Canonical Computations in Cellular Automata and Their Application for Reservoir Computing

Trym A. E. Lindell[1], Barbora Hudcová[34], Stefano Nichele[12]

[1] Artificial Intelligence Lab, Department of Computer Science, Faculty of Technology Art and Design, Oslo Metropolitan University, Oslo, Norway
[2] Machine Learning Research Group, Østfold University College, Halden, Norway
[3] Department of Algebra, Faculty of Mathematics, Charles University, Prague, Czech Republic
[4] Czech Institute of Informatics, Robotics and Cybernetics, CTU, Prague, Czech Republic
trym.lindell@oslomet.no

Abstract

Cellular Automata (CAs) have potential as powerful parallel computational systems, which has lead to the use of CAs as reservoirs in reservoir computing. However, why certain Cellular Automaton (CA) rules, sizes and input encodings are better or worse at a given task is not well understood. We present a method that enables identification and visualization of the specific information content, flow and transformations within the space-time diagram of CA. We interpret each spatio-temporal location in CA's space-time diagram as a function of its input and call this novel notion the CA's Canonical Computations (CCs). This allows us to analyze the available information from the space-time diagrams as partitions of the input set. The method also reveals how input-encoder-rule interactions transform the information flow by changing features like spatial and temporal location stability as well as the specific information produced. This general approach for analysing CA is discussed for the engineering of reservoir computing systems.

Introduction

Cellular automata have the ability to both produce immense complexity from relatively simple rules as well as reduce entropy through self organization (Wolfram, 1983). The space-time diagrams of CA can produce complex patterns that vary from repetitive to chaotic, and the dynamics that produce these patterns have been shown to be able to transmit information as well as store and compute on it (Lizier et al., 2012). Some CA have even been shown to be Turing complete (Cook et al., 2004). These capacities have led researchers to attempt to use CA to solve a range of different computational tasks (Mitchell et al., 2005).

A relatively novel approach to this is found in the field of artificial intelligence where CA have been used as reservoirs within reservoir computing systems (ReCA) (Yilmaz, 2014; Nichele and Molund, 2017; Nichele and Gundersen, 2017; Kleyko et al., 2017; McDonald, 2017; Morán et al., 2018). This approach has multiple beneficial features compared to traditional AI, like high energy efficiency and direct hardware implementations (Yilmaz, 2014). While some guidelines may be derived for designing ReCA system based on CA rule classifications and parameterizations like Wolfram's four classes (Wolfram, 1984) or Langton's lambda parameter (Langton, 1990), designing reservoir computing systems using CA is challenging (Glover et al., 2021) because the effect of the relevant ReCA parameters on the local computations of the CAs has not been explored at a detailed level. More fine grained understanding of this topic is thus needed.

In this paper we develop a method to identify the exact information content a given CA computes within its space-time diagrams. Furthermore, we present a mathematical formulation of how the CA transition rule operates on this information to transform it from one time step to the other. Given a set of input that is encoded into the CA's configurations, our proposed method shows that CAs locally compute functions of the encoded information. These functions induce partitions of the input set, a notion we name canonical computations. ReCA systems can then combine this local information contained in the partitions to produce desired outputs. Insights gained from this method is then used to suggest reservoir system and CA choices that may improve performance on computational tasks.

Background

Cellular Automata

Cellular automata consist of a d-dimensional grid $\mathbb{Z}^d$ whose elements are called cells. Each cell can be in a state from a finite set S; each ensemble $c \in S^{\mathbb{Z}}$ is called a configuration. Every cell is further assigned a uniform, finite neighborhood of cells. In this paper, we focus on the class of 1-dimensional nearest-neighbor CAs; for such, the neighborhood of cell $i \in \mathbb{Z}$ is simply $(i-1, i, i+1)$. The dynamics of the CA is given by a local rule $f : S^3 \to S$ which updates the states of all cells in parallel. This gives rise to the global rule $F : S^{\mathbb{Z}} \to S^{\mathbb{Z}}$ given for each $c \in S^{\mathbb{Z}}$ and each $i \in \mathbb{Z}$:

$$F(c)_i = f(c_{i-1}, c_i, c_{i+1}). \tag{1}$$

Furthermore, we restrict our analysis to the simplest form of CAs which are called Elementary Cellular Automata (ECAs) and are limited to state values in $\{0, 1\}$. The limitations imposed on state values, neighbourhood size and dimensionality results in a limited number of transition rules,

namely 256. These rules are enumerated according to Wolfram's naming scheme defined in Wolfram (1983).

The rules produce different types of behaviour that can be classified according to their dynamics. These classifications can often be used to guide rule selection for computational tasks as certain rule types show complex behavior that may be conducive to this purpose while others do not. Multiple classification schemes and metrics exists, with the ECAs informally classified by Wolfram into four classes (Wolfram, 1984) according to how they evolve over time. Other approaches have grouped CA based on features like compression of space-time diagrams (Zenil, 2009) and transient length (Hudcová and Mikolov, 2020), while others have used parametrizations (Langton, 1990; Wuensche et al., 1992)

