
rule1.7  .3  .13  .5 –10–14–15

rule1.7  .8  .4  .5.–10–14–15
rule1.7  .8  .9  .5.–10–14–15
rule1.7  .8  .13  .5.–10–14–15

rule1.7  .12  .4  .5.–10–14–15
rule1.7  .12  .9  .5.–10–14–15
rule1.7  .12  .13  .5.–10–14–15
```

```
rule1.11  .3  .4  .5.–10–14–15
rule1.11  .3  .9  .5.–10–14–15
rule1.11  .3  .13  .5.–10–14–15

rule1.11  .8  .4  .5.–10–14–15
rule1.11  .8  .9  .5.–10–14–15
rule1.11  .8  .13  .5.–10–14–15

rule1.11  .12  .4  .5.–10–14–15
rule1.11  .12  .9  .5.–10–14–15
rule1.11  .12  .13  .5.–10–14–15
```

```
rule6.2  .3  .4  .5.–10–14–15
rule6.2  .8  .4  .5.–10–14–15
rule6.2  .12  .4  .5.–10–14–15

rule6.2  .3  .9  .5.–10–14–15
rule6.2  .8  .9  .5.–10–14–15
rule6.2  .12  .9  .5.–10–14–15

rule6.2  .3  .13  .5.–10–14–15
rule6.2  .8  .13  .5.–10–14–15
rule6.2  .12  .13  .5.–10–14–15
```

```
rule6.7   .3   .4   .5  −10−14−15
rule6.7   .3   .9   .5  −10−14−15
rule6.7   .3   .13   .5  −10−14−15

rule6.7   .8   .4   .5.−10−14−15
rule6.7   .8   .9   .5.−10−14−15
rule6.7   .8   .13   .5.−10−14−15

rule6.7   .12   .4   .5.−10−14−15
rule6.7   .12   .9   .5.−10−14−15
rule6.7   .12   .13   .5.−10−14−15
```

```
rule6.11   .3   .4   .5.−10−14−15
rule6.11   .3   .9   .5.−10−14−15
rule6.11   .3   .13   .5.−10−14−15

rule6.11   .8   .4   .5.−10−14−15
rule6.11   .8   .9   .5.−10−14−15
rule6.11   .8   .13   .5.−10−14−15

rule6.11   .12   .4   .5.−10−14−15
rule6.11   .12   .9   .5.−10−14−15
rule6.11   .12   .13   .5.−10−14−15
```

```
rule1.3   .11   .13   .5−10−14−15
rule1.3   .7   .13   .5−10−14−15

rule1.8   .11   .13   .5−10−14−15
```

```
rule6.3   .11   .13   .5−10−14−15

rule6.8   .11   .13   .5−10−14−15
```

Program schemes for morphic CAs

Rule function scheme

$$\text{rules}_i = \text{Mod}[\text{ReLabel}_i[\{a, b, c\}], n] \to d,\quad a, b, c, d \in \{0, 1, 2, 3\},\ 1 \le i \ge 5,\ 2 \le n \ge 4$$

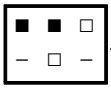
Relabeling

(Debug) In[5]:=

```
ReLabel[L_List] := L /. Map[#[[1]] → #[[2]] &,
    Transpose[{DeleteDuplicates[L], Range[Length[Union[L]]]}]]
```

Two functions are morphogrammatically equivalent if they are equal under relabeling.

Hence, the listed functions below are all representing in the context of $CA^{(3,2)}$

the same morphogram R7, i.e.  .

List of functions

```
{0, 0, 1} → 0, {0, 0, 2} → 0, {0, 0, 3} → 0, {1, 1, 0} → 0, {1, 1, 2} → 0, {1, 1, 3} → 0,
{2, 2, 0} → 0, {2, 2, 1} → 0, {2, 2, 3} → 0, {3, 3, 0} → 0, {3, 3, 1} → 0, {3, 3, 2} → 0
```

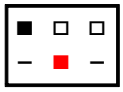
Morphogrammatic equivalence

(Debug) In[6]:=

```
Mod[ReLabel[{0,0,1}],2]==
Mod[ReLabel[{0,0,2}],2]==
Mod[ReLabel[{0,0,3}],2]==
Mod[ReLabel[{1,1,0}],2]==
Mod[ReLabel[{2,2,0}],2]==
Mod[ReLabel[{2,2,1}],2]==
Mod[ReLabel[{2,2,3}],2]==
Mod[ReLabel[{3,3,0}],2]==
Mod[ReLabel[{3,3,1}],2]==
Mod[ReLabel[{3,3,2}],2]
```

(Debug) Out[6]= True

Permutations

Morphogram R13 :  **, is represented in $CA^{(4,3)}$ by the following functions.**

List of functions

(Debug) In[7]:=
```
rca[{13}] :=
 {
  {0, 1, 1} -> 2,
 {0, 2, 2} -> 1,
 {0, 3, 3} -> 2,
 {1, 0, 0} -> 2,
 {1, 2, 2} -> 0,
 {1, 3, 3} -> 2,
 {2, 3, 3} -> 1,
 {2, 0, 0} -> 1,
 {2, 1, 1} -> 0,
 {3, 1, 1} -> 2,
 {3, 0, 0} -> 1,
 {3, 2, 2} -> 0
 }
```

Morphogrammatic equivalence

(Debug) In[8]:=
```
Mod[ReLabel[{0,1,1,2}],3] ==
 Mod[ReLabel[{0,2,2,1}],3] ==
 Mod[ReLabel[{0,3,3,2}],3] ==
 Mod[ReLabel[{1,0,0,2}],3] ==
 Mod[ReLabel[{1,2,2,0}],3] ==
 Mod[ReLabel[{1,3,3,2}],3] ==
 Mod[ReLabel[{2,3,3,1}],3] ==
Mod[ReLabel[{2,1,1,0}],3] ==
Mod[ReLabel[{2,0,0,1}],3] ==
Mod[ReLabel[{3,0,0,1}],3] ==
Mod[ReLabel[{3,1,1,2}],3] ==
Mod[ReLabel[{3,2,2,0}],3]
```

(Debug) Out[8]= `True`

Basic composition scheme for morphoRules

The function *RCA* denotes the composed rules based on the *rule space* defined by the combination of the sub-rules *rca*.

This approach is nicely modular and allows easily to compose morphic elementary cellular automata.
The rule space is defined by the composition of the 15 basic morphograms for $CA^{(4,4)}$.
The functional definition of the morphograms might differ depending on the understanding of its mechanism.

(Debug) In[91]:=
```
ruleMN[{a_, b_, c_, d_, e_, f_}] :=
    Flatten[{rca[{a}], rca[{b}], rca[{c}], rca[{d}], rca[{e}], rca[{f}]}]
```

(Debug) In[92]:=
```
ruleM[{a_, b_, c_, d_, e_}] :=
    Flatten[{rca[{a}], rca[{b}], rca[{c}], rca[{d}], rca[{e}]}]
```

(Debug) In[93]:=
```
ruleCl[{a_, b_, c_, d_}] :=
    Flatten[{rca[{a}], rca[{b}] , rca[{c}], rca[{d}]}]
```

(Debug) In[12]:=
```
morphoRules := { ruleCl, ruleM, ruleMN}
```

General CA scheme

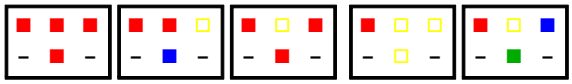
```
Presentation[
 ArrayPlot[CellularAutomaton[
  ruleDef (MG^(m,n)), morphoRules, ruleSpace
  {init}, steps],
 {Graphics}]]

presentation =
 {TabView, SlideViev, MenuView, etc}
```

Introduction

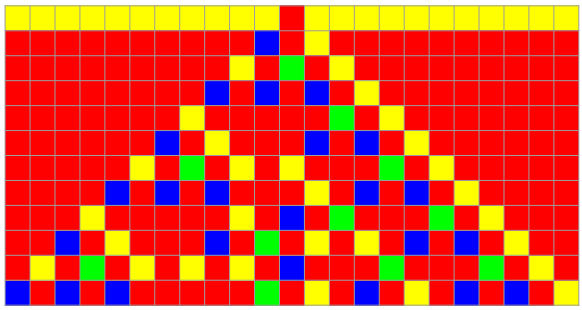
How does is work? Relabeling (i.e., trito-normal form, tnf) and separation.

Example

```
ruleM[{1, 11, 3, 9, 15}]
```



```
ColorRules -> {1 -> Red, 0 -> Yellow, 2 → Blue, 3 → Green}
```



Step 1

```
ArrayPlot[CellularAutomaton[
  ruleM[{1, 11, 3, 9, 15}],
  {{1}, 0}, 1],
 ColorRules -> {1 -> Red, 0 -> Yellow, 2 → Blue, 3 → Green}, Mesh → True]
```

=

0	1	0
2	1	0

[0, 1, 0] corresponds : [1, 0, 1], hence the next step is [1, 0, 1] -> 0 or [1, 0, 1] -> 1.
Because rule 3 is involved and not rule 8, the next step is 1 and not 0.

```
Mod[ReLabel[{0, 1, 0}], 2]
```

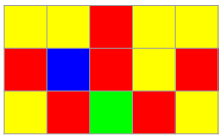
{1, 0, 1}

Hence,

0	1	0
–	1	–

Step 2

(Debug) Out[83]=



0	0	1	0	0
□	2	1	0	□
□	□	3	□	□

[2,1,0] corresponds by relabeling to [1,2,0], hence the next step is: [1,2,0] -> 0 or [1,2,0] -> 1 or [1,2,0] -> 2 or [1,2,0] -> 3.
In contrast to a set-based approach, there are no other possibilities involved on the level of morphogrammatics. Because rule 15 is involved and not rule 5 or rule 10 or rule 14, the next step is 3 and not 0, 1 or 2.

The head of the morphogram

(Debug) In[84]:=
```
Mod[ReLabel[{2, 1, 0}], 3]
```

```
{1, 2, 0}
```

The morphogram 15

(Debug) In[88]:=
```
Mod[ReLabel[{2, 1, 0, 3}], 4]
```

