Autopoiesis meets mechanistic computation: A proof of concept of computational post-cognitivism

Stefan Riegl (science@stefanriegl.com)
Radboud University Nijmegen
6525GD Nijmegen, Netherlands

Serge Thill (serge.thill@donders.ru.nl)
Donders Institute for Brain, Cognition, and Behaviour
Radboud University Nijmegen
6525GD Nijmegen, Netherlands

Abstract

Recent research suggests that post-cognitivist and computationalist paradigms are not necessarily incompatible. Here, we provide further support in favour of this proposition. Specifically, we demonstrate that it is possible to provide an implementation of two relevant verbal theories, Autopoietic Theory and Mechanistic Computation, that can analyse the AND-gate in Game of Life from the point of view of an autopoietic observer, identifying unities that show the property of either autopoiesis, mechanistic computation, or both. The explicit implementation also highlights the kind of considerations that a formalisation of a computational post-cognitivist theory has to address, which are not necessarily apparent from verbal theories alone.

Keywords: Autopoietic Theory; Mechanistic Computation; Computational Post-Cognitivism

Introduction

Cognitivism usually assumes a form of Computationalism (*Co*), treating cognition as a kind of information processing. Cognitivist paradigms are rejected by various flavours of Post-Cognitivism (*PC*), in particular because they reject the representationalism these entail. Autopoietic Theory (*AT*) (Varela, 1979) is, for example, a prominent and theory of *PC*.

For a long time, it was assumed that computation requires representation, which meant that *PC* naturally rejected computationalism as well. However, recent advances in neural network research showed that computation is possible without explicit representation[1] (Piccinini, 2004, 2008), which led to the proposal that *PC* does not necessarily imply anti-computationalism (Villalobos & Dewhurst, 2017). The proposal, however, remains debated, with most of the discussions philosophical in nature (Casper & Artese, 2020).

In this paper, we provide initial empirical support that such a theory of Comutational Post-Cognitivism (*CoPC*) is feasible. Specifically, we demonstrate that it is indeed possible, as postulated by Villalobos and Dewhurst (2017), for an observer to interpret a system as being both computationalist and autopoietic. We achieve this using a simulation built on Game of Life (*GoL*) as virtual environment, as it has previously been shown that autopoietic properties can be identified in *GoL* (Beer, 2015). We employ Mechanistic Computation (*MC*), a modern theory of computation that does not imply representation, as suggested by Villalobos and Dewhurst.

To provide a strong demonstration in line with underlying theory, we first formalised the verbal theories *AT* and *MC* into mathematical descriptions (Riegl, 2022). Development of the implementation (and running the corresponding simulations) interleaved with updates of the formalisations, which were checked against the verbal theories[2]. Here, we focus on the simulation part of the development process as demonstration, in particular to provide a demonstration that it is indeed possible to implement both verbal theories without contradiction. This serves as a proof of concept that a formalisation of *CoPC* is feasible and paves the way for further refinements of the initial formalisation underlying this implementation.

Background

Autopoietic theory

AT is an ear early post-cognitivist theory (Maturana & Varela, 1980). It proposes a functional view of what processes underlie the phenomenological organism and how those processes interact (*autopoiesis* meaning "self-producing"). Essentially, an organism must actively preserve the processes that make up this organism such that it (and said processes) can persist continuously. The definition of autopoiesis evolved over time, but can be roughly summarized as follows (Varela, 1979, p. 13): An autopoietic system is a network of processes that transform components into components, which satisfies two properties: (1) Through the processes, the components continuously regenerate and realize the network that produced them. (2) The components constitute the autopoietic system as a concrete unity in space.

Here, a process can be understood as a course of action extended over time that turns a set of interacting components into a (possibly empty) set of components. Linked by components, processes can be seen as forming a chain in the simple case, or generally a network. Processes can be compared by the relations between components they maintain or change. When a network of processes maintains the same relations of processes over time, the network is an autopoietic system.

Mechanistic computation

Established accounts of computation (such as Turing machines, algorithms, and so on) have to deal with the "prob-

[1] For a more detailed discussion on the historical context of the complete argument, see (Villalobos & Dewhurst, 2017).

