Available online at www.sciencedirect.com

ScienceDirect

Physics of Life Reviews 31 (2019) 134–156

PHYSICS of LIFE reviews

www.elsevier.com/locate/plrev

Review

Self-referential basis of undecidable dynamics: From the Liar paradox and the halting problem to the edge of chaos

Mikhail Prokopenko [a,*], Michael Harré [a], Joseph Lizier [a], Fabio Boschetti [b], Pavlos Peppas [c,d], Stuart Kauffman [e]

[a] *Centre for Complex Systems, Faculty of Engineering and IT, The University of Sydney, NSW 2006, Australia*
[b] *CSIRO Oceans and Atmosphere, Floreat, WA 6014, Australia*
[c] *Center for AI, School of Software, FEIT, University of Technology Sydney, NSW 2007, Australia*
[d] *Department of Business Administration, University of Patras, Patras 265 00, Greece*
[e] *University of Pennsylvania, Philadelphia, PA 19104, USA*

Available online 8 January 2019
Communicated by Felix Schoeller

Abstract

In this paper we explore several fundamental relations between formal systems, algorithms, and dynamical systems, focussing on the roles of undecidability, universality, diagonalization, and self-reference in each of these computational frameworks. Some of these interconnections are well-known, while some are clarified in this study as a result of a fine-grained comparison between recursive formal systems, Turing machines, and Cellular Automata (CAs). In particular, we elaborate on the diagonalization argument applied to distributed computation carried out by CAs, illustrating the key elements of Gödel's proof for CAs. The comparative analysis emphasizes three factors which underlie the capacity to generate undecidable dynamics within the examined computational frameworks: (i) the program-data duality; (ii) the potential to access an infinite computational medium; and (iii) the ability to implement negation. The considered adaptations of Gödel's proof distinguish between computational universality and undecidability, and show how the diagonalization argument exploits, on several levels, the self-referential basis of undecidability.

Keywords: Self-reference; Diagonalization; Undecidability; Incomputability; Program-data duality; Complexity

1. Introduction

It is well-known that there are deep connections between dynamical systems, algorithms, and formal systems. These connections relate the Edge of Chaos phenomena observed in dynamical systems, to the Halting problem recognized in computability theory, as well as to Gödel's Incompleteness Theorems established within the framework of formal systems. Casti, for example, has explored interconnections between dynamical systems, Gödelian formal logic systems, Turing machines, as well as Chaitin's complexity results, arguing that

[*] Corresponding author.
E-mail address: mikhail.prokopenko@sydney.edu.au (M. Prokopenko).

https://doi.org/10.1016/j.plrev.2018.12.003

> "the theorems of a formal system, the output of a UTM [Universal Turing Machine], and the attractor set of a dynamical process (e.g., a 1-dimensional cellular automaton) are completely equivalent; given one, it can be faithfully translated into either of the others." [1].

A similar triangle of equivalences between Physics (dynamical systems), Mathematics (formal systems) and Computation (algorithms) is discussed by Ilachinski in the context of the Anthropic Principle:

> "Just as Gödel's theorem makes use of logical self-reference to prove the existence of unprovable truths within a mathematical system, and Turing's theorem makes use of algorithmic self-reference to show that a computer cannot fully encompass, or understand, itself, the anthropic principle limits the perceived structure of the universe by the fact that the universe is effectively perceiving itself." [2].

These arguments bring forward several key concepts which underlie the analogies — *undecidability*, *universality* and *self-reference* — and implicate them in the notions of *chaos* and *complexity*.