```
{1, 2, 3, 0}
```

1	2	3
–	0	–

$=_{MG}$

1	0	2
–	3	–

=

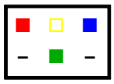
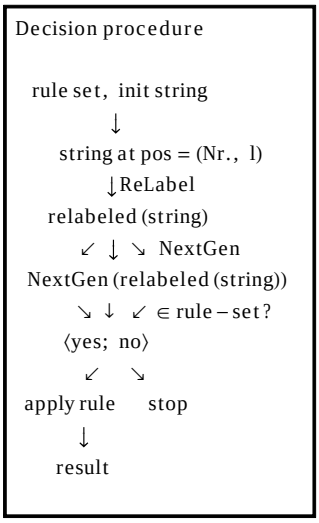
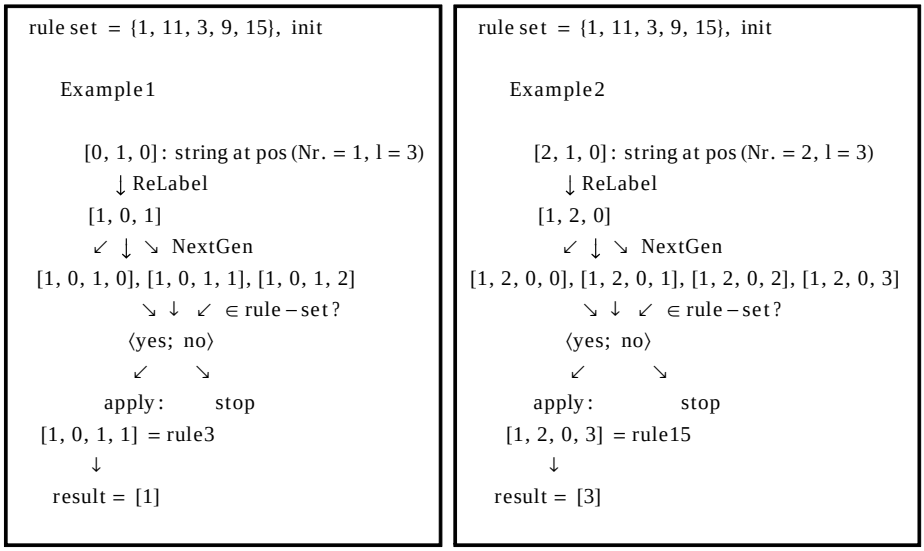


Diagram of the separation procedure





http : // memristors.memristics.com/CA - Compositions/Memristive %20 Cellular %20 Automata %20 Compositions.pdf

On the level a functional implementation of the morphic cellular automata, like in this paper, the genuine concept of morphic interaction is replaced by a corresponding set of ' pre - given' functions.

TabView for $CA^{(4,4)}$ rules

Table of explicit rules for $CA^{(4,4)}$

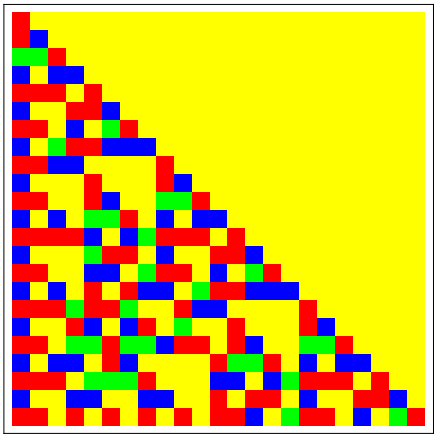
Applications

(Debug) In[15]:=
```
ruleM[{a_, b_, c_, d_, e_}] :=
    Flatten[{rca[{a}], rca[{b}], rca[{c}], rca[{d}], rca[{e}]}]
```

rule = 6.7.3.13.15

(Debug) In[89]:=
```
ArrayPlot[CellularAutomaton[
  ruleM[{6, 7, 3, 13, 15}],
  {{1}, 0}, 22],
 ColorRules -> {1 -> Red, 0 -> Yellow, 2 → Blue, 3 → Green}]
```

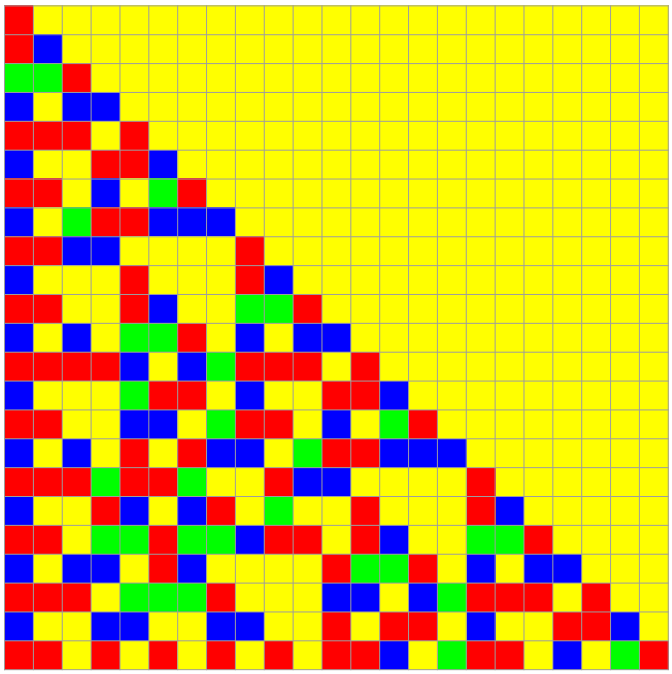
(Debug) Out[89]=



Mesh view

(Debug) In[17]:=
```
ArrayPlot[CellularAutomaton[
  ruleM[{6, 7, 3, 13, 15}],
  {{1}, 0}, 22],
