Asynchronous Fixability of a ternary network: “rock-paper-scissor” rule

Florian Bridoux ✉
GREYC, Normandie Univ, UNICAEN, ENSICAEN, CNRS, GREYC, 14000 Caen, France

Gaétan Richard ✉
GREYC, Normandie Univ, UNICAEN, ENSICAEN, CNRS, GREYC, 14000 Caen, France

Abstract

2012 ACM Subject Classification Replace ccsdesc macro with valid one

Keywords and phrases Dummy keyword

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Funding This work was sponsored by ANR C_SyDySi (19-CE48-0007-01)

Introduction

Complex system is a field of research that consider a large number of elements that follow a simple and local rule but that exhibits in there whole complex behaviour. One of its main objective is to build links between properties of the local rule and global behaviour. One use of such research is to give tools and hints to scientists doing modelling. Among many other model, *discrete neural networks* consist on a finite graph representing interaction of binary elements (such as genes [6]) with simple local rules. They were introduced to model neural activity [7]. This formalisation has latter been studied from the computer science approach and has lead to results linking the structure of the underlying graph and dynamical properties of the system (see for example [2]). Latter on, many other work has been done to study the derived model of *automata network* and in particular, one axis try to determine whether the system can be (asynchronously) fixed [3].

Due to the nature of the system, many of previous work has been done on the binary case. Here, we shall extend to the ternary case. The problem is difficult as the binary case has still many unsolved cases. Thus, we have little hope to give complete generic result. However, we aim to present a meaningful example exhibiting interesting properties and being able to extend or disproof results achieved in binary. The rule is base on the *“rock-paper-scissor”* rule that has been studied by Hellouin de Menibus and Le Borgne in [5].

In a first section, we shall giving the different definition and give some insight of the complexity of fixability problem in the generic case. After that, we shall study the spectific case of our rule and show several behavior. In addition to refining the complexities, we shall show that this rule is fixable over any strongly connected graph.

1 Automaton network and fixability

1.1 Definitions

Let $\mathfrak{G} = (V, E)$ be a directed graph ($E \subset V^2$). For an vertex (called *site*) $v \in V$, we define the *predecessors* of v as the set $P(v) = \{v' | (v', v) \in E\}$. A *path* from v_0 to v'_n is a sequence of edges $((v_i, v'_i))_{i \in [0..n]}$ such that $v'_i = v_{i+1}$. It is a *cycle* if $v_0 = v'_n$. A graph is *strongly connected* if, for any distinct pair of edge (v, v'), there is a path from v to v' (and from v' to v).

42nd Conference on Very Important Topics (CVIT 2016).
Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:7
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Let S be a finite set of *states*. A *configuration* is a triplet (V, E, C) where (V, E) is a directed graph and $C \in V^S$ a colouring of vertices.

In general, an automata network is defined as a set of $|V|$ (distincts) functions $f_1, f_2, \cdots, f_n$ such that $f_v : S^{P(v)} \to S$.

In this paper, we shall focus on the case where all functions are uniform and do not depend on the order in the set of predecessors. We keep a dependency on the state of the current vertex and the set of states of its predecessors. We denote as $\mathcal{S}(S)$ the subsets of S and give the following definition:

▶ **Definition 1.** *An* automata network *as a function* $r : (S \times \mathcal{S}(S)) \to S$.

We also chose the *fully asynchronous* updating where only one site is updated at any time. Given a configuration $\mathfrak{C} = (V, E, C)$, the action of the network automata r applied on site $v \in V$ is defined by $\mathfrak{C}' = (V, E, C')$ with for all $v' \in V$,

$$C'(v') = \begin{cases} r(v, P(v)) & \text{if v' = v} \\ C(v') & \text{otherwise} \end{cases}$$

A site v of a configuration $\mathfrak{C}$ is *active* if $r(v, P(v)) \neq v$. That is, action of the automata network on v changes the configuration. A configuration c is said to be a *fixpoint* for an automaton r if there is no active site. Given this, we can define *fixablity* as the existence of a sequence reaching a fixpoint.

▶ **Definition 2** (Fixability)**.**

- *An automata network is* fixable *on a configuration if and only if there exists a sequence of activation $v_1, v_2, \cdots, v_n$ such that the resulting configuration is a fixpoint.*
- *An automata network is* fixable *on a graph $\mathfrak{G}$ if and only if it is fixable over all configurations of $\mathfrak{G}$.*

Fixability as been studied mostly in the boolean case [1, 8, 9].

1.2 Complexity

In this section, we shall give some first elements on the complexity in general case of fixability. One difficulty regarding automata network is the encoding of the input. In this paper, we choose to put emphasis on the influence of the underlying graph. Thus, we work using a "fixed" rule that we only require to be `PTIME` computable.