Reservoir Computing

Reservoir computing is a form of artificial intelligence system that initially arose in 2001 and 2002 through the parallel work of Herbert Jaeger and Wolfgang Maas et al. in the form of Echo State Networks (ESN) (Jaeger, 2001) and Liquid State Machines (LSM) (Maass et al., 2002). The main components of a typical reservoir computing system consists of an untrained reservoir and a trained readout. The readout or decoder is generally some linear machine learning system, like a support vector machine or ridge regression, that reads out the states of the reservoir and transforms it into the desired output. The reservoir serves as a form of memory and computational substrate which allows an input to be memorized while also allowing its dynamics to perform computations that utilizes the memorized input. This provides two benefits to the readout; firstly it allows the readout to capture entire sequences of a time series in a snapshot of the dynamics of the reservoir. This is akin to how the waves of a pond, caused by throwing pebbles in it, can tell us when, where, and how many pebbles were thrown at the pond based on its wave patterns. The second benefit comes from the potential of the reservoir to transform linearly inseparable input to linearly separable reservoir states. Thus a simple linear learning mechanism can be used, which reduces the energy needs of the training process drastically compared to the otherwise required non-linear readout. In Figure 1 we show an overview of the components of a classical reservoir computing system using artificial neural networks. An encoder inputs the input example into the states of the reservoir and the effects are propagated throughout the recurrent reservoir while being decoded by the decoder.

The main task in reservoir computing research is finding and optimizing good reservoirs. Multiples systems in addition to the classical artificial neural networks used in echo-state networks and liquid state machines, have been tested for various tasks, including in-vitro neural networks (Aaser et al., 2017), a bucket of water (Fernando and Sojakka, 2003) and an anesthetized cat (Nikolić et al., 2006) as well as cellular automata.

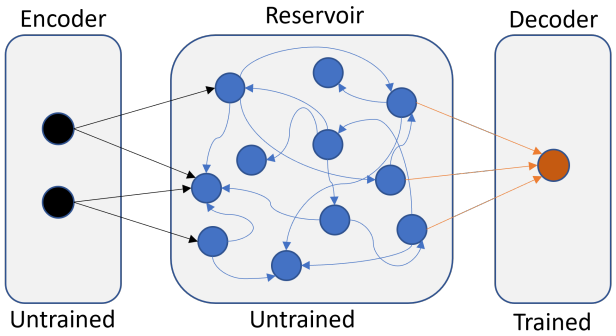

Figure 1: Classical reservoir computing system consisting of an untrained encoder and reservoir and a trained decoder. The reservoir consists of a recurrent neural network.

ReCA

Reservoir computing systems utilizing CAs as the reservoir component was proposed by Yilmaz (2014) due to CAs' computations being drastically cheaper than the artificial neurons used in the classical reservoir types. CAs can furthermore be implemented in Field Programmable Gate Arrays (FPGAs) and GPUs to provide additional increases in computation speed.

The initial experiments tested CA reservoirs on the 5 and 20 bit memory tasks and demonstrated good performance. Later studies have expanded on the original single CA reservoir system using both deep (Nichele and Molund, 2017) and non-uniform CAs (Nichele and Gundersen, 2017). ReCA have furthermore been tested on a range of tasks like medical image modality classification (Kleyko et al., 2017), square vs sine wave and iris data-set classification, non-linear channel equalization reconstruction and chaotic laser time-series prediction (McDonald, 2017) and lastly MNIST handwritten digit classification (Morán et al., 2018).

Designing ReCa systems involves making a set of choices concerning the encoder, the CA itself and the decoder (Glover et al., 2021). An overview of a ReCA system and its components is given in Figure 2 where the neural network of the classical reservoir is exchanged with a CA. We here list and describe the relevant components and choices.

Encoder The encoder transplants the input features of an input example into the configurations of the CA. This can be done either by encoding the input into the initial configuration of the CA or they can be input over time as is the case in the 5-bit memory task (Yilmaz, 2014; Glover et al., 2021).

In addition one can encode the input directly into the configuration if the input and the CA size matches. If the CA

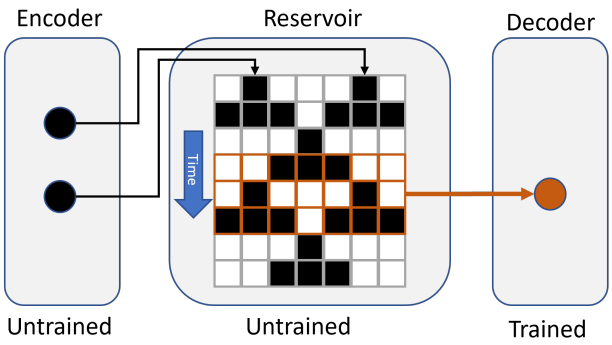

Figure 2: ReCa system encoding the input into the initial configuration of an ECA. A subset of the space-time diagram (indicated in orange) is given to the readout each time step.

is larger than the input, the input features may be mapped to different locations in the CA configuration. This can be done with redundancy such that the same feature is copied multiple times. One may also systematically parameterise the distance between each encoding location.

For temporal input injection, one must also choose how many time steps one should allow the CA to compute before injecting the next feature of the input example.

Lastly one must make a choice concerning how the values of the feature dimensions of the input interacts with the cell states of the CA. The simplest method is to copy the value of the feature onto the encoding cell. Alternatively, one can for example use an XOR encoding to ensure that the feature value is more probable to have an effect on the configuration.

CA parameters The obvious choice one must make is which CA to implement as a reservoir and whether to use its classical version or to use a non-standard architecture such as deep or non-uniform. One must also decide on using closed or open boundary conditions, i.e whether the grid is cyclical or not. Furthermore, one must choose the spatial size of the CA and how many time steps one shall run it for.

Decoder The decoder is often some linear machine learning system although an alternative method based on hyperdimensional computing has also been suggested (Kleyko et al., 2017).