 ColorRules -> {1 -> Red, 0 -> Yellow, 2 → Blue, 3 → Green}, Mesh → True]
```

(Debug) Out[17]=



Rule verification

(Debug) In[20]:=
```
ruleM[{6, 7, 3, 13, 15}]
```

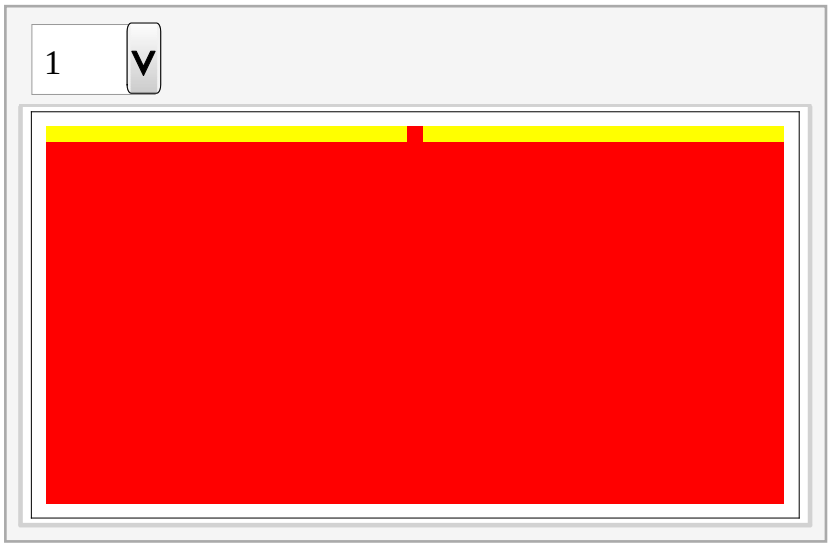
Dynamic representation of classic morphoCAs

Classic rules

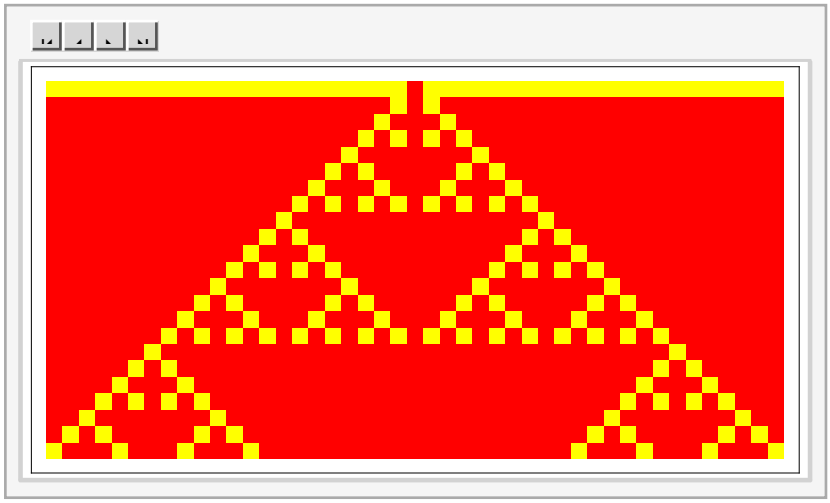
Classical examples : ruleCL, 16

(Debug) In[21]:=
```
ruleCl[{a_, b_, c_, d_}] :=
 Flatten[{rca[{a}], rca[{b}] , rca[{c}], rca[{d}]}]
```

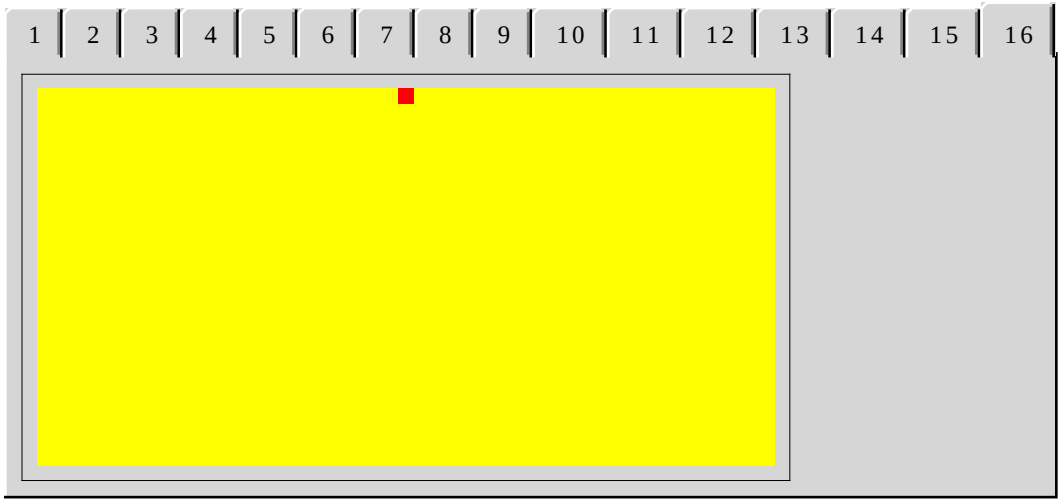
(Debug) Out[22]=



(Debug) Out[23]=



(Debug) Out[67]=

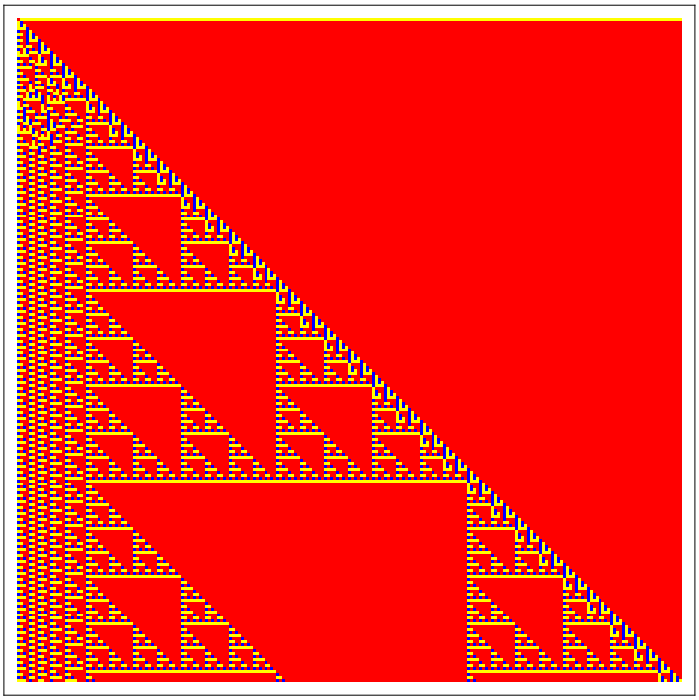


Dynamic representation of transclassic morphoCAs

rule = 1.2.8.13.10

(Debug) In[26]:=
```
ArrayPlot[CellularAutomaton[
  ruleM[{1, 2, 8, 13, 10}],
  {{1}, 0}, 222],
 ColorRules -> {1 -> Red, 0 -> Yellow, 2 → Blue, 3 → Green}]
```

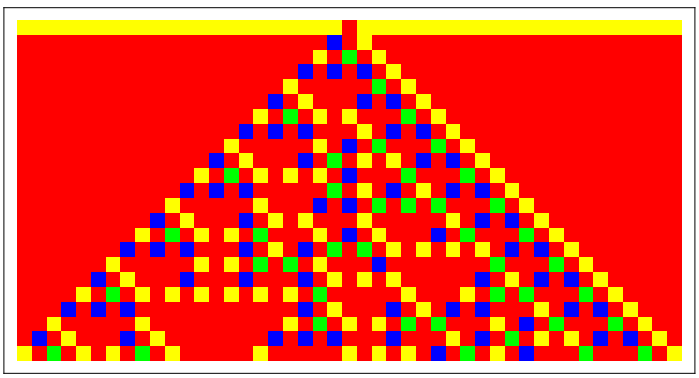
(Debug) Out[26]=



rule = 1.11.3.9.15

(Debug) In[27]:=
```
ArrayPlot[CellularAutomaton[
  ruleM[{1, 11, 3, 9, 15}],
  {{1}, 0}, 22],
 ColorRules -> {1 -> Red, 0 -> Yellow, 2 → Blue, 3 → Green}]
```

(Debug) Out[27]=

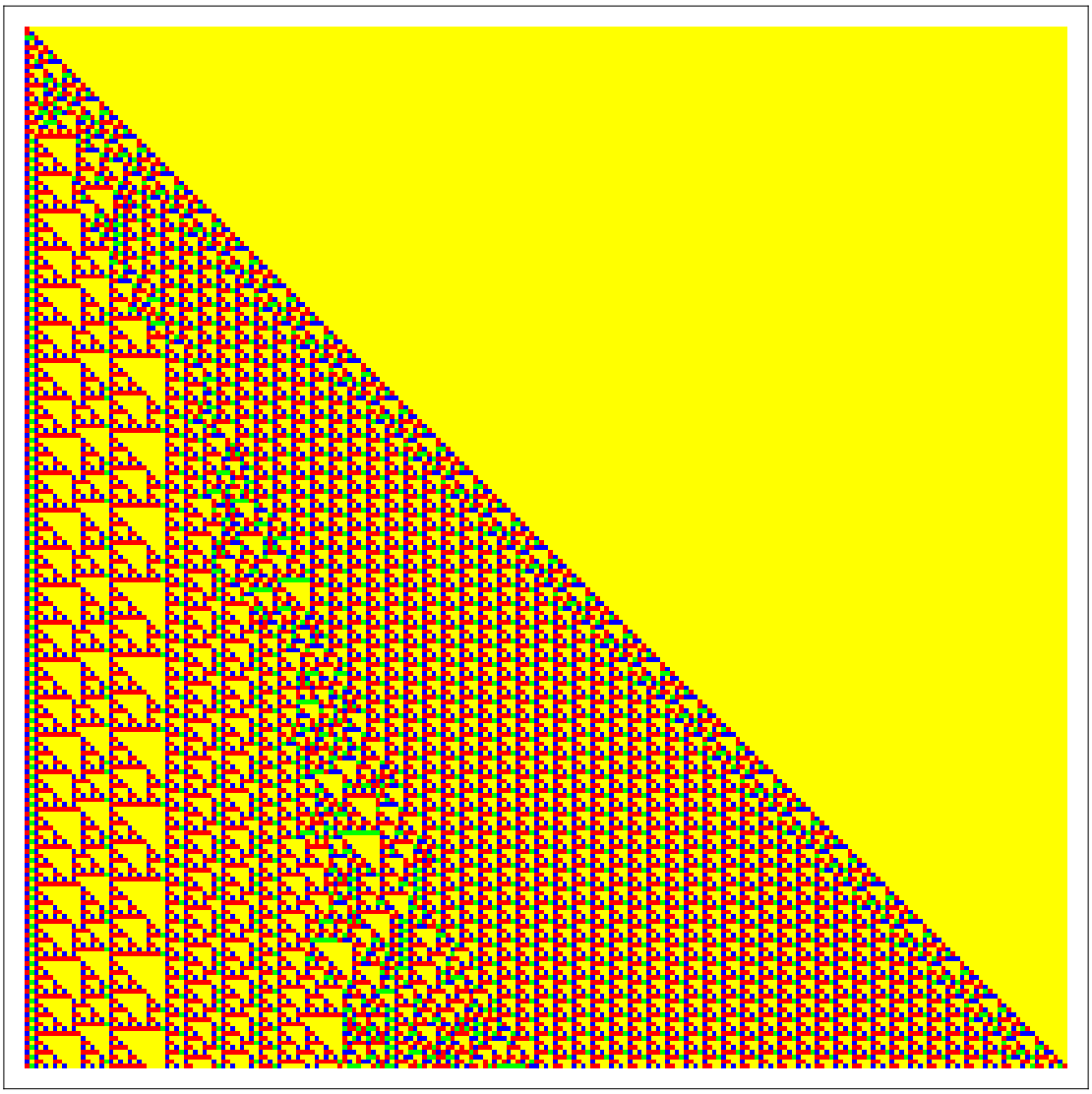


rule = 6.7 3 .13 15

(Debug) In[28]:=