An undecidable problem is typically defined in computability theory as a decision problem for which it can be shown that a correct yes-or-no answer cannot always be produced by an algorithm. One of the most well-known examples of undecidable problems is the Halting problem: given a description of an arbitrary program (e.g., a Turing machine) and an input, it is impossible to construct an algorithm which would determine whether the program will eventually halt or continue to run forever. In the context of formal logic systems, an undecidable statement is a statement expressible in the system's language which can neither be proved nor disproved within the very same system. The phenomenon of undecidability is present in dynamical systems as well, and needs to be distinguished from deterministic chaos:

> "For a dynamical system to be chaotic means that it exponentially amplifies ignorance of its initial condition; for it to be undecidable means that essential aspects of its long-term behaviour — such as whether a trajectory ever enters a certain region — though determined, are unpredictable even from total knowledge of the initial condition." [3],

where the behavior is meant to be unpredictable without full simulation. While describing an example of undecidable dynamics of a physical particle-motion system with mirrors, Moore has also distinguished between "sensitive dependence" and "algorithmic complexity": in the former case the chaotic dynamics are unpredictable due to imperfect knowledge of initial conditions, while in the latter case (undecidability), "even if the initial conditions are known exactly, virtually any question about their long-term dynamics is undecidable" [4,5]. A very well-studied type of discrete dynamical systems where the classes of ordered, chaotic and complex ("Edge of Chaos") dynamics have been identified and characterized is Cellular Automata (CAs), although the ability to quantitatively separate such classes is often questioned [6–8]. Computationally, CAs can be seen as information-processing systems carrying out a computation on data represented by an initial configuration [9]. Being a computational device, a CA may also be analyzed in terms of undecidable dynamics (although one must carefully specify what questions are put to a test), and such an analysis invariably involves the concept of computational universality [10,11].

As pointed out by Bennett [3], "a discrete or continuous dynamical system is called computationally universal if it can be programmed through its initial conditions to perform any digital computation", and moreover, "universality and undecidability are closely related: roughly speaking, if a universal computer could see into the future well enough to solve its own halting problem, it could be programmed to contradict itself, halting only if it foresaw that it would fail to halt." This succinct phrase emphasizes that undecidability is a consequence of universality, and reaches to the core of the self-referential argument utilized in demonstrating undecidability within various computational frameworks.

This brings us to one of the central objectives of this work — elaborating on the role played by self-reference in distributed computation carried out by CAs.

The Liar's Paradox which has captured the imagination of philosophers and logicians for thousands of years is a self-referential statement the truth or falsity of which cannot be assigned without a contradiction: for example, the paradox can be presented as a statement of a person declaring that "everything I say is a lie", or more formally as "this statement is unprovable". It has achieved prominence in modern philosophical logic largely due to the motivation it provided to various proofs of incompleteness, undecidability, and incomputability. A fundamental aspect of this paradox, and the works which incorporated its main idea, is self-reference: the way the statement refers to its own validity. As we shall see, there is a close but subtle difference between the concept of self-reference and the diagonalization argument (dating back to Cantor's diagonal argument), both of which play important roles in formal systems, algorithms, and dynamical systems.

Despite the early realization of fundamental interconnections between formal systems, algorithms (Turing machines), and dynamical systems, the precise set of detailed analogies remains elusive, leading sometimes to inaccurate parallels. For instance, Casti offers a "logical route" to chaos, claiming that "there is a direct chain of connection linking the existence of strange attractors, Chaitin's results on algorithmic complexity, and Gödel's Incompleteness Theorem" [1]. As he points out, Cellular Automata theorists, while distinguishing between "strange attractors" and "quasiperiodic orbits", "lump both types into the same category of "strange attractor" when trying to make contact with the traditional dynamical systems literature" [1]. Obviously, the analysis presented by Casti has since been further illuminated by studies of class IV CAs ("quasiperiodic orbits"), highlighting the differences between their complex dynamics at the edge of chaos from class III CAs ("strange attractors") [11–18].