Another choice that must be made for the decoder is the readout method. One may run the CA for some number of iterations to produce a space-time diagram and use its entirety directly as input to the decoder. Alternatively, one may use only a subset of the space-time diagram at the time by using a moving window over the time dimension as input to the decoder.

Related Works

Information processing in CAs has been explored from multiple perspectives. A profound body of work explored how to construct cellular automata capable of implementing Boolean circuits in various ways (Codd and Ashenhurst, 1968; Banks, 1971). A more closely related line of work studies the computation capacity in given family of automata, rather than constructing new CAs. This has been done, e.g., by assessing how many automata a particular CA can simulate (Israeli and Goldenfeld, 2006; Hudcová and Mikolov, 2021), by exploring the transformational effect a local area of a CA has on another finite area (Biehl and Witkowski, 2021), or by measuring the capacity of ECAs to implement simple mappings with one input bit and assesing the ECA's semantic capacity (Dittrich, 2018). Intrinsic computations in CA has also been analysed in (Feldman et al., 2008) by assessing the statistical properties of such systems via complexity-entropy diagrams.

We highlight that in contrast, we not only measure the CA's capacity to implement specific functions, but also visualize how the CA's capacity evolves over time. We argue that studying the evolution of CA computations is a novel perspective to be explored.

Canonical Computations in CAs

Local Information Computation

In a ReCA system the decoder must utilize the information available in the space-time diagrams produced by a given encoder-CA combination to yield the desired output. The question then is, what information is available?

To answer this question we first observe that the space-time diagrams of ECAs are binary and entirely deterministically produced. This means that any spatio-temporal location is in the "on" state for some partition of the input set and in the "off" state for the complimentary partition. Since each partition is mapped to a Boolean state value this corresponds directly to a Boolean function. We call such functions performed by the spatio-temporal locations the canonical computations of the CA.

Thus, the information available from a given cell is whether the input example was in one or the other partition of the input set. In other words, from a reservoir computing perspective, the canonical computation performed by an ECA reservoir is to partition input sets through Boolean functions. Each cell then potentially computes a different partition which the readout can use to produce a desired output. A simple case of linearlization within this framework would be the case when a partition produced by the ECA corresponds directly to a desired output categorization of the input set.

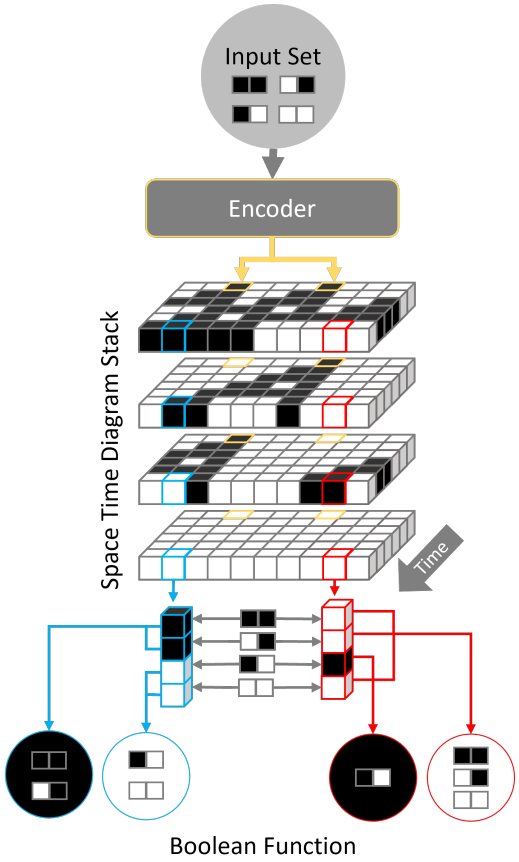

Figure 3: Boolean functions computed by an encoder-CA using rule 110 for the input set $\{0,1\}^2$. We here use a fixed encoder which maps the the two input features to two spatial locations in the initial configuration of the CA, highlighted in yellow. Each 2D space time diagram in the stack represents a different initial configuration based on different input configurations. The diagrams are stacked such that the spatio-temporal locations align. This produces a binary string at each location, two are highlighted in blue and red. These correspond to two different Boolean functions on the input set as indicated in the bottom of the figure. The input configurations contained within each colored circle indicates that these map to the value indicated by the color of the circle. Repeating the process of identifying the computed boolean function across all cells in the space-time diagram, associating these to labels and labeling each spatio-temporal location produces a Canonical Computation Diagram (CC-diagram) as indicated in Figure 4

In Figure 3 we illustrate this idea for the set of binary vectors of size 2. The input examples are encoded into the initial configurations of the ECA, highlighted in yellow, which is then ran for five time steps. The four possible space-time diagrams are stacked on top of each other such that the spatio-temporal locations align. Each position corresponds to a Boolean function of the inputs, and we highlight two in blue and red.

Formal Definition of Canonical Computations

Let us consider a CA reservoir task with inputs from the set $\{0,1\}^k$. There are different schemes for encoding the inputs into CA space-time diagrams.

Spatial encoding A spatial encoding of k input bits into a cyclic grid of size n is simply a mapping $\text{enc} : \{0,1\}^k \to \{0,1\}^n$.

Let $\mathcal{A}$ be an ECA with local rule f, and global rule F operating on a cyclic grid of size n and let us fix an encoding $\text{enc} : \{0,1\}^k \to \{0,1\}^n$. Each sequence of input bits $\mathbf{b} \in \{0,1\}^k$ induces a trajectory

$$\text{enc}(\mathbf{b}), F(\text{enc}(\mathbf{b})), F^2(\text{enc}(\mathbf{b})), ...$$

We define the *canonical computation* at position $i \in \{0,\dots,n-1\}$ and time-step t to be the function:

$$g_{i,t}(\mathbf{b}) = F_i^t\left(\text{enc}(\mathbf{b})\right) \tag{2}$$

for any input $\mathbf{b} \in \{0,1\}^k$.