[2] See also (Guest & Martin, 2021) for reasoning behind this approach to computational modelling.

In L. K. Samuelson, S. L. Frank, M. Toneva, A. Mackey, & E. Hazeltine (Eds.), *Proceedings of the 46th Annual Conference of the Cognitive Science Society.*

lem of computational implementation" (Piccinini, 2015): If a specific instance of computation is defined in an abstract formalism, how can we tell whether a physical system is implementing that computation? To solve this, *MC* (Piccinini, 2015) proposes a mechanistic framework as well as criteria for computationally relevant mechanistic properties based on logic and computability theory. Computing systems are thus seen as a kind of functional mechanism implemented physically. In other words, a physical computing system is "a mechanism whose teleological function is computing a mathematical function f from inputs I (and possibly internal states S) to outputs O" (Piccinini, 2015, p. 121). Importantly, computation in this sense does not require representations and does not invoke "the notion of syntax (or formal property)".

Game of life

GoL was conceived by John Horton Conway and is in its original form a cellular automaton on a two-dimensional grid that evolves over discrete time, sometimes also described as zero-player game (Berlekamp, Conway, & Guy, 2004). Despite having very few and simple rules to determine the automaton's next state, the cellular automaton allows very complex behaviour to emerge, which motivated a lot of research.

Each cell of a typical *GoL* cellular automaton is in one of two states, traditionally called alive and dead state. A cell can change its state from one time step to the other, depending on its Moore-neighbourhood (*i.e.* the eight other cells touching by edge or corner). The state of a cell in the next time step is determined by the rules: a dead cell comes alive *iff* it has exactly three alive surrounding cells while a living cell remains alive *iff* it has either two or three living neighbours. *GoL* features a number of frequently emerging patterns, even from random initialization of the grid. Researchers and enthusiasts created a thorough classification of many of those patterns in an openly accessible online resource[3] over many years.

In the past, many simulation environments were specifically developed to study *AT*. In contrast, *GoL* happened to provide suitable abstractions and enough complexity such that autopoietic concepts can be applied to recurring patterns of *GoL* like the glider (Beer, 2004, 2014, 2015, 2018). For example, the pattern known as glider can be described as an autopoietic system, where cells are components and the application of a *GoL* rule forms a process.

Methods

GoL simulation

The demonstration described below is implemented in *Python*, using *Golly*[4] as *GoL* simulator. Visualisations were created using *Cytoscape*[5]. The simulation environment consists of a prepared world of 56×63 cells (figure 1), containing an AND-gate typical for *GoL*. While it has been shown that cellular automata can perform binary computations and that

[3]LifeWiki, https://conwaylife.com/wiki/
[4]Golly, http://golly.sourceforge.net/
[5]Cytoscape, https://cytoscape.org/

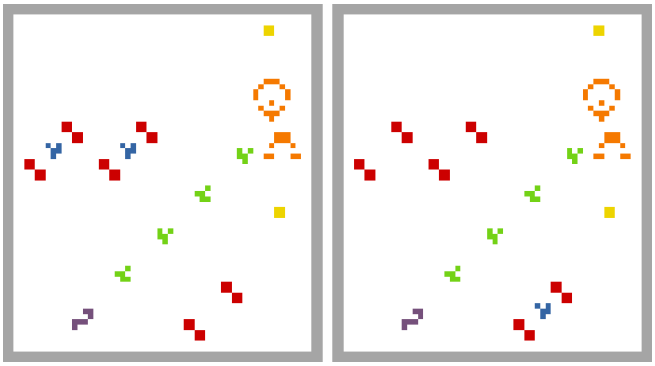


Figure 1: The AND-gate used in this work. Red blocks serve as spatial reference markers and have no role in the model itself. A Gosper glider gun including two blocks (orange, yellow) produces a stream of gliders (green) moving towards an eater (purple) that consumes them. "Input" gliders (blue) can start at either or both locations indicated by red blocks on the left side. If one of these gliders passes through the red blocks at the bottom, the output of the gate is TRUE, else it is FALSE When only one input glider is used, it will collide with the stream and disintegrate. With two gliders (left figure), the first glider crashes into a glider in the stream and both disintegrate without leaving "debris". The second glider can now pass through the hole in the stream of gliders and reach the output location (right figure). Note that this assumes appropriate timing of the input gliders.