It has been long-conjectured that "complex" systems evolve to the "edge of chaos", that is, their dynamical behavior is neither *ordered*, i.e., globally attracting a fixed point or a limit cycle, nor *chaotic*, i.e., sensitive to imperfectly known initial conditions [11,12,19,20]. These broad claims have been questioned, and indeed it has been demonstrated that computational tasks can certainly be achieved away from the edge of chaos [21]. A more appropriate interpretation, without claims appealing to evolution, may be that (i) while all classes of systems undertake intrinsic computation (and indeed the most appropriate type of system for handling particular computational tasks may be distant from the edge of chaos [21]), (ii) there is evidence that the edge of chaos offers computational advantages (e.g. blending information storage and transfer capabilities) that are advantageous for a priori unknown or indeed general purpose computational tasks [22–24]. Ilachinski also directly mapped (a) halting computation of CAs to class I ("frozen" dynamics, i.e., fixed-points) and class II (periodic dynamics, i.e., limit cycles); (b) non-halting computation to class III (chaotic dynamics, i.e., "strange attractors"), and (c) undecidable computation to class IV ("arbitrarily long transients") [2]. Nevertheless, it has also been argued that some chaotic systems may also be universal, and hence, not decidable, contradicting the thesis that universal computation can only happen at the "edge of chaos", while acknowledging that the existence of a chaotic universal CA has not yet been demonstrated and remains an open question [25–27].

However, the difficulty in identifying what kind of CA dynamics corresponds to the undecidability appears not only due to the lack of a standard classification, but also due to different computational structures employed by CAs and say, Turing machines. In particular, one needs to take special care in drawing parallels between a CA running on an initial configuration, on the one hand, and a formal system inferring theorems from a set of axioms, on the other hand. Indeed, undecidable statements of a formal system which may be more akin to "quasiperiodic orbits" (class IV CAs) rather than "strange attractors" (class III CAs), might be so only with respect to a given initial configuration. Furthermore, in order to relate the attractors of CAs dynamics to the outcomes of Turing machines, or to the theorems derived by formal systems, a consideration must be given to carefully setting up a termination condition for CAs.

While the program-data duality allows us to freely move elements of a computational system between the *program* (a CA's rule-table, a Turing machine's transition function, or a formal system's rules of inference) and the *data* (a CA's initial configuration, a Turing machine's input tape, a formal system's axioms), the type of the eventual dynamics and hence, a possible classification, depends on both components. Thus, a classification scheme which, in principle, aims to classify a program running on *all* inputs, cannot distinguish between the types corresponding to halting, non-halting and undecidable decision problems which are specifically defined for a system with both program and data. The classification problem itself has been shown to be undecidable for a broad range of cases [6–8,28].

Finally, while the key role played by the self-reference in proofs of undecidability in various computational frameworks is beyond doubt, its precise use in dynamical systems, and CAs specifically, has not been demonstrated explicitly. As discussed by [29], in a dynamical system, the Liar's paradox may take the following form: "the system is not stable if and only if it can be shown to be stable". This analogy is not a perfect equivalence, as it simply entails that there is no method for determining the stability of such a system [29]. However, rather than pointing out that a dynamical system is computationally equivalent to an algorithm and then restating the paradox in the language of dynamical systems, it could be more elucidating to constructively demonstrate how and where self-reference is implicated in the structure and dynamics of a CA.

Such an undertaking is the main focus of our study: without engaging in a philosophic debate on the nature of the self-reference (which continues to be vigorously discussed in modern philosophical logic), we shall attempt to essentially reconstruct main elements of Gödel's proof for Cellular Automata. In doing so, we shall find more precise and fine-grained parallels between the key elements of three computational frameworks (formal systems, Turing machines, Cellular Automata), some of which have been pointed out previously [1,2,30], while some have become apparent as a result of the direct comparison between the respective adaptations of Gödel's proof. These adaptations, we hope, can serve the second purpose of this study, aiming to make Gödel's proof and the related concepts of self-reference, diagonalization, universality and undecidability more accessible to the cross-disciplinary field of Complex Systems.

2. Methods

2.1. Formal systems and the Liar paradox

2.1.1. Technical preliminaries

We shall briefly define formal systems in order to formulate the Liar's Paradox and establish the connections to self-reference and diagonalization. In doing so, we shall begin with original definitions of mathematical and elementary formal systems by Smullyan [31], which utilise the concept of well-formed formulas built from some symbols. Then we extend the definition of a formal system with a grammar component which specifies how well-formed formulas are constructed in general.