Our problem `Configuration fixability` is thus the following: given a fixed `PTIME` function, given as input a graph and a configuration, answer whether the automata network using this rule is fixable on the configuration. Omitting the configuration leads immediately to `graph fixability`.

▶ **Proposition 3.** *`Configuration fixability` and `Graph fixability` are PSPACE.*

Proof. Let us consider the *dynamics graph* constructed over the set of configuration with an edge $\mathfrak{C}, \mathfrak{C}'$ if there exists an action changing $\mathfrak{C}$ into $\mathfrak{C}'$. This dynamics graph is exponential with respect to the input graph.

Knowing if a configuration is a fixpoint is linear (just check any site).

For `Configuration fixability`, the problem consist in finding whether there exist a path in the dynamics graph leading to a fixpoint. For this, it is sufficient to enumerate all vertex (linear in size), check whether they are a fixpoint. And, if it is the case, check whether there exists a path between the initial configuration and the designed vertex. This problem (*st-connectivity*) is `NL` in the size of the graph and thus `PSPACE` with respect to our input.

For `Graph fixability`, just iterate the previous algorithm over any configuration. ◀

As there is an exponential number of configurations, if a graph is fixable, then, there exist an (exponential) sequence of action that, starting from any configuration, reaches a fixpoint. One interesting question is to determine the shortest sequence that does this. In particular, if a polynomial sized one exists, then the problem becomes NP. This problem can be seen as an extension of the *Synchronisation problem* [10] for finite automaton. For the case of boolean network automata, several results exists for families of function [4, 3].

▶ **Proposition 4.** *`Configuration fixability` is LINSPACE-hard.*

Proof. The reduction is done by directly encoding Turing-Machine.

The graph consist just on a bi-directed line $V = (v_i)_{i\in[0,n]}$ and $E = ((e_{i-1}, e_i) \cup (e_i, e_{i-1}))_{i\in[1,n]}$ that will be used to encode the current configuration of the Turing machine.

The key point is how to encode the Turing-head and its evolution. To do this, we use a trick by adding a binary information indicating if a state is marked. The transition is done in three steps, first, the new position of the head is created in unmarked state, then the old position is removed (along with updating the colour of the tape, this step checks the existence of the new head). At last, the new head is marked (this steps checks the absence of old marked head).

This ensure that, starting from an initial configuration, there is at most one marked head. If it is the case, either there is no unmarked head among the predecessor and the only active site is the new position of the head; or, in the other case, the only active site is the marked head to remove. At last, if there is only one unmarked head, it is the active site.

If the head goes beyond the tape, it enters a special state that only alternate between two colours.

Starting from the initial configuration of the Turing machine, the evolution reaches a fixpoint if and only if the machine reaches an halting state without exiting the linear size of the tape. ◀

2 "rock-paper-scissor" rule

For the rest of the paper, we shall concentrate on a specific simple but yet interesting rule: **rock-paper-scissor**.

In this section, we consider the "rock-paper-scissor" automaton over the set of states $S = \{0, 1, 2\}$ given by the rule

$$r(v, s) = \begin{cases} v + 1 \mod 3 & \text{if } (v + 1 \mod 3) \in s \\ v & \text{otherwise} \end{cases}$$

Intuitively, this rule changes a vertex's states if there is a "winning" state inside its predecessors. Although simple, this rule exhibits interesting behaviours as studied in [5].

2.1 Counting fixpoints

2.2 Fixability on strongly connected graphs

The main result is that this rule is Fixable on any strongly connected graph.

▶ **Theorem 5.** ***Rock-paper-scissor*** *rule is* **`graph fixable`** *on any strongly connected graph.*

To prove this, let us first introduce two notions that will be useful.

▶ **Definition 6.** *Let C be a cycle of a configuration $\mathfrak{C}$, it is said* a-stable *if there is no active site with state $a \in S$ in the cycle.*

▶ **Definition 7.** *Let C be a cycle of a configuration $\mathfrak{C}$, the* discontinuity *of the cycle is defined as the number of edge (v_i, v'_i) for which $C'(v_i) \neq C'(v'_i)$.*

The discontinuity of a $\mathfrak{C}$ is given by the sum of the discontinuity of all its cycle.

Proof. To prove the result, we shall give an algorithm leading to a fixpoint. For this, we prove that our algorithm strictly reduce (except at the first step) the discontinuity of the configuration as long as it is not reaching a fixpoint.

The algorithm is the following:

Listing 1 Find Rock-paper-scissor fixpoint

```
s = 0
repeat
    while there exist a site with colour s that can be activated
        activate the site
    s = s+1 mod 3
until a fixpoint is reached
```

The first easy remark is that after executing the `while` loop, the configuration is s-stable.