```
ArrayPlot[CellularAutomaton[
  ruleM[{6, 7, 3, 13, 15}],
  {{1}, 0}, 222],
 ColorRules -> {1 -> Red, 0 -> Yellow, 2 → Blue, 3 → Green}]
```

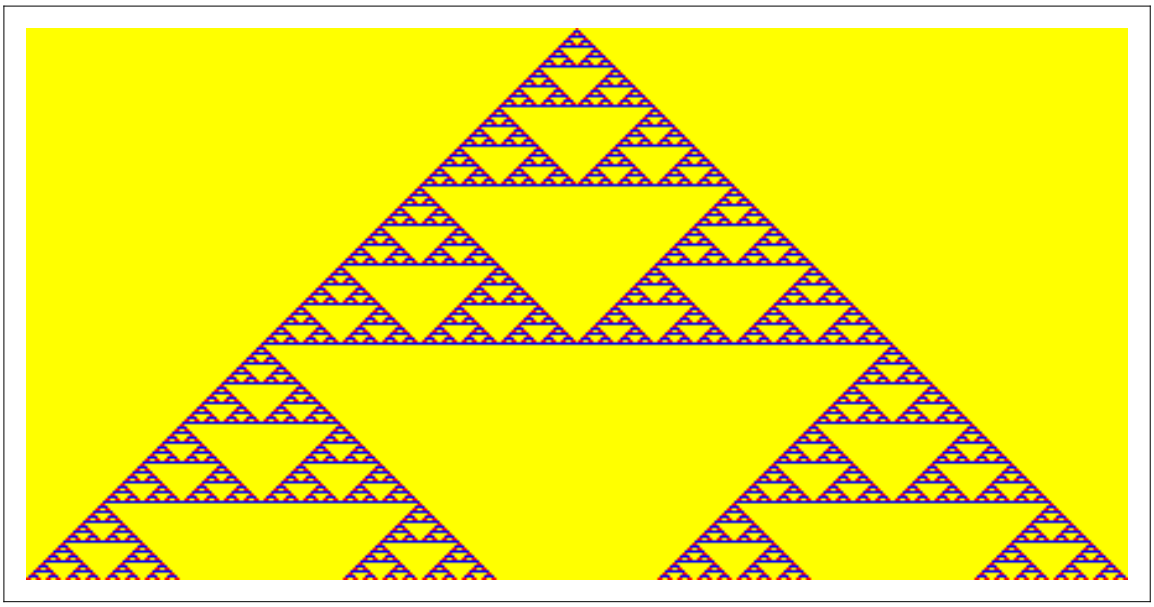
(Debug) Out[28]=



rule = 6.13 .11 12 15

(Debug) In[29]:=
```
ArrayPlot[CellularAutomaton[
  ruleM[{6, 13, 11, 12, 15}],
  {{1}, 0}, 222],
 ColorRules -> {1 -> Red, 0 -> Yellow, 2 → Blue, 3 → Green}]
```

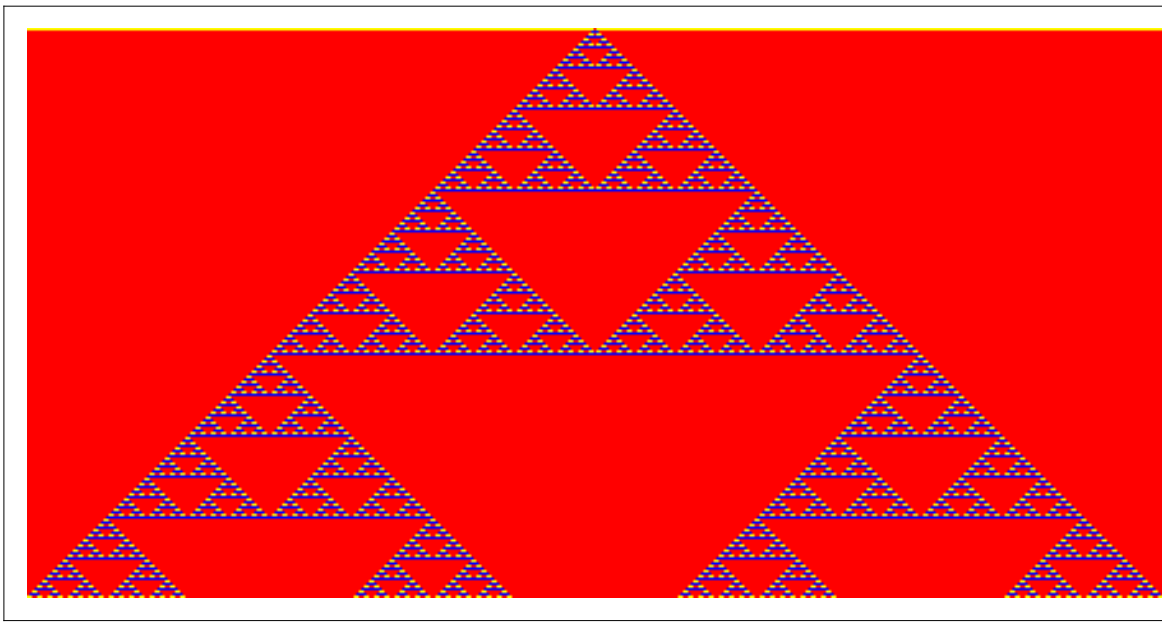
(Debug) Out[29]=



rule = 1.13.11.12.15

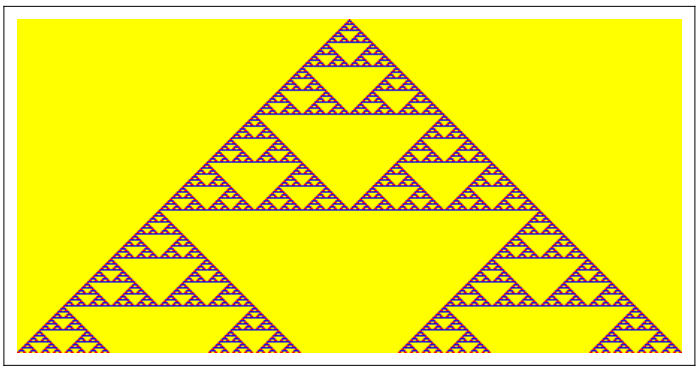
(Debug) In[30]:=
```
ArrayPlot[CellularAutomaton[
  ruleM[{1, 13, 11, 12, 15}],
  {{1}, 0}, 222],
 ColorRules -> {1 -> Red, 0 -> Yellow, 2 → Blue, 3 → Green}]
```

(Debug) Out[30]=



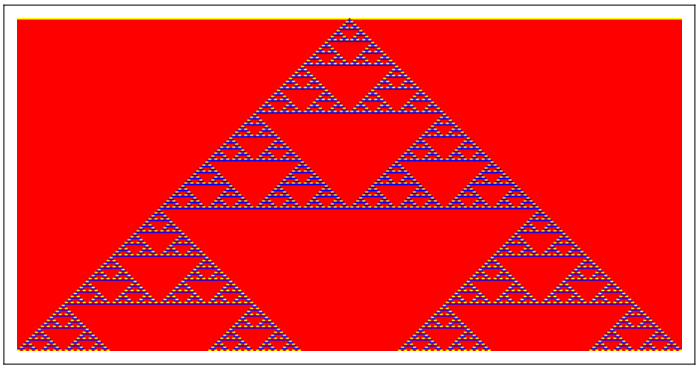
(Debug) In[31]:=
```
ArrayPlot[CellularAutomaton[
  ruleM[{6, 13, 11, 12, 15}],
  {{1}, 0}, 222],
 ColorRules -> {1 -> Red, 0 -> Yellow, 2 → Blue, 3 → Green}]
```

(Debug) Out[31]=



(Debug) In[32]:=
```
ArrayPlot[CellularAutomaton[
  ruleM[{1, 13, 11, 12, 15}],
  {{1}, 0}, 222],
 ColorRules -> {1 -> Red, 0 -> Yellow, 2 → Blue, 3 → Green}]
```

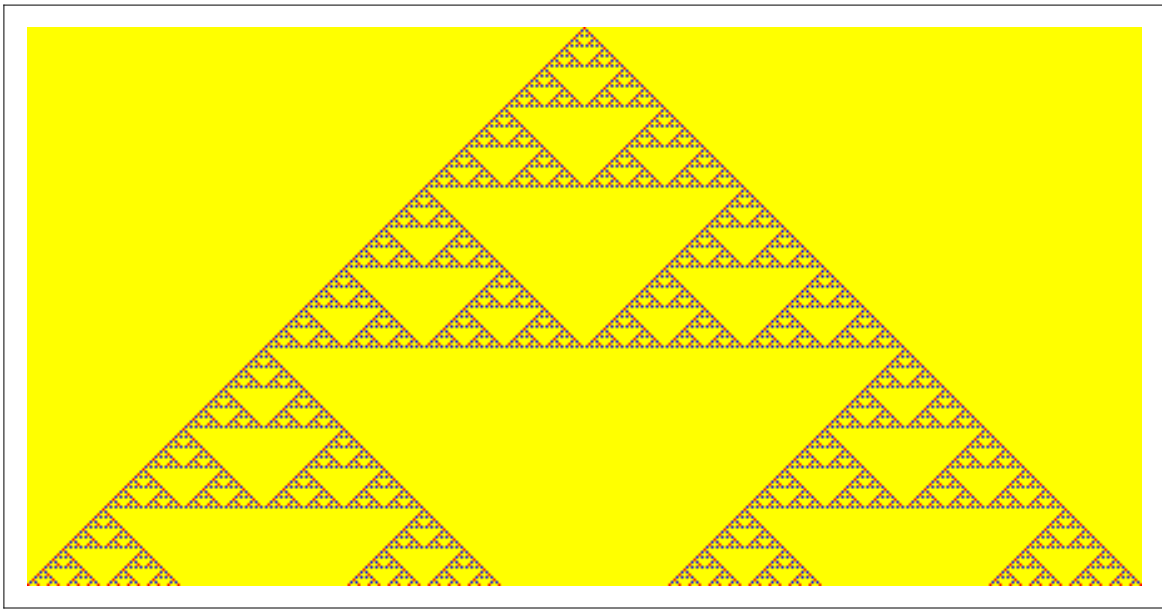
(Debug) Out[32]=



rule = 6.8 .11 .13 .14

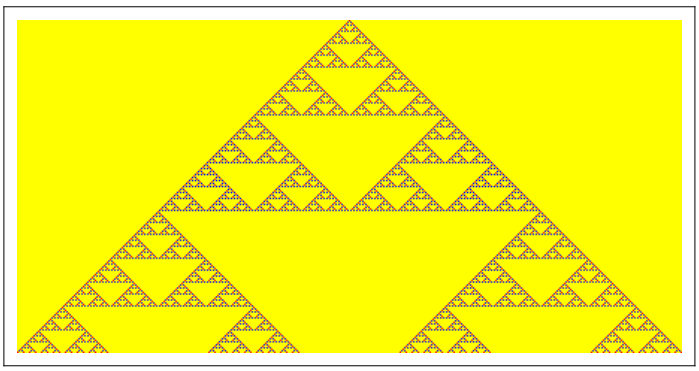
(Debug) In[33]:=
```
ArrayPlot[CellularAutomaton[
  ruleM[{6, 8, 11, 13, 14}],
  {{1}, 0}, 222],
 ColorRules -> {1 -> Red, 0 -> Yellow, 2 → Blue, 3 → Green}]