GoL is Turing-complete (Berlekamp et al., 2004), the objective here instead is to discuss how elements of *AT* and *MC* can be identified based on the respective theoretical foundations.

Identifying necessary components of autopoietic systems and computational mechanisms

Observer In *AT*, "everything said is said by an observer" (Varela, Maturana, & Uribe, 1974), which is the starting point for further investigation of the world, including autopoiesis. However, if we assume an observer to be realised as a living organism, it is necessarily an autopoietic system itself (Maturana & Varela, 1980). This introduces an explanatory circularity that requires us to take a step back and look at the observer-environment-system as a whole with the aid of the concepts below. For the present purposes, the implementation and *GoL* simulation is evaluated from the point of view of such an observer. However, for simplicity the observer here is assumed to literally only observe, i.e. it does not interact actively with the *GoL* world and is not part of the simulation.

Unities The state of the environment in the *GoL* simulation is analysed after each time step. Unities are initially identified as single cells alive at one moment in time (a property spatially bounded to that cell). Later, unities can be composed of other unities and extend over time.

On that basis, the link between two surrounding alive cells can be identified as a relation. Linked cells (i.e. a set of unities

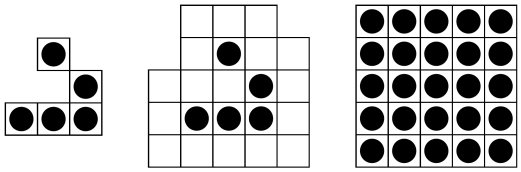


Figure 2: The alive cells of a glider without (left) and with (centre) a boundary of dead cells around them. Only five alive cells without a boundary in some arbitrary background (right) are not enough to successfully distinguish it as complex unity.

and a set of relations) then form a structure. This structure, by extension, is spatially bounded again and, if it can be separated from a background by the property of being linked alive cells with a boundary, it can be understood as complex unity (as illustrated in figure 2). This procedure is repeated to distinguish more and more complex unities such as the glider, and continues until no further unities can be found.

Processes A process is a transformation of a set of components[6] at one moment in time into a set of components at a later moment. To identify a process, we recall two suitable moments in time and consider the set of components from each. What suitable means is not trivial to define, but generally it holds that if two moments are close enough to each other, the spatial distance of the sets of components cannot be very large. For a pair of sets of consecutive moments, the joined subspace of all components from the first set must overlap with the joined subspace of all components from the second set. A process is then defined on the two sets of components at different moments in time, where the first set is transformed to the second set. In particular, processes that transform into less or no components can be considered destructive processes.

Autopoietic systems Given the definition of autopoiesis, the subordinate goal of distinguishing autopoietic systems requires identifying cyclical patterns, i.e. sub-networks of processes (sub-graphs) that repeat after a certain period. This opens up a number of questions:

(A) How to handle sub-networks that start and end with two or more processes that do not share components? First, the identified processes are understood as a linked network of processes. This network can be modelled as a directed graph, where each process is a node. Two nodes have an edge, if the corresponding processes are such that the set of "outgoing" components of one process and the set of "incoming" components of the other process overlap (i.e. they share at least one component). This operation corresponds to the "concatenation" (Maturana & Varela, 1980) of processes into a network, which can be repeated recursively.

(B) How to treat patterns (like the glider), if spawned from a larger, repeating pattern (like the glider gun)? Each process in the network transforms components from one moment in time to the one following afterwards. This means that processes in the network can be ordered (keeping their links to other processes). Over the ordered processes we can slide a time window of some size. The window size corresponds to the length of the cycle (of processes) to be detected. Then, for a given time window, only the processes within the time window are considered further and split into sub-networks that are graph-islands, where any two nodes within a graph-island are connected, and any two nodes between two graph-islands are not connected. For detection of different cycle lengths the procedure is repeated. Starting with smaller window sizes and removing processes involved in detected cycles from further consideration allows to only identify smaller unities first.