Following Smullyan [31], we can define a mathematical system with at least three items

$$\mathcal{F} = \langle \mathcal{A}_\mathcal{F}, \mathcal{X}_\mathcal{F}, \mathcal{R}_\mathcal{F} \rangle$$

where

1. $\mathcal{A}_\mathcal{F}$ is an alphabet, i.e., an ordered finite set of symbols, so that $\mathcal{A}^*_\mathcal{F}$ is the set of words (strings) that can be formed as finite linear sequences of symbols from $\mathcal{A}_\mathcal{F}$ (i.e., $\mathcal{A}^*_\mathcal{F}$ is formed by the Kleene operator applied to $\mathcal{A}_\mathcal{F}$);
2. $\mathcal{X}_\mathcal{F} \subseteq \mathcal{A}^*_\mathcal{F}$ is a specific set of axioms;
3. $\mathcal{R}_\mathcal{F}$ is a finite set of relations in $\mathcal{A}^*_\mathcal{F}$ called rules of inference.

Axioms serve as premises for further inferences, by the *inference rules*, which can be stated in a generic form:

$$\text{zero or more premises} \Rightarrow \text{conclusion}$$

For example, the *modus ponens* rule of propositional logic $a, a \to b \Rightarrow b$, infers the conclusion b whenever a and $a \to b$ have been obtained (either as given axioms, or as previous inferences). Axioms and inference rules are used to derive (i.e., prove) theorems of the system.

Typically, an expression W is said to be derivable or formally provable in $\mathcal{F}$ if and only if there is a finite sequence of expressions $W_1, \ldots, W_n$ in which $W \equiv W_n$ and every W_i is either an axiom or results from the application of an inference rule to earlier expressions in the sequence [32,33]. We follow the standard notation $\mathcal{F} \vdash W$ expressing that W is derivable in the formal system $\mathcal{F}$, in other words that there is a proof of W in $\mathcal{F}$, i.e., W is a theorem of $\mathcal{F}$. However, in order to call W a theorem, one still needs to either apply some external criterion distinguishing W from intermediate derivations in advance, as a target expression, or recognize its standing as having a special salience at the meta-level, capturing it as a theorem (current developments are not able to formally distinguish such salience).

In forming the set of words $\mathcal{A}^*_\mathcal{F}$ we did not need to follow any additional syntactic constraints, but one may choose to focus only on *well-formed formulas* (abbreviated as wff's), constructed from the alphabet $\mathcal{A}_\mathcal{F}$ following some grammar. The formalization of a grammar $\mathcal{G}_\mathcal{F} = \langle \mathcal{A}_\mathcal{F}, \mathcal{N}_\mathcal{F}, \mathcal{P}_\mathcal{F}, \mathcal{S}_\mathcal{F} \rangle$ consists of the following components [34]:

1. a finite set $\mathcal{A}_\mathcal{F}$ of terminal symbols;
2. a finite set $\mathcal{N}_\mathcal{F}$ of nonterminal symbols, that is disjoint with $\mathcal{A}^*_\mathcal{F}$, i.e., the strings formed from $\mathcal{A}_\mathcal{F}$;
3. a finite set $\mathcal{P}_\mathcal{F}$ of production rules of the form $(\mathcal{A}_\mathcal{F} \cup \mathcal{N}_\mathcal{F})^* \mathcal{N}_\mathcal{F} (\mathcal{A}_\mathcal{F} \cup \mathcal{N}_\mathcal{F})^* \to (\mathcal{A}_\mathcal{F} \cup \mathcal{N}_\mathcal{F})^*$, so that each production rule maps from one string of symbols to another, with the "head" string containing an arbitrary number of symbols provided at least one of them is a nonterminal;
4. the start symbol $\mathcal{S}_\mathcal{F} \in \mathcal{N}_\mathcal{F}$.