Let us now consider what happen to the dicontinuity of a given cycle during the while loop (except the first one). As the previous `while` was done, the cycle is $s-1$-stable. This implies in particular that any colour s in the cycle is followed either by s or $s+1$. As we activate site in state s (and they become $s+1$ in this case), site can only be activated once at most in the loop. Let us look at the set of activated site in the cycle. Each consecutive sequence of activated site is followed by a site in state $s+1$ (since it cannot be $s-1$ by stability and cannot be s because this site could be activated). Thus changing states of activated site have reduced the discontinuity by one at the end of each consecutive sequence. As it may only add one discontinuity at the start of the sequence. The discontinuity of each cycle decrease.

Moreover, the first activation of a site in the loop is done by having a edge (v_0, v_1) with v_0 in state $c+1$. Since the graph is strongly connected, this edge belongs to a cycle and thus activation of this site will reduce the discontinuity by 1.

Since discontinuity is an integer, the algorithm stops and reaches thus a fixpoint. ◀

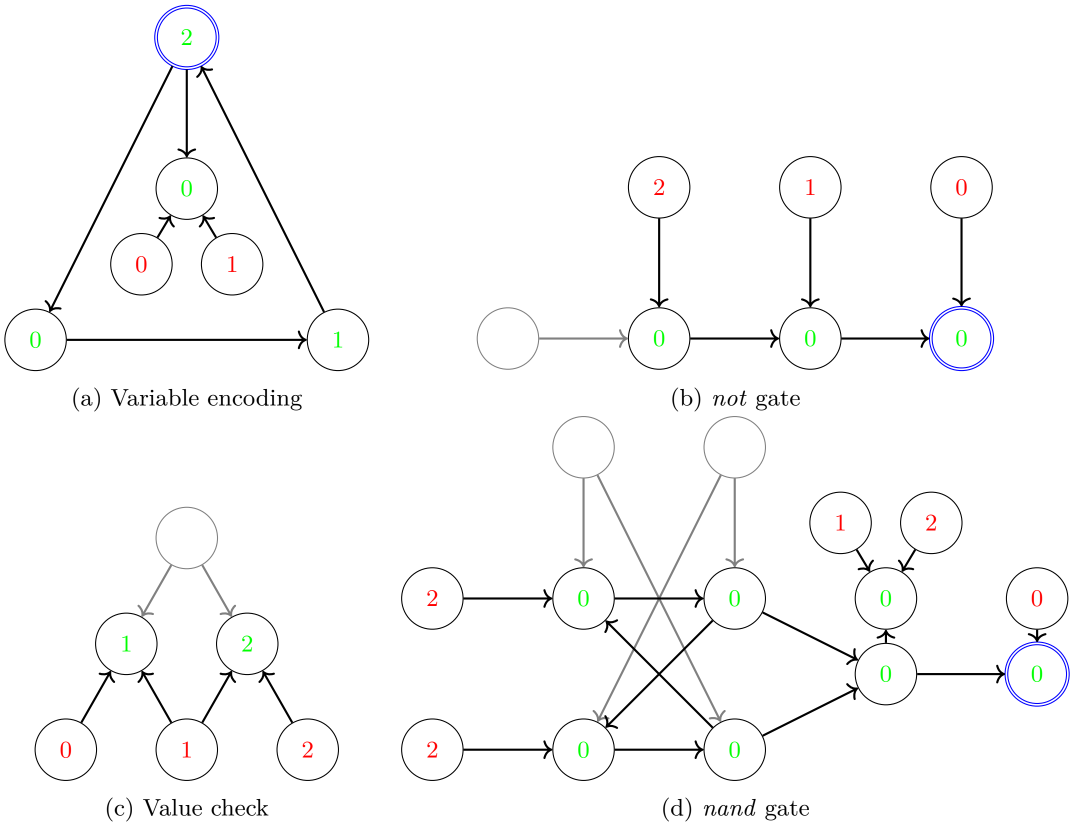
The given algorithm has two characteristics. First, it gives an exponential step solution to the fixability. Second, it depends on the configuration.

One easy remark is that there exists a non strongly connected graph which is not fixable.

2.3 Fixability on configuration

▶ **Theorem 8.** *Configuration Fixability for **Rock-paper-scissor** rule is NP-Hard.*

Proof. The reduction is done from `SAT` problem.



(a) Variable encoding (b) *not* gate (c) Value check (d) *nand* gate

Figure 1 Widgets

The constructed widgets have several vertices (depicted in red) without any inbound edge. Such vertices state will thus never change.

To any SAT formula, it is possible to construct a (polynomial) configuration encoding this formula. We will prove that this configuration is fixable if and only if the formula is satisfiable. This is done in two steps: first, we prove that there is a fixpoint on the underlying graph that has the red vertices with there proper colours if and only if the formula is satisfiable. Then, we prove that, in the latter case, this fixpoint is reachable from the constructed configuration.