(C) Can two processes at different times be equal if they happen at different locations (e.g. glider movement after 4 steps)? We can call set of processes of a sub-network, where all processes of the set start at a given moment in time, a "segment". Two segments are considered equal, if the same number and type of processes happen, and the spatial arrangement of components for all those processes is the same for "incoming" and "outgoing" components respectively.

If the first and the last segment can be considered equal, a candidate for a cycle is found. After a cycle has been found, all current segments are remembered and the current time window can be slid stepwise further to see how far the cyclical system extends into the future. The analysis above depends on two assumptions: (1) that no interaction with the environment (perturbations) happened between the first and the last segment; and (2) that the environment is deterministic. Assumption (1) holds because one would, in general, also observe the interaction with the environment, including the chain of processes that lead up to the interaction. Those processes would then also appear in the sub-network from which the segments are derived, and a cycle could not be found. Assumption (2) is trivially true in the *GoL* setting, hence we can expect the sub-network to continue evolving at the end of the time window just as it evolved at the start of the time window.

Determining the temporal extent of the cyclical system distinguishes it as a unity that extends over time. This unity consists in a network of processes that regenerates itself and has a clear boundary. This satisfies the definition of autopoiesis and hence the unity can be identified as autopoietic system (Beer, 2015).

Computational mechanisms If the argument of Villalobos and Dewhurst (2017) pans out, we should be able to make use of the properties identified so far to identify computational mechanisms. Below, we illustrate the implementation using the AND-gate above as a working example for simplicity, but the algorithm itself is not limited to such simple cases and does not depend on *GoL*-specific properties.

We first define an *episode* as a set of linked processes (i.e. a network) that was observed within a certain subspace and a given time window. Figure 3 demonstrates this on a more abstract level. To discover a computational mechanism, the

[6]For more clarity, a unity is referred to as a component, when it is part of a larger (complex) unity or appears in the scope of a process.

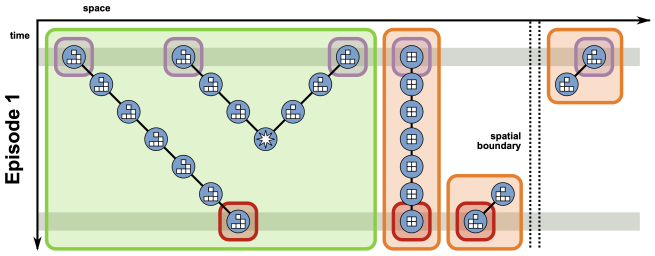


Figure 3: An example of an incomplete detection of the computational process of a logical AND based on one episode of observed processes. Shown is a two-dimensional coordinate system, with time on one axis and (simplified) space on the other. Components are denoted by blue circles with pictograms, connected by a line where there is a corresponding process. The small pictograms in circles represent the unities at a moment in time independently of their (possibly varying) concrete structures, such as the glider, the block, or the event of a clash of two gliders and subsequent annihilation. Glider-unities move through time and space, while block-unities only move through time. Some gliders enter or leave the fixed subspace of observation, which is denoted by the left or right spatial boundary. The horizontal grey bars denote the first and last moment of observation.

defining elements of a mechanism need to be worked out, including input elements, output elements, supporting "scaffold" of the mechanism, and remaining observed elements that do not matter for the mechanism. For this, consider the highlighted areas in figure 3, which display the assignment of certain roles to unities. Green and orange are normative sets of components: The green set of components should be included in the mechanism, while the orange set should not. With only one episode observed, it is possible to identify potential *input* and *scaffold* (both currently violet) elements and *output* (red) elements. It is also possible to understand the sub-network that only has output components without also having input or scaffold components (see bottom-right in the diagram) as irrelevant. In other words, if there is an effect without known cause, we cannot make an inference. However, some problems remain: the connection between all the sub-networks of relevant components is not clear, it is not obvious how to distinguish relevant components from the irrelevant, and there is not enough information to infer a function like the logical AND.