In this context, we can view the ECA as evolving the set of available canonical computations as opposed to evolving the static configurations. We formalize this in Observation 1.

Observation 1 (CA Dynamics of Computation)**.** *Let $\mathcal{A}$ be an ECA with local rule f and global rule F operating on a finite cyclic grid of size n, and let* $\text{enc} : \{0,1\}^k \to \{0,1\}^n$ *be an encoding of inputs. Then, the canonical computations of $\mathcal{A}$ at time-step $t+1$ are obtained solely from the canonical computations at time-step t and the local rule f.*

Proof. Let us fix a position $i \in \{0,\dots,n-1\}$, time-step t, and input $\mathbf{b} \in \{0,1\}^k$. Then:

$$\begin{aligned} g_{i,t+1}(\mathbf{b}) &= F_i^{t+1}\left(\text{enc}(\mathbf{b})\right) \\ &= f(F_{i-1}^t(\text{enc}(\mathbf{b})), F_i^t(\text{enc}(\mathbf{b})), F_{i+1}^t(\text{enc}(\mathbf{b}))) \\ &= f\left(g_{i-1,t}(\mathbf{b}), g_{i,t}(\mathbf{b}), g_{i+1,t}(\mathbf{b})\right). \end{aligned} \tag{3}$$

where the indices $i-1, i+1$ are computed modulo n. □

Fixing notation from Observation 1, each canonical computation $g_{i,t}$ induces a partiton of the input set $\{0,1\}^k$ into sets $P_{i,t}^0 = \{\mathbf{b} \,|\, g_{i,t}(\mathbf{b}) = 0\}$ and $P_{i,t}^1 = \{\mathbf{b} \,|\, g_{i,t}(\mathbf{b}) = 1\}$.

Then, we can reformulate Observation 1 in terms of the partitions and show that the partitions at time-step $t+1$ can be computed explicitly as:

$$\begin{aligned} P_{i,t+1}^0 &= \bigcup_{a,b,c\in\{0,1\}, f(a,b,c)=0} \left(P_{i-1,t}^a \cap P_{i,t}^b \cap P_{i+1,t}^c\right) \\ P_{i,t+1}^1 &= \bigcup_{a,b,c\in\{0,1\}, f(a,b,c)=1} \left(P_{i-1,t}^a \cap P_{i,t}^b \cap P_{i+1,t}^c\right) \end{aligned} \tag{4}$$

Where again, indices $i-1, i+1$ are computed modulo n.

Using either the functional notation in (3) or the partition notation in (4) allows us to compute the canonical computations at time-step $t+1$ just from the ones at time-step t without having to reconstruct the full space-time diagrams of the CA.

Temporal Encoding A common scheme of encoding input bits into CA dynamics when using them as reservoirs is to incorporate the input, bit by bit, with the CA configuration at regular time intervals.

For an ECA with global rule F operating on a cyclic grid of size n; the temporal encoding can be viewed as a mapping $\text{enc} : \{0,1\} \times \{0,1\}^n \to \{0,1\}^n$. Given a sequence of input bits $b_1, \ldots, b_k$ and a time interval $T \in \mathbb{N}$ at which we "inject" the input, we obtain the following CA "trajectory":

$$
\begin{aligned}
&c_1 \coloneqq \text{enc}(b_1, 0^n), \qquad F(c_1),\ F^2(c_1),\ \ldots, F^T(c_1),\\
&c_2 \coloneqq \text{enc}(b_2, F^T(c_1)), F(c_2),\ F^2(c_2),\ \ldots, F^T(c_2),\\
&c_3 \coloneqq \text{enc}(b_3, F^T(c_2)), F(c_3),\ \ldots\\
&\vdots
\end{aligned}
$$

In such a case, we can analogously define the canonical computations for each position and time-step of the trajectory, which is just a function of the input bit sequence. We would also obtain analogous relationships between the canonical computations at time step t and $t+1$; the only exceptions being the time-steps at which a new input bit is "injected" and the canonical computations have to be updated accordingly.

Visualization Algorithm

A simple brute force algorithm can be made to visualize the different partitions produced by a ReCA system and thus the information flow within space-time diagrams. We can in turn use this visualization to identify differences between the possible ReCA parameters.

We first compute all space-time diagrams for a given input set. Stacking the space-time diagrams as shown in Figure 3 produces a binary string for each space-time diagram cell. This can be seen in the figure as the blocks highlighted in blue or red. Since each string corresponds to a specific binary partition we can then label it accordingly. Populating a "canonical computation diagram" (CC diagram) i.e a 2D matrix of the same shape as a related space-time diagram with these labels, then allows us to visualize the information flow within the diagrams as seen from the perspective of the decoder. A python implementation can be downloaded from github:
https://github.com/DeepCANFR/Canonical-Computations.git

Canonical Computation Diagrams

A collection of CC diagrams for all ECA rules, using multiple sizes and input sets can be found at:
https://osf.io/dxgys/?view_only=67c27e321f164ec88cad31d2f444d343

Here we present two examples of input set-encoder-CA combinations.

Example 1:
Input set = $\{\mathbf{0},\mathbf{1}\}^{\mathbf{2}}$
Encoder = $\mathbf{enc} : \{\mathbf{0},\mathbf{1}\}^{\mathbf{2}} \to \{\mathbf{0},\mathbf{1}\}^{\mathbf{n}},\ \mathbf{2} < \mathbf{n}$.