```

(Debug) Out[33]=



(Debug) In[34]:=
```
ArrayPlot[CellularAutomaton[
  ruleM[{6, 8, 11, 13, 14}],
  {{1}, 0}, 222],
 ColorRules -> {1 -> Red, 0 -> Yellow, 2 → Blue, 3 → Green}]
```

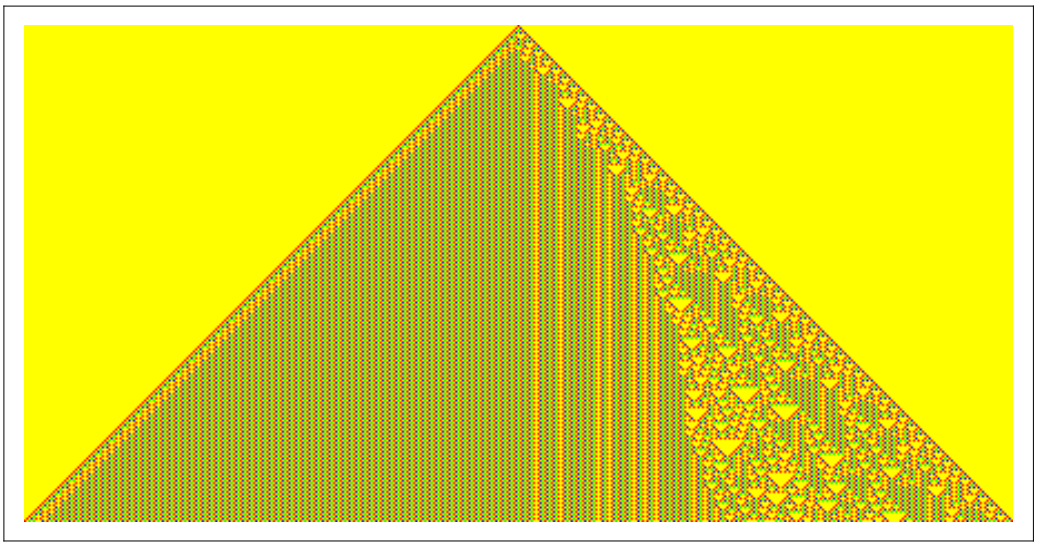
(Debug) Out[34]=



rule = 6.2.8.13.15

(Debug) In[35]:=
```
ArrayPlot[CellularAutomaton[
  ruleM[{6, 2, 8, 13, 15}] ,
  {{1}, 0}, 222],
 ColorRules -> {1 -> Red, 0 -> Yellow, 2 → Blue, 3 → Green}]
```

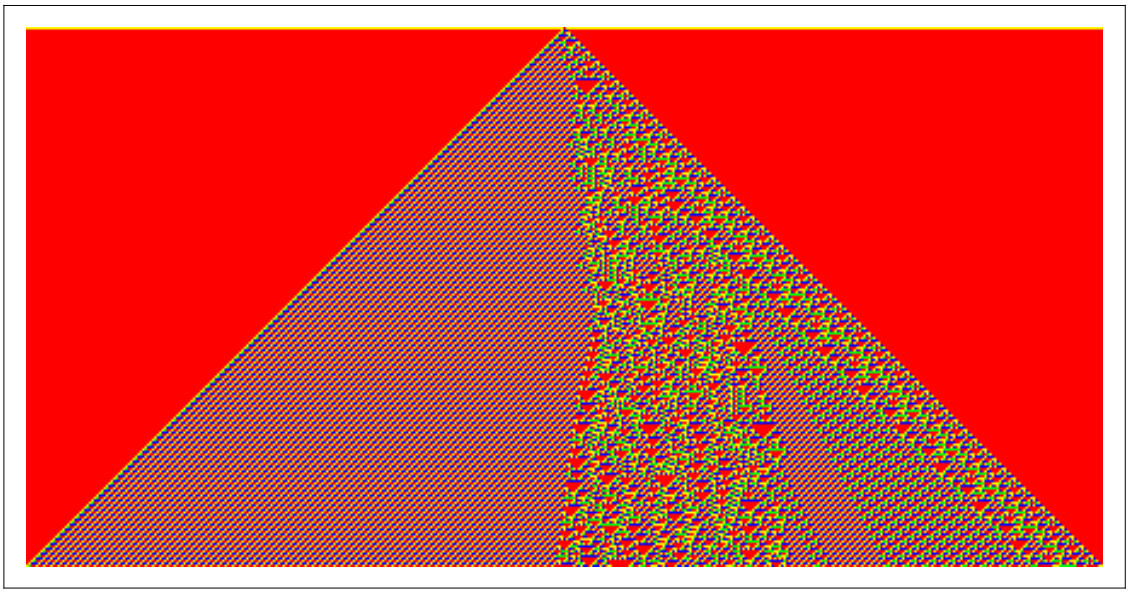
(Debug) Out[35]=



rule = 1.7.12.13.15

(Debug) In[36]:=
```
ArrayPlot[CellularAutomaton[
  ruleM[{1, 7, 12, 13, 15}],
  {{1}, 0}, 222],
 ColorRules -> {1 -> Red, 0 -> Yellow, 2 → Blue, 3 → Green}]
```

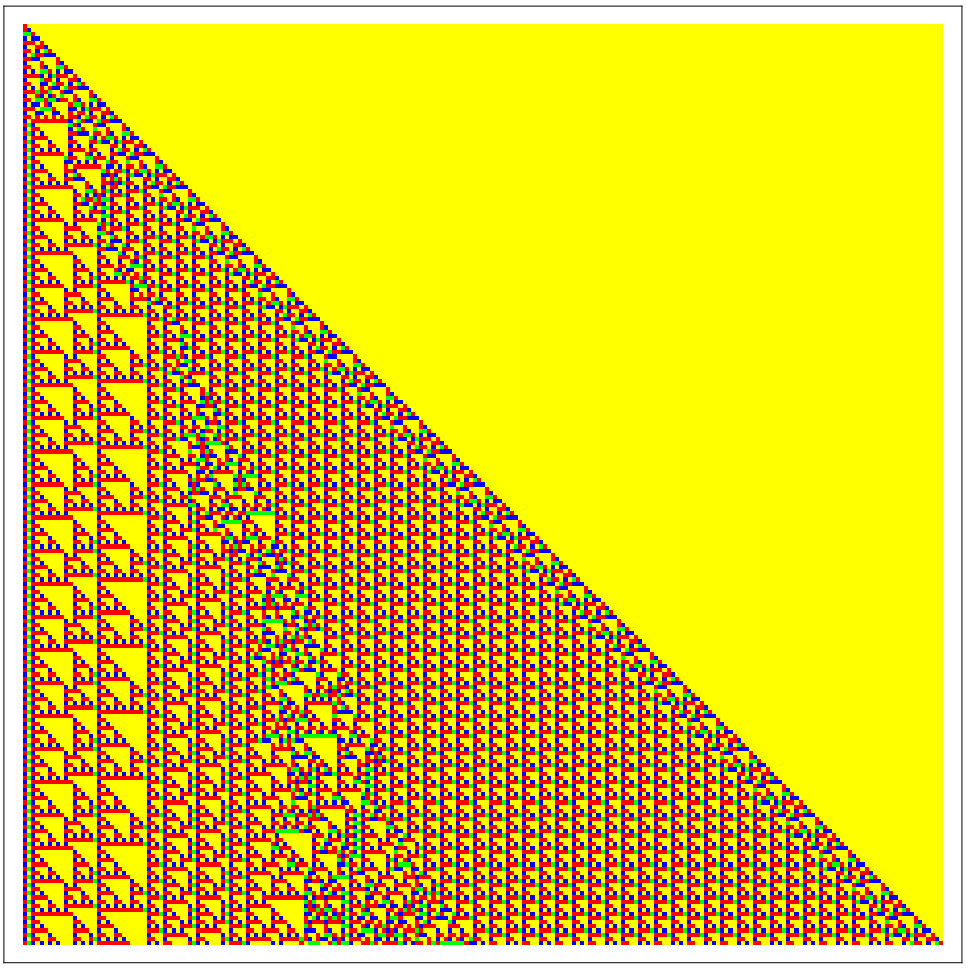
(Debug) Out[36]=



rule = 6.7.3.13.15

(Debug) In[37]:=
```
ArrayPlot[CellularAutomaton[
  ruleM[{6, 7, 3, 13, 15}],
  {{1}, 0}, 222],
 ColorRules -> {1 -> Red, 0 -> Yellow, 2 → Blue, 3 → Green}]
```

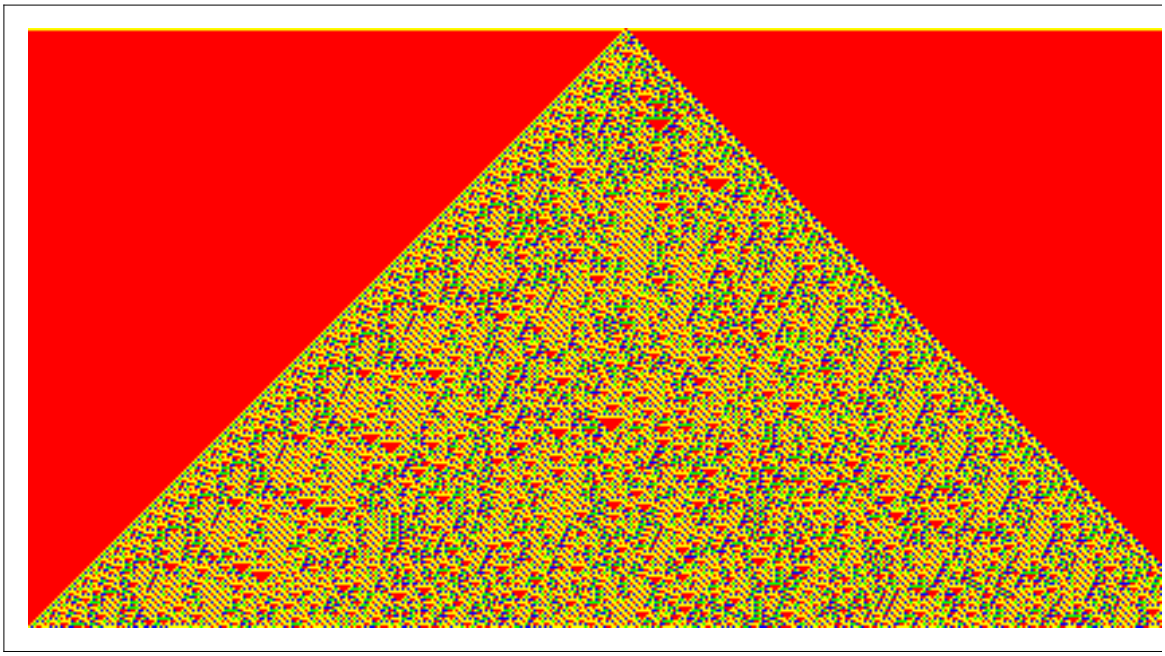
(Debug) Out[37]=



rule = 1.7.8.13.15

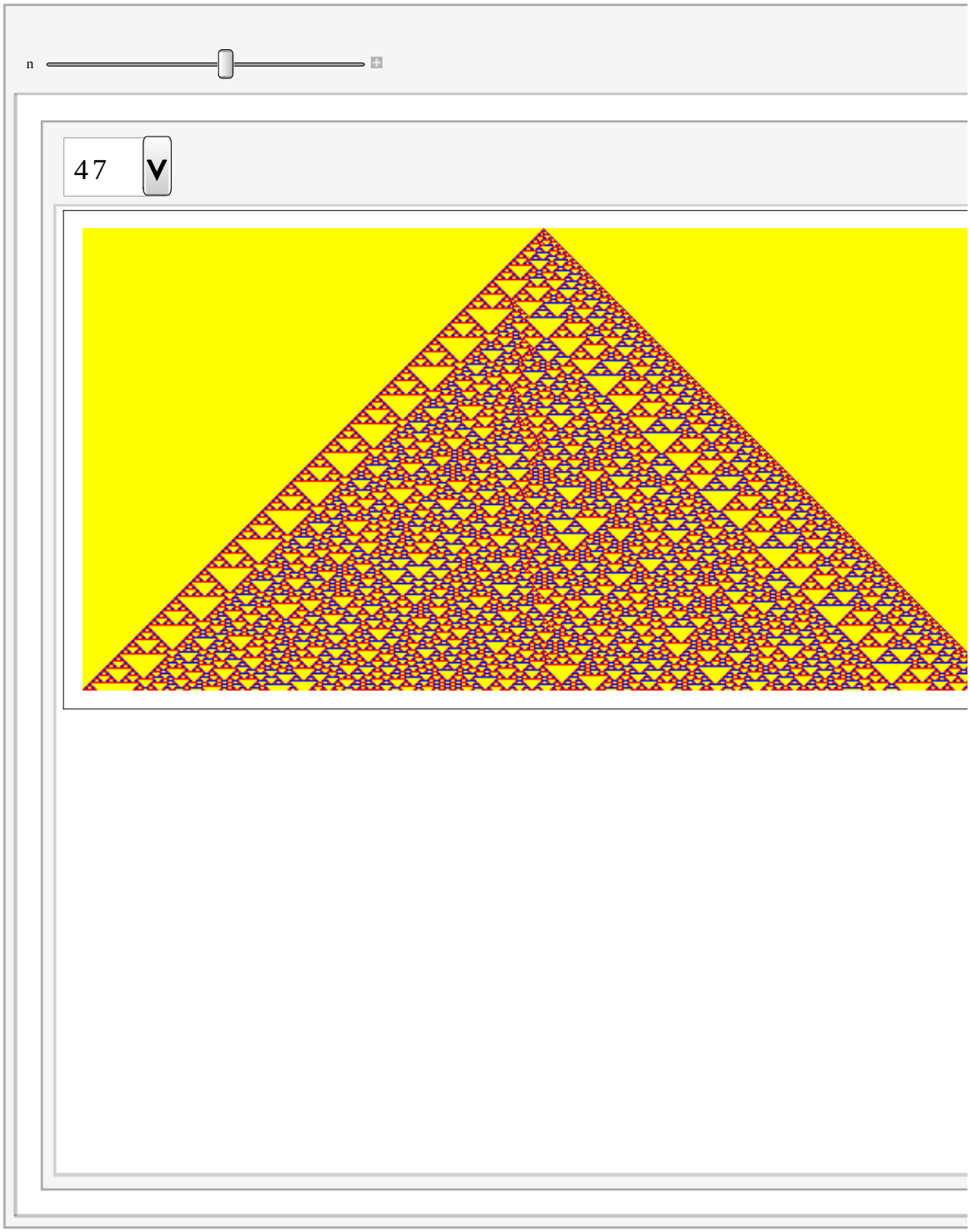
(Debug) In[38]:=
```
ArrayPlot[CellularAutomaton[
  ruleM[{1, 7, 8, 13, 15}],
  {{1}, 0}, 222],
 ColorRules -> {1 -> Red, 0 -> Yellow, 2 → Blue, 3 → Green}]
```

(Debug) Out[38]=



Dynamic presentation of $CA^{(4,3)}$ examples, 68 automata

(Debug) Out[82]=



Extended rules: ruleMN

(Debug) In[39]:=