To mitigate this situation, we observe more than one episode and then look out for what different input condition may result in different output condition. With more than one episode it becomes possible to compare components across episodes in terms of e.g. identified inputs and outputs (see figure 4). Distinctions can be made between input components (violet), which may change across episodes, and scaffold components (blue), which are the same across all episodes and enable the computational mechanism. It is also possible to separate relevant (green) from irrelevant (orange) components by removing all sub-networks that do not have any non-scaffolding input components, possibly linked across episodes.

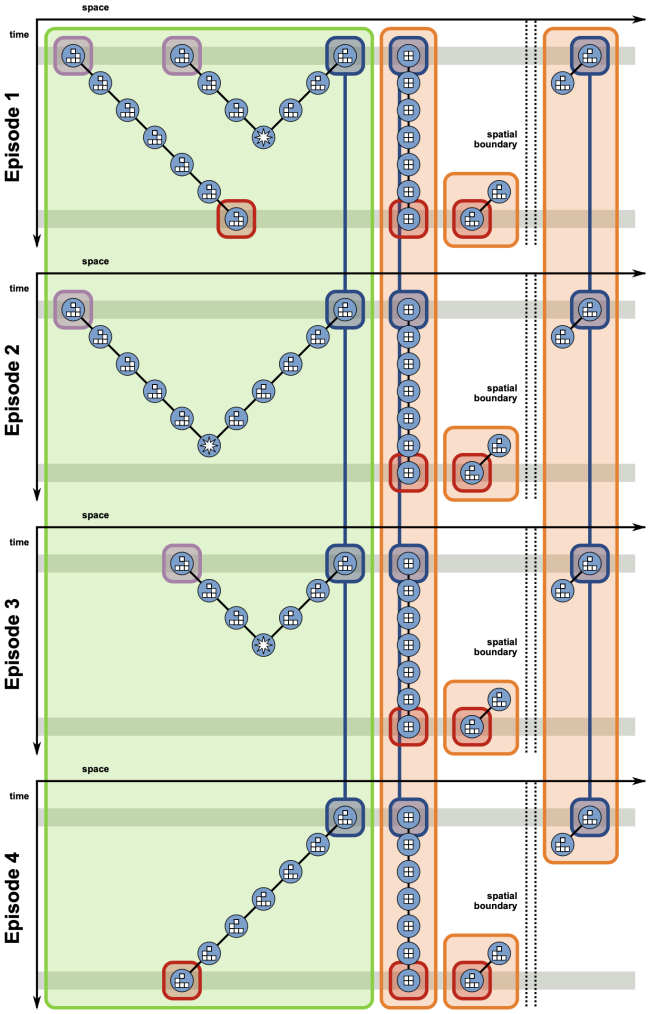


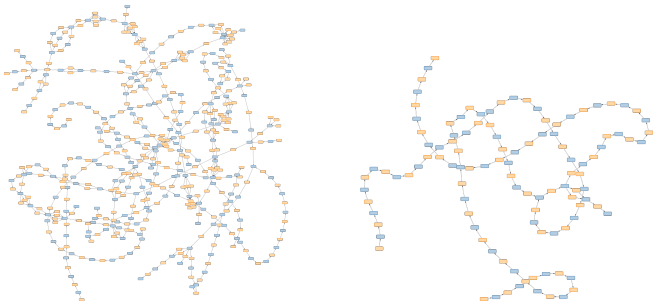
Figure 4: An example of a successful detection of the computational process of a logical AND based on four episodes.

Using the same kind of reasoning as before it is now possible to identify two non-scaffolding input components. This is already possible with one more episode, but the formally complete domain of the logical AND-function is only captured with all four episodes, based on an interpretation that maps the states glider present/not-present to TRUE/FALSE. While each episode features an instance of a computational mechanism, the interpretation allows mapping the total of all episodes to an extensively defined mathematical function.