In this example we set up a CA on a cyclic grid of size n as our reservoir substrate and encode binary vectors of size 2 to two locations in the CA's initial configuration (indicated in yellow in Figure 3). We tested variations of encoding distance, i.e the distance between the two transplanted input features in the initial configuration, and grid size. We also produce the full set of possible Boolean functions and assign each a numeric label from 1 to 16. This means that the colors in the CC diagrams corresponds to the same Boolean functions across different rules, grid sizes and encoder settings.

The CC diagram for rule 45 and color labels corresponding to each Boolean function is shown in Figure 4. This allows us to see how the available information changes over time and space as the dynamics of the CA evolves. We can also see that the system has linearlized problems like XOR as the function corresponding to it can be found in the diagram, highlighted in yellow. Additional diagrams for Rules 110, 70, 30 and 45 are shown in Figure 5

Example 2:
Input set = $\{\mathbf{0},\mathbf{1}\}^{\mathbf{k}}$
Encoder = $\mathbf{enc} : \{\mathbf{0},\mathbf{1}\}^{\mathbf{k}} \to \{\mathbf{0},\mathbf{1}\}^{\mathbf{n}},\ \mathbf{k} = \mathbf{n}$.

We produce CC diagrams for the set of binary vectors of sizes k with k = n on a cyclic grid. Since the number of possible functions in this case is extremely large we label them according to the order they are found as we iterate over the CC diagram with the direction of time. Example diagrams can be seen in Figure 6 for rules 110, 70, 30 and 45 at size 10.

Analysing Information Flow for ReCA

Number of Canonical Computations

The maximum number of CCs possible for a given ECA is equal to the number of cells in the space-time diagram. However, even if certain ECA rules rarely reach fixed point attractors or limit cycles they may reach a limit to the creation of novel CCs. If one provides the full space-time diagram to the decoder, there is at this point no reason to continue computation over additional time-steps as this would not provide any new information and we can thus save computational resources.

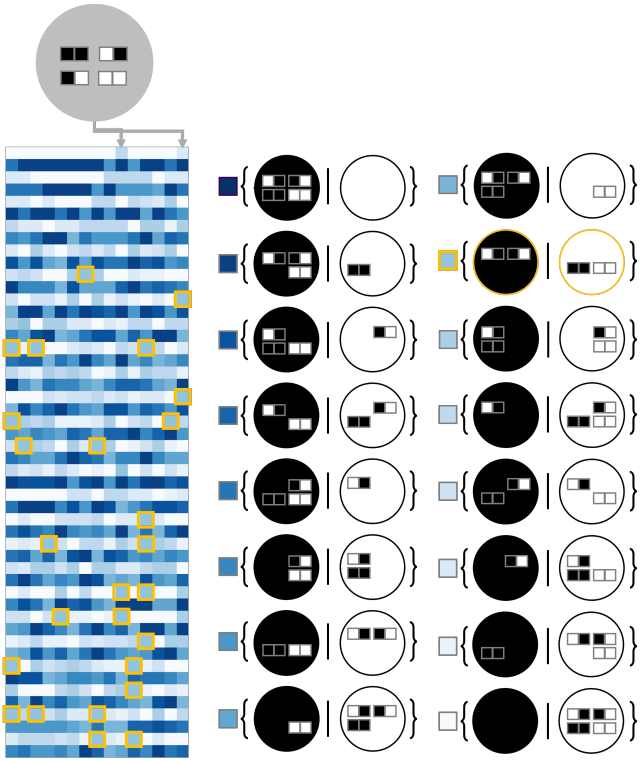

Figure 4: Canonical computation diagram for rule 45, CA size = 15, encoder distance = 5, with labels corresponding to each possible boolean function on the input set $\{0,1\}^2$. The input features were encoded into the two non-white cells in the first row of the CC-diagram. In relation to Figure 3 the boolean functions in the label list are displayed in the same manner as in the bottom of Figure 3. The XOR function is highlighted in yellow throughout the diagram and in the label list.

It should here be noted that the nr of possible CCs grows extremely fast with the size of the input set as it equal to the size of the powerset of the input set.

If one uses temporal input in encoder-CA setups that reach a maximum number of CCs, we may wish to wait until the CC limit is reached before injecting the next input. This way we allow the system to reach its richest computational space. The method presented here may thus be used to identify the optimal waiting period between input injections.

It is however not obvious that maximizing the number of CCs is optimal for a ReCA system as this will increase the computational cost of finding the optimal encoder weights.

Temporal Stability

If only a subset of the space-time diagram is used as input to the decoder at a time (illustrated in orange in Figure 2), as is the case in Glover et al. (2021), we may find that the availability of CCs at a given time-point becomes problematic. For CA rules that continuously generate novel CCs it may be difficult for the decoder to produce any meaningful output as the information the static weights capture continuously changes. Furthermore, the larger the space of available CCs, the less probable it is that the decoder finds CCs relevant to solving a given task if they exists.