```
ruleMN[{a_, b_, c_, d_, e_, f_}] :=
  Flatten[{rca[{a}], rca[{b}] , rca[{c}], rca[{d}] , rca[{e}], rca[{f}]}]
```

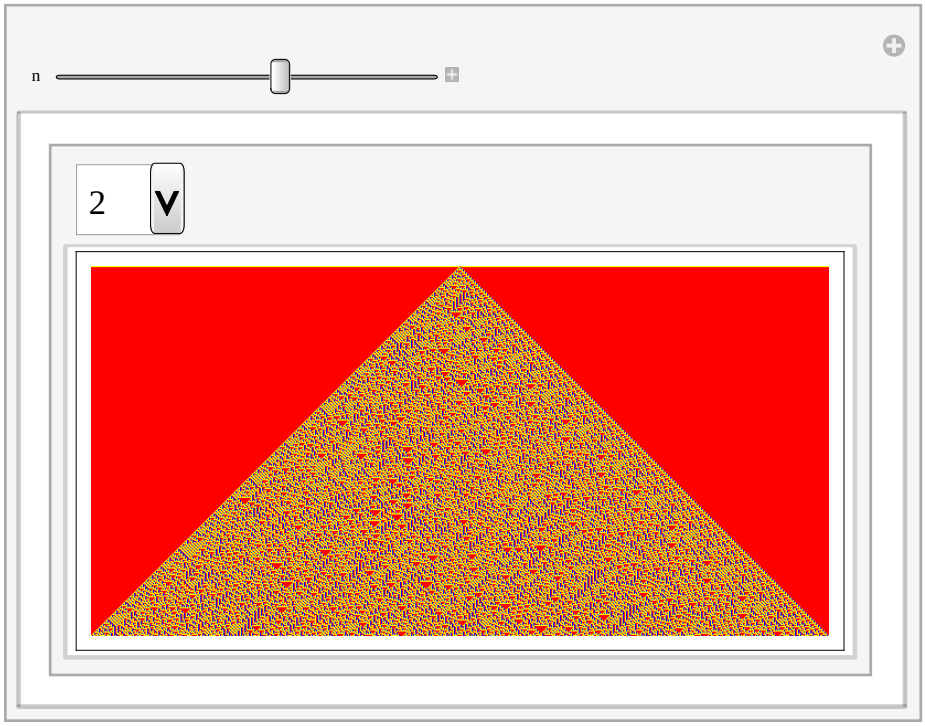
Examples

ruleMN[{1, 7, 8, 13,5, 15}],
ruleMN[{1, 7, 8, 13,10, 15}],

ruleMN[{1, 7, 8, 13,14, 15}]

SlideView of ruleMN examples

(Debug) Out[42]=

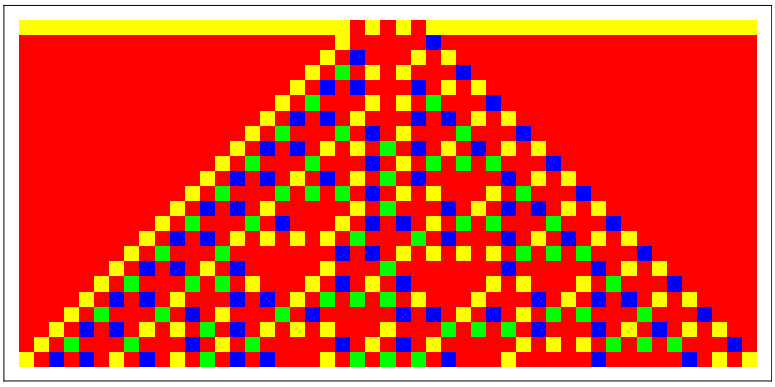


Different initial conditions

Multple seeds: {1,0,1,0,1}

(Debug) In[43]:=
```
ArrayPlot[CellularAutomaton[
  ruleM[{1, 7, 3, 13, 15}],
  {{1, 0, 1, 0, 1}, 0}, 22],
 ColorRules -> {1 -> Red, 0 -> Yellow, 2 → Blue, 3 → Green}, ImageSize → 400]
```

(Debug) Out[43]=

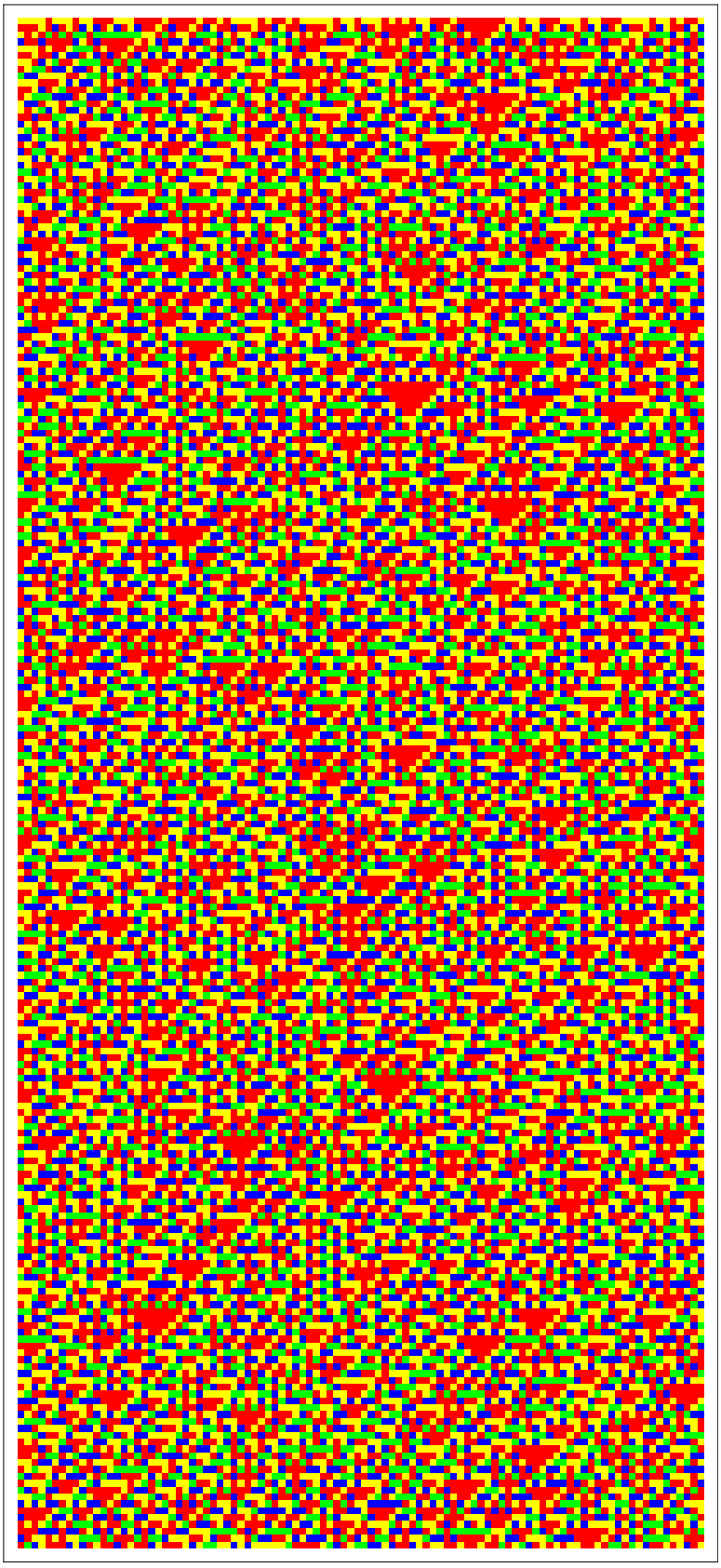


Random: [1,100]

(Debug) In[44]:=
```
ruleM[{1, 7, 3, 13, 15}]
```

(Debug) In[45]:=
```
ArrayPlot[CellularAutomaton[
  ruleM[{1, 7, 3, 13, 15}],
  RandomInteger[1, 100], 222],
 ColorRules -> {1 -> Red, 0 -> Yellow, 2 → Blue, 3 → Green}, ImageSize → 400]
```

(Debug) Out[45]=



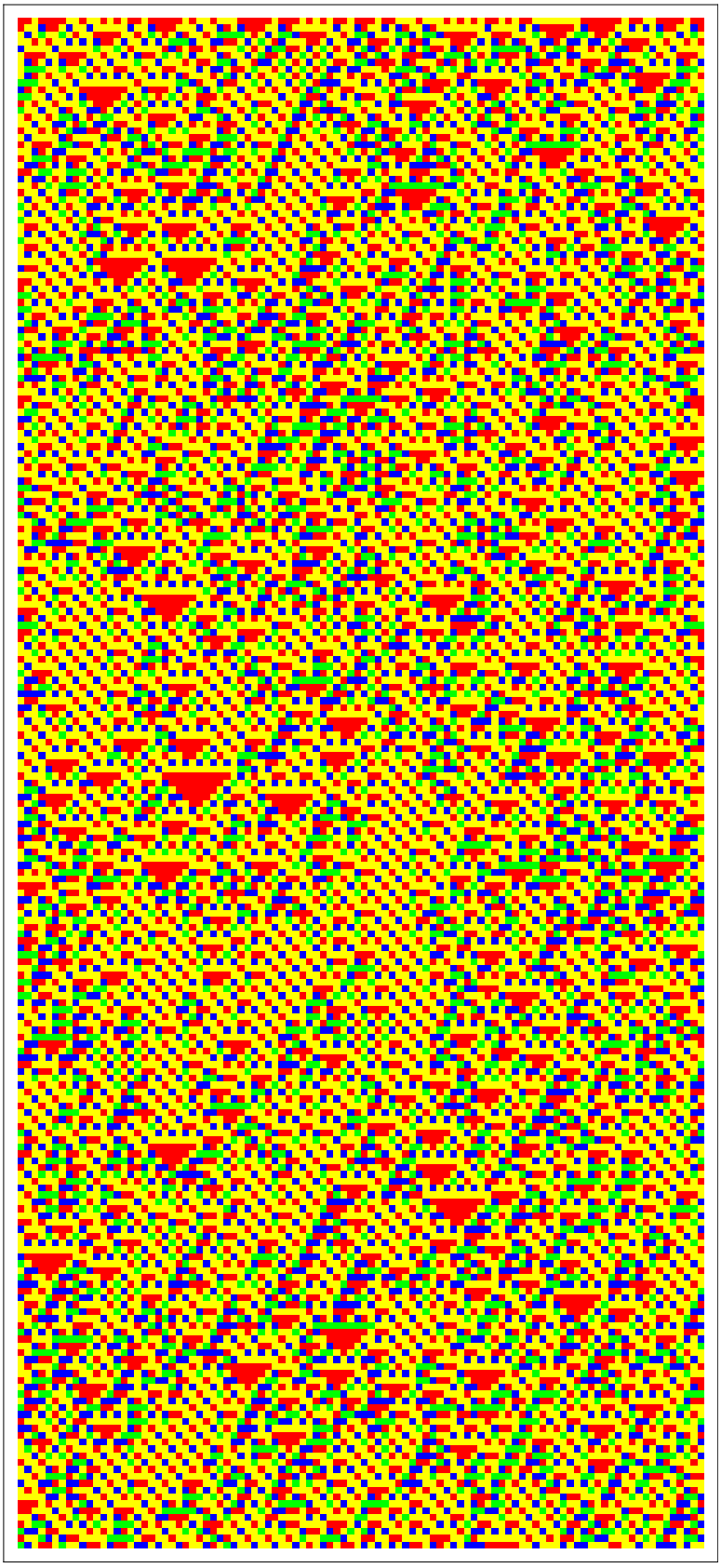
rule test

(Debug) In[46]:=