Simulation results

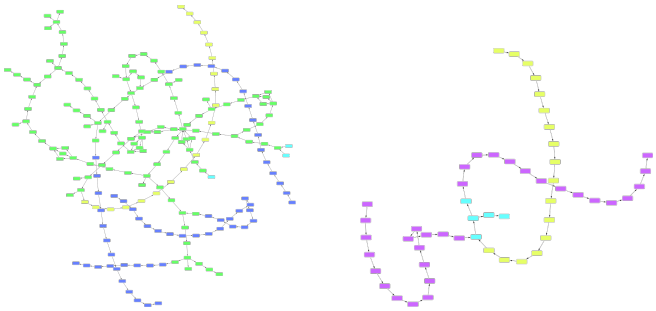
Components and processes in the AND-gate

To demonstrate detection of components and processes in the sense of *AT*, the *GoL* world was let run and the model observer "observed", i.e. analyses were made for each *GoL* step and related with each other over time. The simulation was



(a) The glider gun. (b) The eater.

Figure 5: Networks of components (orange) and processes (blue) for a limited time window of observation.



(a) The glider gun. (b) The eater.

Figure 6: Simplified networks of only processes, coloured based on their distinguished unities over time: static blocks (blue), South-West-flying glider (yellow), and eater (purple), the glider gun (green), and undistinguished processes (cyan).

run for 35 steps after the initial condition, allowing to contain a bit more than one full cycle of the glider gun. The network of processes and components for the glider gun producing a new glider and the eater while absorbing a glider, as identified by the implementation described above, can be seen in figure 5. Similar graphs can be produced, e.g. for static blocks or South-West-flying gliders, but as they show no interaction with other unities they are less interesting.

Each network of components and processes can be abstracted into a network of only processes, with an edge only between processes connected by a component (Beer, 2015). Figure 6 shows the process-networks for the glider gun and the eater. Nodes of the same colour were identified as belonging unambiguously to the same trajectory of a unity that extends over time. Here, the process-networks for the glider gun can be seen to be more complex. First, it contains two blocks (represented by blue nodes) that are so-called *still lives* and remain passive unless perturbed by other unities. As the glider gun cycles through its structures, the blocks interact for a few time steps with the other part of the glider gun until they return to their previous form. Also clearly visible is the glider (yellow nodes) that emerges somewhere in the middle of the simulation. All the other nodes (green) were identified as not belonging to any oscillator of shorter period, but being part of a bigger oscillator i.e. the glider gun.

Meanwhile, the eater, being another still life, can be seen as an oscillator of period 1, and thus the cycle of processes contains only one kind of process. Hence, all the processes belonging to the trajectory of a still life in this graph are the same (unlike the processes that belong to e.g. a glider). The pre-existing eater (purple nodes on the left) and the approaching glider (yellow nodes) merge during the impact (cyan nodes). After a short time of instability (also cyan nodes) the dead-or-alive cells arrive back in a configuration, that forms the structure of an eater pattern, such that the processes of maintaining the eater components are visible after the impact (purple nodes on the right).

The autopoietic unities of the glider gun and the eater are circular in their kind of processes, and the entirety of composing components is spatially bounded (seen by the limited number of edges). They also have phases of interactions with other unities (production or destruction of a glider), but regenerate their bounded network after a limited number of steps.

Computational mechanisms

To demonstrate the detection of computational mechanisms, the same *GoL* world was run for 68 steps and observed in the same rectangular subspace over each of the four episodes. The number of steps was chosen based on the time it takes for the more distant glider to pass through the hole in the stream of gliders. For each episode a network of processes was observed. The network of processes for episode 1 can be seen in figure 7. The different parts match in principle the different types of sub-networks as illustrated in figure 4.

The sub-networks that feature light-blue nodes are counted towards the computational mechanism. The sub-networks with orange nodes appear in all episodes and were considered irrelevant to the computational mechanism. They include the trajectories of a double-block over the all observed time steps (with blue scaffold and red output nodes), two gliders at the start of the observation over 9 and 39 steps (with blue scaffold nodes), and two incomplete gliders at the end of the observation over 5 and 35 steps (no input or output nodes).

The sub-networks that were identified as being part of the mechanism mirror the more abstract description presented in figure 4: Episode 1 includes two input, one scaffold, and one output unity. The smaller sub-network with light-blue nodes represents the collision between one input and the scaffold unity, such that the other input unity (glider) can fill the role as output unity in the end. The sub-networks for episodes 2 and 3 would similarly include one input and scaffold unity each and no output unity; and for episode 4 it would include no input and one scaffold unity, and the single output unity is the glider about to crash into the eater. The networks of processes of all four episodes in concert are then matched up as outlined in the previous section. Given that the sub-networks are reproducible, each one reflects one mapping rule for an extensively defined mathematical function.