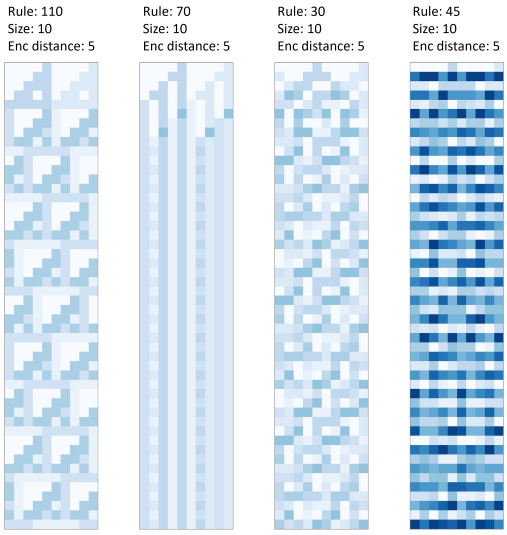

Figure 5: CC diagrams with input set = $\{0,1\}^2$, CA size = 10, encoder distance = 5 for rules 110, 70, 30 and 45. Note that at this encoder distance and CA size the distance between the two input locations are identical on either side of the cyclic grid.

This is illustrated in Figure 6 where we we can see how rule 45 continuously generates novel CCs within the visualized temporal extent of the CC diagram, while the other rules reach cycles.

Spatial Stability

A similar problem to temporal stability arises from the stability of the spatial locations of information in the CC diagram. Even if the same information is available to the readout over a given time interval, the spatial location of the information may change. This again produces an issue in interaction with the spatially static readout weights.

We can see this illustrated in the CC diagrams for rule 110, 70, 30 and 45 in Figure 5. For the binary vectors of size 2 input set and encoder distance of 4, rule 70 quickly produces multiple stable CCs that exist perpetually for sizes 10-30. Rule 110 at size 10, with encoder distance = 4, produces the same CC configuration in cycles of length 25 after time point 5. Thus, the weights of the decoder will capture the same information within this cycle, but not between the configurations of the cycle. This could be remedied by using step sizes for the readout window equal to the cycle length. Rule 45 with the same encoder and size settings does produce cycles of length 1290, but the cycle length rapidly

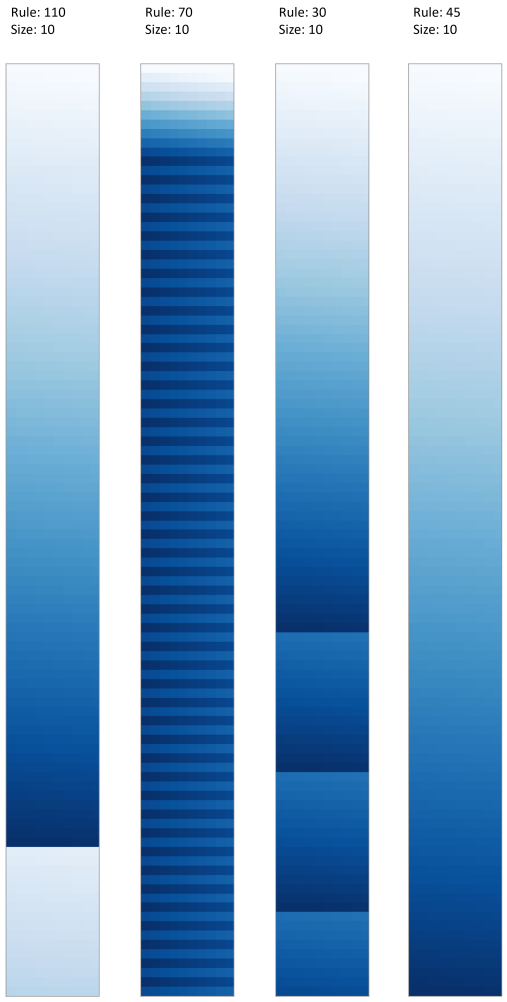

Figure 6: CC diagrams with input set = $\{0,1\}^{10}$ and CA size = 10 for rules 110, 70, 30 and 45. Note that the because the input size and CA are equal in this case we do not give an encoder distance.

increases with grid size. Thus, while rule 45 produces the largest number of CCs of the three rules it is likely not suited well for temporal readout methods due to the lack of spatial stability of information within the diagram.

Notably, the time to a CC cycle (time until the configurations of computations in the CC diagram repeat) may be larger than the time to a CA cycle (time until the configurations of the space-time diagram repeat). For rule 45 at size 10 there are $2^{10} = 1024$ possible configurations, meaning that the maximal cycle length possible at this size is 1024. However, for rule 45 at this size, with the input set of all binary vectors of size 10, the CC cycle we find is of length 2580. This is possible because each CC diagram is created by combining multiple regular space-time diagrams. The cycle length of the CC diagram is thus just the least common multiple of the cycle lengths of each of the regular diagrams.

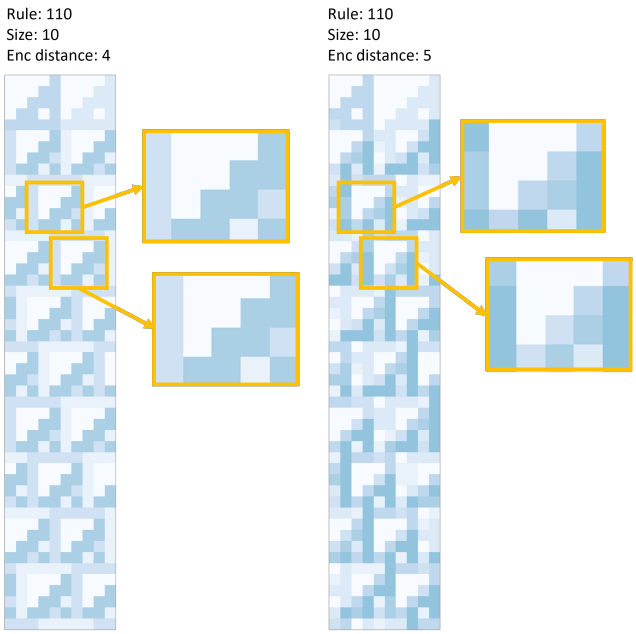

Figure 7: CC diagrams with input set = $\{0,1\}^2$, CA size = 10, encoder distance = 4 and 5 for rule 110. The CC configurations are identical but spatially shifted in the encoder distance = 4 case, while different in the encoder distance = 5 case. The difference between convolutions applied to the CC diagrams are highligted in yellow