The first initial remark is than any state without predecessors always keep the same state throughout the configuration. Such states are depicted in red in figure 1.

Let us look at widget (c). As the left green 1 vertex has three predecessor with two fixed, the only possible value for the input state is one of the two fixed value (thus, 0 or 1). The same argument applied to the green 2 vertex implies that input must be either 1 or 2. Combining both make this widget ensure that input is fixed at 1 in any fixpoint. Using the same argument on the central vertex of widget (a) lead to the conclusion that this widget forces the output to be 0 or 1 in any fixpoint.

For the widget (b), the key point is that given a vertex with two predecessors with distinct values, then the only possible value in a fixpoint is the greater value of the two. Applying the previous remark, if the input of a *not* gate is 0, then, in a fixpoint, the leftmost vertex is necessarily 0, the central one must be 1 and the output 1. In the case of input 1, the vertex must be respectively 2,2,0. Thus this widget implements a not gate.

The last is the nand widget. For this widget, we reuse the same argument as previously. The first remark is that the only possibles values in a fixpoint for the last but one rightmost vertex (*pre-output*) is either 1 or 2 and those values implies that the output is respectively 1 and 0. If the left input is 1, then, it implies that the top-left vertex must be 2 in a fixpoint. If the right input is 1, then the top-right must be also 2 implying that the pre-output must be 2 and the output 0. If the right input is 0, then the top-right must be 0 and thus the pre-output 1 and the output 1. The case 0 as left input and 1 as right input also outputs 1 by symmetry. The last case is when both inputs are 0. In this case, the top left and bottom left vertices must be 0. Due to the return edge inside, the two right vertices must be also 0 and thus, the pre-output must be 1 and the ouput 1.

With all this, we can deduce that if there is a fixpoint reachable from the initial state, then in this solution, the end result is 1, each variable is either 0 or 1 and each gate realising either *not* or *nand*. Thus the encoded formula is satisfiable.

The last thing to do is to prove that assuming that the formula is satifiable, then our graph can reach this fixpoint. As the graph is a DAG, we can restrict ourself to prove how to reach the fixpoint for any widget separately.

◀

2.4 About fixability on graph

3 Conclusion and perspectives

References

1. Noga Alon. Asynchronous threshold networks. *Graph. Comb.*, 1(1):305–310, 1985. doi:10.1007/BF02582959.
2. Julio Aracena, Jacques Demongeot, and Eric Goles Ch. Positive and negative circuits in discrete neural networks. *IEEE Trans. Neural Networks*, 15(1):77–83, 2004. doi:10.1109/TNN.2003.821555.
3. Julio Aracena, Maximilien Gadouleau, Adrien Richard, and Lilian Salinas. Fixing monotone boolean networks asynchronously. *Inf. Comput.*, 274:104540, 2020. doi:10.1016/j.ic.2020.104540.
4. Maximilien Gadouleau and Adrien Richard. On fixable families of boolean networks. In Giancarlo Mauri, Samira El Yacoubi, Alberto Dennunzio, Katsuhiro Nishinari, and Luca Manzoni, editors, *Cellular Automata - 13th International Conference on Cellular Automata for Research and Industry, ACRI 2018, Como, Italy, September 17-21, 2018, Proceedings*, volume 11115 of *Lecture Notes in Computer Science*, pages 396–405. Springer, 2018. doi:10.1007/978-3-319-99813-8_36.
5. Benjamin Hellouin de Menibus and Yvan Le Borgne. Asymptotic behaviour of the one-dimensional “rock-paper-scissors” cyclic cellular automaton. *Annals of Applied Probability*, 2020. URL: https://hal.archives-ouvertes.fr/hal-02084842.
6. Nicolas Le Novere. Quantitative and logic modelling of molecular and gene networks. *Nature Reviews Genetics*, 16(3):146–158, 2015.
7. Warren Mcculloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:127–147, 1943.
8. Tarek Melliti, Damien Regnault, Adrien Richard, and Sylvain Sené. On the convergence of Boolean automata networks without negative cycles. In *Proceedings of AUTOMATA'13*, volume 8155 of *LNCS*, pages 124–138. Springer, 2013.

9 Tarek Melliti, Damien Regnault, Adrien Richard, and Sylvain Sené. Asynchronous simulation of Boolean networks by monotone Boolean networks. In *Proceedings of ACRI'16*, volume 9863 of *LNCS*, pages 182–191. Springer, 2016.

10 Jean-Éric Pin. On two combinatorial problems arising from automata theory. In C. Berge, D. Bresson, P. Camion, J.F. Maurras, and F. Sterboul, editors, *Combinatorial Mathematics*, volume 75 of *North-Holland Mathematics Studies*, pages 535–548. North-Holland, 1983. doi: https://doi.org/10.1016/S0304-0208(08)73432-7.