The terminal symbols may appear in the output of the production rules but cannot be replaced using the production rules, while nonterminal symbols can be replaced. For example, the grammar $\mathcal{G}_\mathcal{F}$ with $\mathcal{N}_\mathcal{F} = \{\mathcal{S}_\mathcal{F}\}$, $\mathcal{A}_\mathcal{F} = \{a, b\}$, and $\mathcal{P}_{\mathcal{F}}$ with two production rules $\mathcal{S}_{\mathcal{F}} \to a\mathcal{S}_{\mathcal{F}}b$ and $\mathcal{S}_{\mathcal{F}} \to ba$, generates wff's $a^n bab^n$, for $n \geq 0$, e.g., ba, $abab$, $aababb$, and so on, by applying the first rule n times, followed by one application of the second rule.

Following more recent treatments of formal systems, one may explicitly include components of a grammar G in the definition

$$\mathcal{F} = \langle \mathcal{A}_{\mathcal{F}}, \mathcal{N}_{\mathcal{F}}, \mathcal{P}_{\mathcal{F}}, \mathcal{X}_{\mathcal{F}}, \mathcal{R}_{\mathcal{F}} \rangle$$

where

1. $\mathcal{A}_{\mathcal{F}}$ is an alphabet, i.e., an ordered finite set of symbols;
2. $\mathcal{N}_{\mathcal{F}}$ is a finite set of nonterminal symbols, including the start symbol $\mathcal{S}_{\mathcal{F}} \in \mathcal{N}_{\mathcal{F}}$, that is disjoint with $\mathcal{A}^*_{\mathcal{F}}$;
3. $\mathcal{P}_{\mathcal{F}}$ is a finite set of production rules of the form $(\mathcal{A}_{\mathcal{F}} \cup \mathcal{N}_{\mathcal{F}})^* \mathcal{N}_{\mathcal{F}} (\mathcal{A}_{\mathcal{F}} \cup \mathcal{N}_{\mathcal{F}})^* \to (\mathcal{A}_{\mathcal{F}} \cup \mathcal{N}_{\mathcal{F}})^*$;
4. $\mathcal{X}_{\mathcal{F}}$ is a specific set of axioms, each of which must be a wff;
5. $\mathcal{R}_{\mathcal{F}}$ is a finite set of relations in the set of wff's, called rules of inference.

That is, while the production rules in $\mathcal{P}_{\mathcal{F}}$ are used to produce wff's, the rules of inference in $\mathcal{R}_{\mathcal{F}}$ are required to derive theorems. We would like to point out that if we consider all words (strings) in $\mathcal{A}^*_{\mathcal{F}}$ as wff's, then the grammar would not be constraining the space of possible inferences. In a special case that a formal system contains negation, a system is called consistent if there is no wff W such that both W and $\neg W$ can be proved.

It is usually required that there is a decision procedure (utilizing $\mathcal{P}_{\mathcal{F}}$) for deciding whether a formula is well-formed or not. In other words, it is generally assumed that the production rules are decidable: there is an algorithm such that, given an arbitrary string x, it can decide whether x is a wff or not. Inference rules need also be decidable in the following sense: for each inference rule $R \in \mathcal{R}_{\mathcal{F}}$, there needs to be an algorithm such that, given a set of wff $x_1, \ldots, x_n$ and a wff y, the algorithm can decide if R can be applied with input $x_1, \ldots, x_n$ and produce output y. In general, we assume that we deal with *recursive* formal systems, that is, the set of axioms is decidable and the set of all provable sentences (i.e., the set of all theorems) is *recursively enumerable* or semi-decidable: if, given an arbitrary wff, there is an algorithm which correctly determines when the formula is provable within the system, but may either produce a negative answer or return no answer at all when the formula is not provable within the system.

Many important problems expressible in formal systems are undecidable, and this is captured in Gödel's Incompleteness Theorems about any formal system with first-order logic (first-order predicate calculus) and containing Peano's axioms of arithmetic: (i) any such formal system is such that, if it is consistent, then it is incomplete: there are wff's which can neither be proved nor disproved; (ii) moreover, such a formal system cannot demonstrate its own consistency.