Shift Cycles

For rule 110, at size 10 with input set = $\{0,1\}^2$ and encoder distance = 4, we also observe that the same information appears in the same configurations but spatially shifted at relatively short cycles, which we name shift cycles. For rules displaying this type of behaviour we can capture relevant information more efficiently than described for standard cycles by using convolutional readouts as these will capture constellations of information independent of their spatial location within the readout window. However, increasing the decoder distance to 5 increases the cycle length to 75 even though the appearance of the CC diagram is relatively similar. In Figure 7 we highlight these difference by extracting convolutions from the two CC diagrams. The two convolutions are identical but spatially shifted for encoder distance of 4, while they differ slightly with encoder distance 5 which can be seen in the leftmost column of the convolution.

Encoding Distance

With the encoder set to use different distances between the encoded input feature locations, we can see how padding the input feature can be detrimental to certain rules. If the rule in question maps the 010, 100 and 001 neighbourhood configurations to 0 the information contained in the input is always

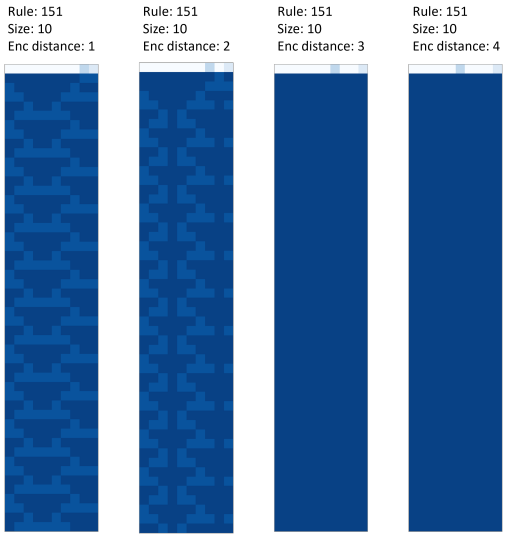

Figure 8: CC diagrams with input set = $\{0,1\}^2$, size = 10, encoder distance = 0-3 for rule 151. In the cases where the encoded input falls within the range of each others neighbourhoods we see propagation of information over time and space while any other distance leads to an instant collapse to a single CC

destroyed. Alternatively if the rule only sends the 010 neighbourhood unmodified left, right or forward we may similarly see a lack of complex patterns as the features never interact. This can be seen in the CC diagram for rule 151, size = 10 and input set = $\{0,1\}^2$ in Figure 8. Using an encoder distance of 0 or 1 between the two features produces more different CCs as well as longer cycles with information being sustained over time and transmitted over space, while any other distance only produces a collapse of computations.

CA Size

Varying the size of the CA is known to affect ReCA performance (Glover et al., 2021). Fixing the encoder distance to 4 for rule 18 and input set = $\{0,1\}^2$, we can see this effect. In Figure 9 we show CC diagrams for sizes 10, 15 and 20. At size 10 the CA rapidly reaches a cycle of length 6 with no shifts, while at size 15 we can see a quick collapse of computation after only 8 time steps. At size 20 we get very long cycles of length 372. Interestingly, at size 10 we compute one more CC than at sizes 15 and 20.

Conclusion

The CC diagrams reveal the exact identity and location of the information available within the space-time diagrams of a given input-set and encoder-CA setup. This provides us with a novel view into the computational dynamics of these systems that may be useful for the general analysis of information flow in CA. Our main contribution is thus introducing the novel notion of the CC diagrams themselves.

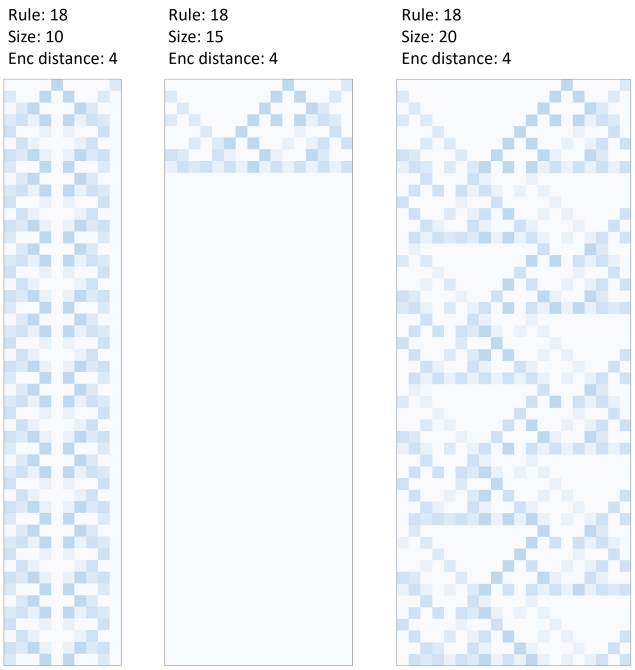

Figure 9: CC diagrams with input set = $\{0,1\}^2$, size = 10, 15 and 20, encoder distance = 4 for rule 18. The effect of changing the size of the CA is seen in the flow of CCs throughout the diagram with rapid cycling in the size = 10 case, early collapse to a single CC in the size = 15 case and long cycles at size 20.