```
ruleM[{1, 7, 3, 13, 15}]
```

(Debug) In[47]:=
```
ArrayPlot[CellularAutomaton[
  ruleM[{1, 7, 8, 13, 15}],
  RandomInteger[1, 100], 222],
 ColorRules -> {1 -> Red, 0 -> Yellow, 2 → Blue, 3 → Green}, ImageSize → 400]
```

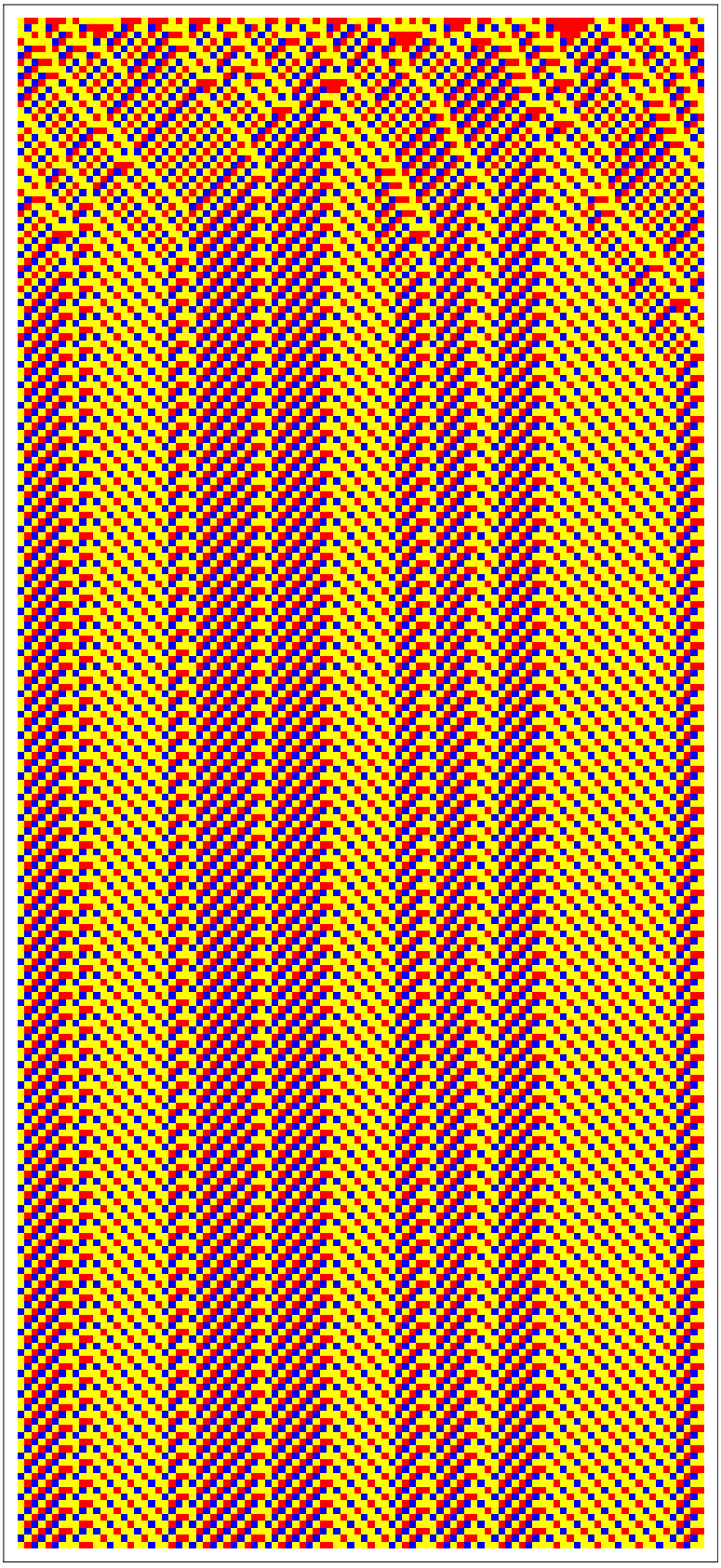
(Debug) Out[47]=



ruleMN[{1,7,8,13,14,15}], 222, random, example 1

(Debug) In[48]:=
```
ArrayPlot[CellularAutomaton[
  ruleMN[{1, 7, 8, 13, 14, 15}],
  RandomInteger[1, 100], 222],
 ColorRules -> {1 -> Red, 0 -> Yellow, 2 → Blue, 3 → Green}, ImageSize → 400]
```

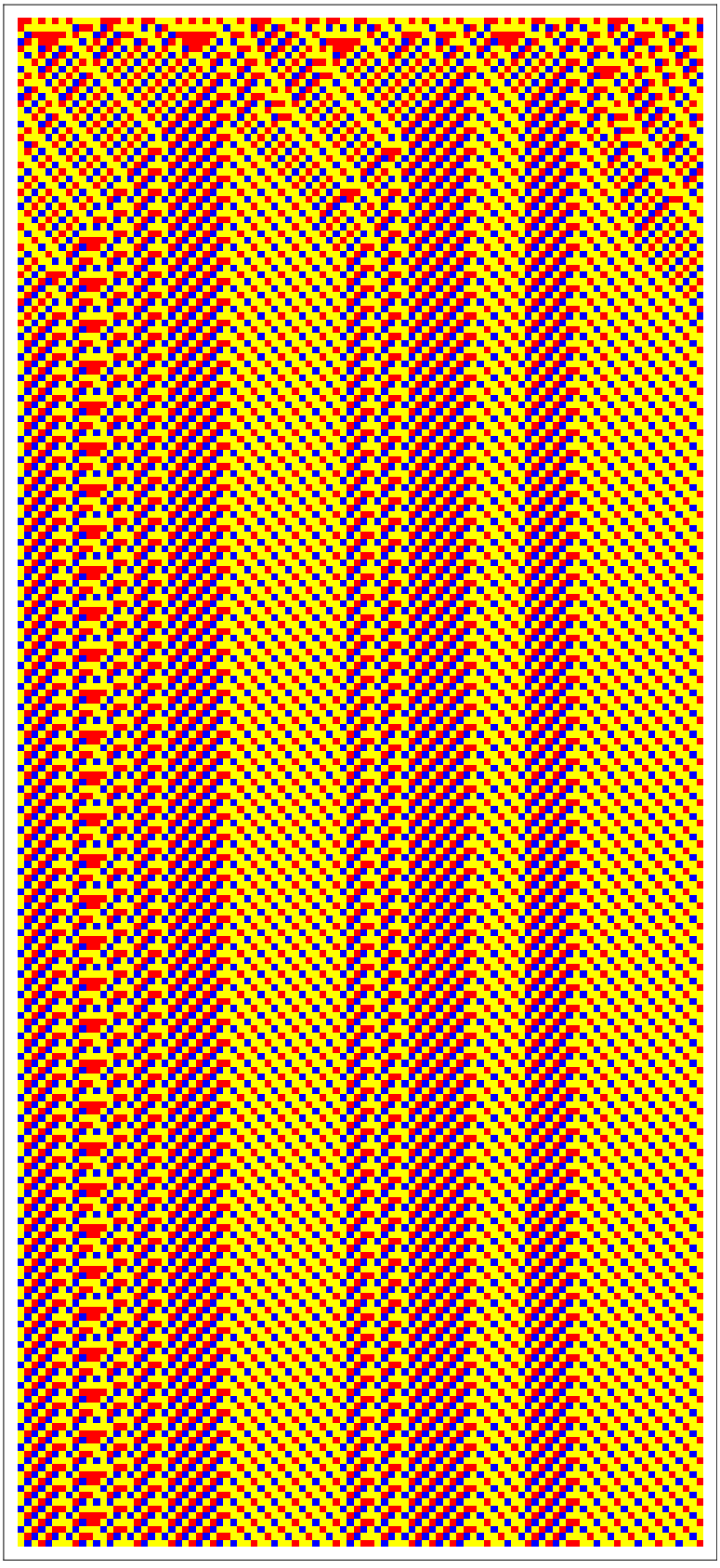
(Debug) Out[48]=



ruleMN[{1,7,8,13,14,15}], 222, random, example 2

(Debug) In[49]:=
```
ArrayPlot[CellularAutomaton[
  ruleMN[{1, 7, 8, 13, 14, 15}],
  RandomInteger[1, 100], 222],
 ColorRules -> {1 -> Red, 0 -> Yellow, 2 → Blue, 3 → Green}, ImageSize → 400]
```

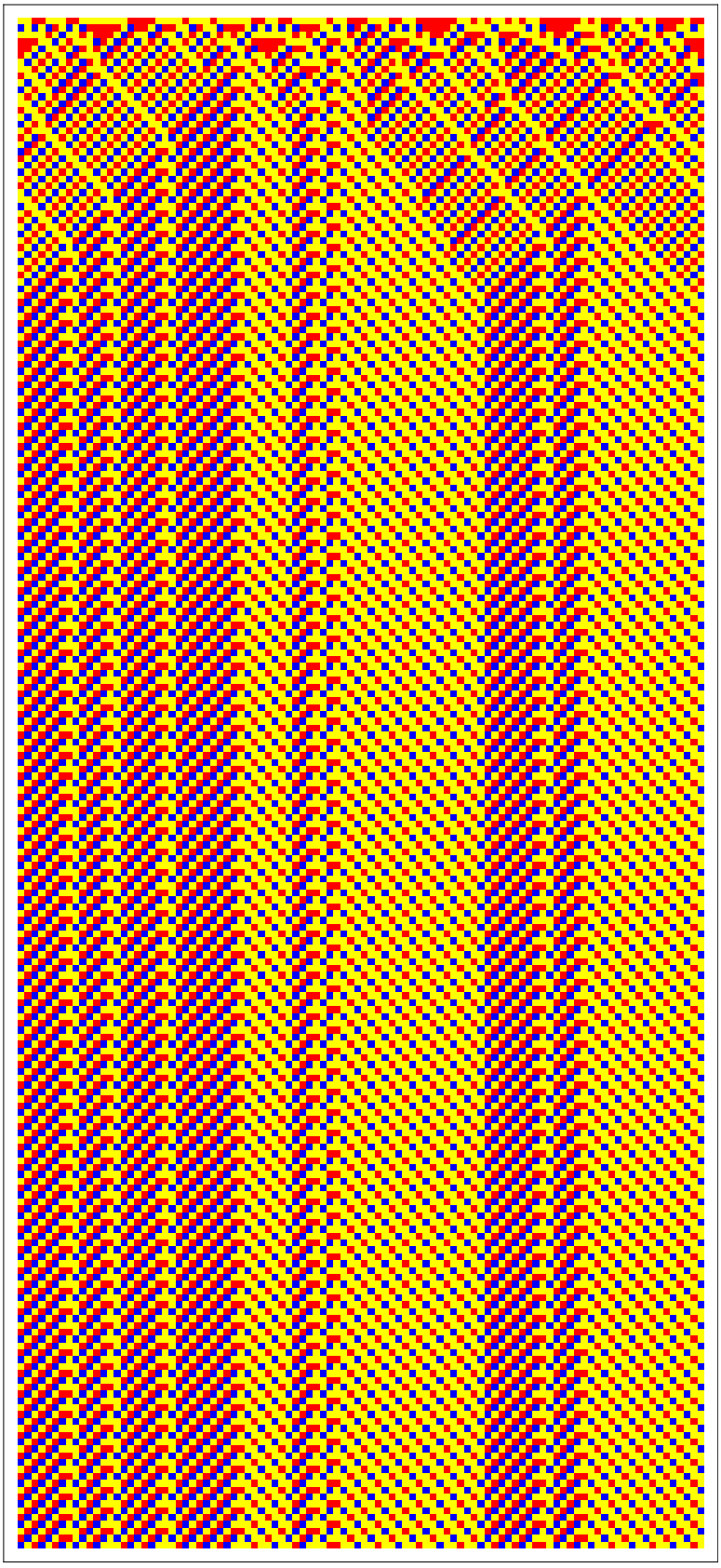
(Debug) Out[49]=



ruleMN[{1,7,8,13,14,15}], 222, random, example 3

(Debug) In[50]:=
```
ArrayPlot[CellularAutomaton[
  ruleMN[{1, 7, 8, 13, 14, 15}],
  RandomInteger[1, 100], 222],
 ColorRules -> {1 -> Red, 0 -> Yellow, 2 → Blue, 3 → Green}, ImageSize → 400]
```

(Debug) Out[50]=



rule test

(Debug) In[51]:=