2.1.2. *Formal undecidability*

We shall discuss several essential steps required in a typical proof of Gödel Incompleteness Theorems. Firstly, as we are dealing with arithmetic, we need to name, i.e., give a formal term ("numeral"), to each number: this is achieved by canonically denoting the natural number n by numeral $\underline{n}$. Assuming that the primitive symbols, i.e. constant signs, such as '0' (zero) or 'S' (denoting "an immediate successor of ...") [32,35] are available (directly or via interpretation), the canonical way to represent a natural number 'n' in a formal system is via the numeral $\underline{n}$;

$$\underline{n} \equiv \underbrace{S \ldots S}_{n \text{ times}} 0.$$

One of the core insights of Gödel was to encode the wff's of a formal system by natural numbers, by an "arithmetization", or "Gödel numbering", of the wff's. Formally, for every wff W, the "Gödel numbering" scheme produces a natural number $\mathcal{G}(W)$, i.e., the "Gödel number", which is further encoded by a numeral. Such a code, the name of the "Gödel number" of a formula W, is denoted as $\ulcorner W \urcorner$.

To exemplify this, we firstly assign a natural number to each primitive symbol s of the formal system (called the symbol number of s), e.g., symbol "0" is assigned number 1 and symbol "=" is assigned number 5. Then we consider the wff W: "0 = 0". The Gödel number for this formula is uniquely produced as the corresponding product of powers of consecutive prime numbers $(2, 3, 5, \ldots)$, as $\mathcal{G}(\text{“}0 = 0\text{”}) = 2^1 \times 3^5 \times 5^1 = 2 \times 243 \times 5 = 2430$. The name of the Gödel number $\ulcorner\text{“}0 = 0\text{”}\urcorner$ is the numeral $\underline{2430}$. Importantly, knowing $\mathcal{G}(\text{“}0 = 0\text{”}) = 2430$ allows us to uniquely decode back into the wff's (due to the unique-prime-factorization theorem), by finding the unique sequence of prime factors, with associated exponents [35,36]. Gödel numbers are computable, and it is important to note that it is also effectively decidable whether a given number is a Gödel number or not. Formally, $\ulcorner W \urcorner$ is the numeral $\underline{\mathcal{G}(W)}$, where $\mathcal{G}(W)$ is the Gödel number of W [32,37]:

$$\ulcorner W \urcorner \equiv \underbrace{S \ldots\ldots S}_{\mathcal{G}(W)\text{ times}} 0.$$

One of the essential steps implicit in Gödel's proof is the Self-reference lemma [36,38]:

Lemma 1. *Let $Q(x)$ be an arbitrary formula of formal system $\mathcal{F}$ with only one free variable. Then there is a sentence (formula without free variables) W such that*

$$\mathcal{F} \vdash W \leftrightarrow Q(\ulcorner W \urcorner) .$$

This lemma is sometimes called the Fixed-point lemma or the Diagonalization lemma. This result was explicitly presented in 1934 by Carnap [39], phrased in different language, and was also used by Tarski in 1936 in proving the undefinability theorem: *arithmetical truth cannot be defined in arithmetic* [40]. The Self-reference Lemma establishes that for any formula $Q(x)$ that describes a property of a numeral, there exists a sentence W that is logically equivalent to the sentence $Q(\ulcorner W \urcorner)$. The arithmetical formula $Q(x)$ describes a property of its argument, e.g., a numeral x, and hence, the expression $Q(\ulcorner W \urcorner)$ describes a property of the numeral $\ulcorner W \urcorner$. This is the numeral of the Gödel number of the formula W itself. Since the formula W is logically equivalent to the formula $Q(\ulcorner W \urcorner)$, one can say that the formula W is referring to a property of itself (being an argument of the right-hand side).