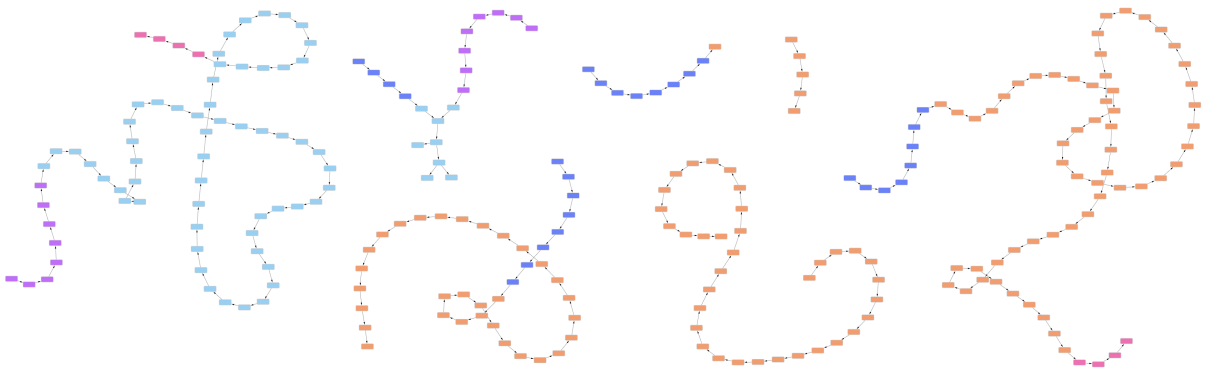


Figure 7: The observed network of processes for episode 1. The colours denote the same component roles in the scope of the algorithm, except for light-blue nodes, which are green in the earlier diagram. The number of required input/scaffold and output nodes to match over episodes was set to 8 and 4, respectively.

Discussion

We demonstrated how in the *GoL* environment an observer can find things that are autopoietic (the glider and others), things that are computational (the AND-gate mechanism), and things that are neither (e.g. the "debris" left after sufficiently many steps of a randomly initialised world). The remaining question is whether there can be things that are both autopoietic and computational.

For any thing (i.e. a network of processes) to be computational it is sufficient to be a computational mechanism, that is it needs to consist of a certain configuration or structure of unities, which however could over time. A computational mechanism can thus be seen as one process or a network of processes. Hence, having a network of processes that is autopoietic and computational is not in principle impossible.

Note that, generally, a computational mechanism also includes other formal properties, such a set of rules. Here we only consider the spatio-temporal components, since they have the potential for contradiction, i.e. falsifying the hypothesis that "there are things that are both". According to Maturana and Varela (1980) any autopoietic system is necessarily operationally closed and structurally coupled, which is a very different approach from defining systems as input-output-machines. However, this does not preclude the possibility for autopoietic organisms to also behave like input-output-machines (Varela, 1979) under certain circumstances.

In his analysis of autopoiesis in *GoL*, Beer (2015) concludes not only the glider to be an autopoietic unity, but also still lives such as the block or the blinker, with which we agree. While it may seem like preservation of components is trivial for simple unities like block or blinker, it is important to remember that the maintenance of the same components depends on few, but active processes that (as a network) satisfy the definition of autopoiesis.

Going beyond these examples, the eater also is a still life, like the block, and likewise autopoietic (Beer, 2015). The glider gun as a whole (including its two blocks) can also be considered autopoietic, since it is (like the glider) a network of processes that is operationally closed (first criterion in the definition of autopoiesis), realised in space, and establishes a boundary to its surroundings (second criterion). As a consequence, the system of glider gun, eater, and the stream of gliders that is connecting both, can be considered one large, linked network of processes (complex unity) that is an autopoietic system, since as a whole it properly satisfies the definition of autopoiesis as outlined earlier (Varela, 1979).