We also highlight how insights gained from canonical computation diagrams may be useful for the specific use case of ReCA design by guiding selection of encoder, decoder and CA parameters to increase a ReCA system's performance. Testing of these predictions would however be necessary to verify the claims proposed in this paper, this will be part of our extended work on the topic.

Acknowledgements

This work was partially financed by the Research Council of Norway's DeepCA project, grant agreement 286558. Additionally, this work was supported by Grant Schemes at CU, reg. no. CZ.02.2.69/0.0/0.0/19_073/0016935, and is part of the RICAIP project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 857306. We would also like to thank Tom Glover for useful and inspiring feedback and discussions.

References

Aaser, P., Knudsen, M., Ramstad, O. H., van de Wijdeven, R., Nichele, S., Sandvig, I., Tufte, G., Stefan Bauer, U., Halaas, Ø., Hendseth, S., et al. (2017). Towards making a cyborg: A closed-loop reservoir-neuro system. In *ECAL 2017, the Fourteenth European Conference on Artificial Life*, pages 430–437. MIT Press.

Banks, E. R. (1971). Information processing and transmission in cellular automata.

Biehl, M. and Witkowski, O. (2021). Investigating transformational complexity: Counting functions a region induces on another in elementary cellular automata. *Complexity*, 2021:1–8.

Codd, E. and Ashenhurst, R. (1968). Cellular automata, academic press. *Inc. New York and London*.

Cook, M. et al. (2004). Universality in elementary cellular automata. *Complex systems*, 15(1):1–40.

Dittrich, P. (2018). Towards measuring the semantic capacity of a physical medium demonstrated with elementary cellular automata. *BioSystems*, 164:177–185.

Feldman, D. P., McTague, C. S., and Crutchfield, J. P. (2008). The organization of intrinsic computation: Complexity-entropy diagrams and the diversity of natural information processing. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 18(4):043106.

Fernando, C. and Sojakka, S. (2003). Pattern recognition in a bucket. In *Advances in Artificial Life: 7th European Conference, ECAL 2003, Dortmund, Germany, September 14-17, 2003. Proceedings 7*, pages 588–597. Springer.

Glover, T. E., Lind, P., Yazidi, A., Osipov, E., and Nichele, S. (2021). The dynamical landscape of reservoir computing with elementary cellular automata. In *ALIFE 2022: The 2022 Conference on Artificial Life*. MIT Press.

Hudcová, B. and Mikolov, T. (2020). Classification of complex systems based on transients. *arXiv preprint arXiv:2008.13503*.

Hudcová, B. and Mikolov, T. (2021). Computational hierarchy of elementary cellular automata. *arXiv preprint arXiv:2108.00415*.

Israeli, N. and Goldenfeld, N. (2006). Coarse-graining of cellular automata, emergence, and the predictability of complex systems. *Physical Review E*, 73(2):026203.

Jaeger, H. (2001). The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148(34):13.

Kleyko, D., Khan, S., Osipov, E., and Yong, S.-P. (2017). Modality classification of medical images with distributed representations based on cellular automata reservoir computing. In *2017 IEEE 14th International Symposium on Biomedical Imaging (ISBI 2017)*, pages 1053–1056. IEEE.

Langton, C. G. (1990). Computation at the edge of chaos: Phase transitions and emergent computation. *Physica D: nonlinear phenomena*, 42(1-3):12–37.

Lizier, J. T., Prokopenko, M., and Zomaya, A. Y. (2012). Local measures of information storage in complex distributed computation. *Information Sciences*, 208:39–54.

Maass, W., Natschläger, T., and Markram, H. (2002). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560.

McDonald, N. (2017). Reservoir computing & extreme learning machines using pairs of cellular automata rules. In *2017 international joint conference on neural networks (ijcnn)*, pages 2429–2436. IEEE.

Mitchell, M. et al. (2005). Computation in cellular automata: A selected review. *Non-standard computation*, pages 95–140.

Morán, A., Frasser, C. F., and Rosselló, J. L. (2018). Reservoir computing hardware with cellular automata. *arXiv preprint arXiv:1806.04932*.

Nichele, S. and Gundersen, M. S. (2017). Reservoir computing using non-uniform binary cellular automata. *arXiv preprint arXiv:1702.03812*.

Nichele, S. and Molund, A. (2017). Deep reservoir computing using cellular automata. *arXiv preprint arXiv:1703.02806*.

Nikolić, D., Haeusler, S., Singer, W., and Maass, W. (2006). Temporal dynamics of information content carried by neurons in the primary visual cortex. *Advances in neural information processing systems*, 19.

Wolfram, S. (1983). Statistical mechanics of cellular automata. *Reviews of modern physics*, 55(3):601.

Wolfram, S. (1984). Universality and complexity in cellular automata. *Physica D: Nonlinear Phenomena*, 10(1-2):1–35.

Wuensche, A., Lesser, M., and Lesser, M. J. (1992). *Global dynamics of cellular automata: an atlas of basin of attraction fields of one-dimensional cellular automata*, volume 1. Andrew Wuensche.

Yilmaz, O. (2014). Reservoir computing using cellular automata. *arXiv preprint arXiv:1410.0162*.

Zenil, H. (2009). Compression-based investigation of the dynamical properties of cellular automata and other systems. *arXiv preprint arXiv:0910.4042*.