```
ruleMN[{1, 7, 8, 13, 14, 15}]
```

Towards analysis

Steps from n to m

from to [111, 222]

(Debug) In[75]:=
```
ArrayPlot[CellularAutomaton[
  ruleM[{1, 7, 3, 13, 15}],
  RandomInteger[1, 100], {{111, 222}}],
 ColorRules -> {1 -> Red, 0 -> Yellow, 2 → Blue, 3 → Green}, ImageSize → 400]
```

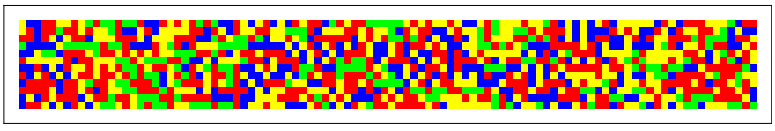
(Debug) Out[75]=

Selections

Every tenth step from 111 to 222

(Debug) In[76]:=
```
ArrayPlot[CellularAutomaton[
  ruleM[{1, 7, 3, 13, 15}],
  RandomInteger[1, 100], {{111, 222, 10}}],
 ColorRules -> {1 -> Red, 0 -> Yellow, 2 → Blue, 3 → Green}, ImageSize → 400]
```

(Debug) Out[76]=

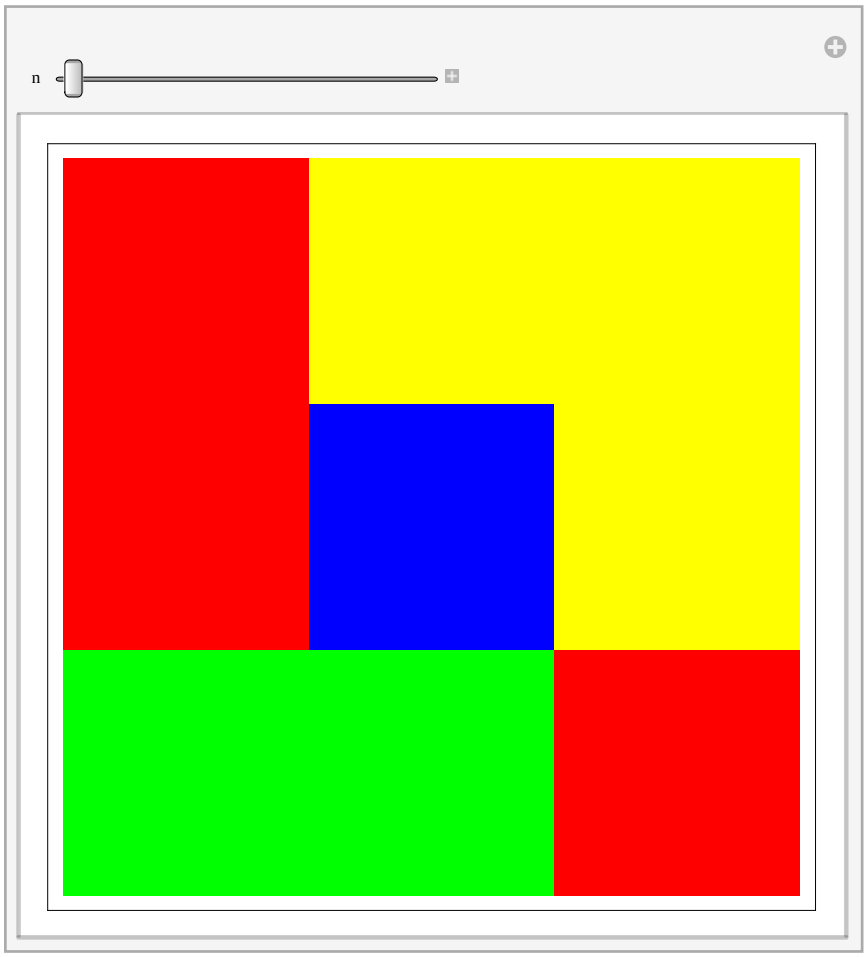


Dynamic range of steps

rule = 6.7 .3 .13 .15, range = 555

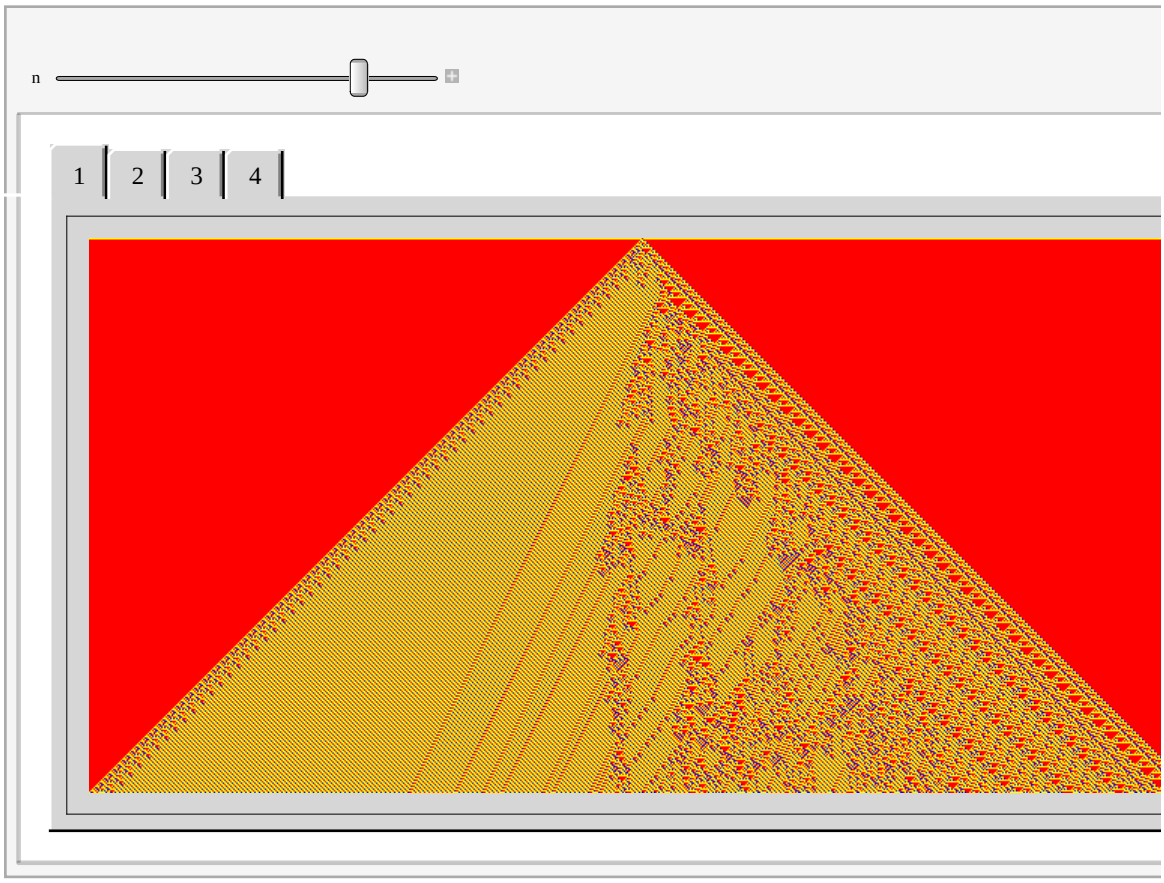
(Debug) In[77]:=
```
Manipulate[ArrayPlot[CellularAutomaton[
   ruleM[{6, 7, 3, 13, 15}], {{1}, 0}, n],
  ColorRules -> {1 -> Red, 0 -> Yellow, 2 → Blue, 3 → Green},
  ImageSize → 400], {n, 2, 555, 1}]
```

(Debug) Out[77]=



Differentiations for [1, 7, 8, 13, -], with dynamic step range

(Debug) Out[74]=



rule = 6.7 .3 .13 .15, seed = {1, 0, 1, 0, 1}

(Debug) Out[78]=