Adding input and output vehicles to that system leads to the computational mechanism of the AND-gate. However, this mechanism is not autopoietic, since the respective network of processes has no means of regenerating the input and output unities. So the answer to the initial question is that only the scaffolding-part that realises the functional mechanism of the AND-gate (i.e. everything but input and output unities) could be understood as a computational, autopoietic system. In another, looser meaning of the word, the actual (computational) mechanism that "does the work" may be autopoietic, without the (computational) vehicles that are being "worked with".

Note however that not all computational systems (minus inputs and outputs) are necessarily autopoietic. In the case of the *GoL*-specific AND-gate, the computational mechanism (seen as process) is autopoietic because it is operationally closed (while remaining to open to interaction), reproduces its original state (including an uninterrupted stream of gliders), and then maintains this state until the next possible instance of computation takes place.

Conclusion

We illustrated a proof of concept implementation of a simple observer in *GoL* able to distinguish both an autopoietic system and a computational mechanism, supporting the idea that computationalist post-cognitivism is indeed possible (Villalobos & Dewhurst, 2017). In particular, the implementation makes explicit details that a formal theory of *CoPC* has to address, which remain implicit in current verbal formulations of *AT* and *MC*. The next step is to elaborate the working formalisation into a rigorous mathematical framework.

References

Beer, R. D. (2004, July). Autopoiesis and Cognition in the Game of Life. *Artificial Life*, *10*(3), 309–326. doi: 10.1162/1064546041255539

Beer, R. D. (2014, April). The Cognitive Domain of a Glider in the Game of Life. *Artificial Life*, *20*(2), 183–206. doi: 10.1162/ARTL_a_00125

Beer, R. D. (2015, February). Characterizing Autopoiesis in the Game of Life. *Artificial Life*, *21*(1), 1–19. doi: 10.1162/ARTL_a_00143

Beer, R. D. (2018, July). On the Origin of Gliders. In *ALIFE 2018: The 2018 Conference on Artificial Life* (pp. 67–74). MIT Press. doi: 10.1162/isal_a_00019

Berlekamp, E. R., Conway, J. H., & Guy, R. K. (2004). *Winning ways for your mathematical plays, volume 4*. AK Peters/CRC Press.

Casper, M.-O., & Artese, G. F. (2020, October). Maintaining coherence in the situated cognition debate: What computationalism cannot offer to a future post-cognitivist science. *Adaptive Behavior*, 105971232096705. doi: 10.1177/1059712320967053

Guest, O., & Martin, A. E. (2021, January). How Computational Modeling Can Force Theory Building in Psychological Science. *Perspectives on Psychological Science*, 1745691620970585. doi: 10.1177/1745691620970585

Maturana, H. R., & Varela, F. J. (1980). *Autopoiesis and Cognition: The Realization of the Living* (Vol. 42). Dordrecht: Springer Netherlands. doi: 10.1007/978-94-009-8947-4

Piccinini, G. (2004, September). Functionalism, Computationalism, and Mental Contents. *Canadian Journal of Philosophy*, *34*(3), 375–410. doi: 10.1080/00455091.2004.10716572

Piccinini, G. (2008, January). Computation without Representation. *Philosophical Studies*, *137*(2), 205–241. doi: 10.1007/s11098-005-5385-4

Piccinini, G. (2015). *Physical computation: A mechanistic account*. Oxford: Oxford University Press.

Riegl, S. (2022). *Computational autopoietic theory? How post-cognitivism does not (necessarily) entail anti-computationalism*. Unpublished master's thesis, Radboud University, Nijmegen, The Netherlands.

Varela, F. J. (1979). *Principles of biological autonomy*. New York: North Holland.

Varela, F. J., Maturana, H. R., & Uribe, R. (1974, May). Autopoiesis: The organization of living systems, its characterization and a model. *Biosystems*, *5*(4), 187–196. doi: 10.1016/0303-2647(74)90031-8

Villalobos, M., & Dewhurst, J. (2017). Why post-cognitivism does not (necessarily) entail anti-computationalism. *Adaptive Behavior*, *25*(3), 117–128.