Strictly speaking, as pointed out by [36], the lemma only provides a (provable) material equivalence between W and $Q(\ulcorner W \urcorner)$, and one should not claim "any sort of sameness of meaning". It is, nevertheless, illustrative to consider a related result, a variant of the Mocking Bird Puzzle [38], which reflects the idea of the Lemma's proof and constructs a self-referential relation.

> "We are given a collection of birds. Given any birds B, C, if a spectator calls out the name of C to B, the bird B responds by calling back the name of some bird $B(C)$ (Thus each bird B induces a function from birds to birds.) If $B(C) = C$, then we say that B is fixated on C. We call B egocentric if B is fixated on itself. We are given that the set of functions induced by the birds is closed under composition (more explicitly, for any birds B, C there is a bird D such that for every bird X, $D(X) = B(C(X))$). We are also given that there is a bird M (called a mocking bird) such that for every bird B, $M(B) = B(B)$. The problem is to prove that every bird is fixated on at least one bird, and that at least one bird is egocentric."

The proof has several instructive steps [38], reproduced here for convenience. Firstly, applying the closure under composition to a mocking bird M we note that there must be a bird D such that for every bird X, we have $D(X) = B(M(X))$. Then substituting D for X, we obtain $D(D) = B(M(D))$. By definition of a mocking bird, $M(D) = D(D)$, and so we reduce to $D(D) = B(D(D))$, showing that bird B is fixated on the bird $D(D)$, completing the first part (proving that every bird is fixated on at least one bird). Hence, the mocking bird must also be fixated on some bird E, that is, $M(E) = E$. Again, by definition of a mocking bird, $M(E) = E(E)$, yielding $E(E) = E$ and completing the second part (proving that at least one bird is egocentric).

Obviously, the substitutions in this proof were made simple by ignoring the encoding and decoding of birds as arguments but it is still interesting to note that the mocking bird can be seen as an analogy of universal computation (a universal Turing machine or a universal cellular automaton, capable of emulating computation of any other device, see Section 2.3.4).

One now needs to define the provability predicate $\text{Provable}_{\mathcal{F}}(x)$ which captures the property of x being provable in $\mathcal{F}$. Let the formula $\text{Proof}_{\mathcal{F}}(y, x)$ strongly represent the binary relation "y is (the Gödel number of) a proof of the formula (with the Gödel number) x" (following [36], we note that it is always decidable whether a given sequence of formulas y constitutes a proof of a given sentence x, according to the rules of the formal system $\mathcal{F}$). The property of being provable in $\mathcal{F}$ can then be defined as $\exists y \text{Proof}_{\mathcal{F}}(y, x)$, abbreviated as $\text{Provable}_{\mathcal{F}}(x)$.

The final step leading to Gödel's First Incompleteness Theorem is an application of the Self-reference lemma to the negated provability predicate $\neg\text{Provable}_{\mathcal{F}}(x)$:

$$\mathcal{F} \vdash W \leftrightarrow \neg\text{Provable}_{\mathcal{F}}(\ulcorner W \urcorner) \,. \tag{1}$$

This then formally demonstrates that the system $\mathcal{F}$ can derive that W is true if and only if it is not provable in $\mathcal{F}$. Furthermore, if the system $\mathcal{F}$ is consistent, then it can be shown that the sentence W is neither provable nor disprovable in $\mathcal{F}$, showing the system to be incomplete. It is important to point out that the Gödel sentence W can be constructed as a well-formed formula of the system $\mathcal{F}$.

Common treatments of this seminal result interpret this theorem somewhat less formally, e.g., stating that the Gödel sentence W expresses or refers to its own unprovability [41], analogous to the Liar paradox: (the sentence claiming "this sentence is false" that can be neither true nor false). This can be traced back to the original Gödel's work, where he informally wrote: "We therefore have before us a proposition that says about itself that it is not provable." [42, p. 149].

What is important for our main purposes is that Gödel's First Incompleteness Theorem can be used to demonstrate undecidability [36]. A formal system $\mathcal{F}$ is decidable if the set of its theorems is strongly representable in $\mathcal{F}$ itself: there is some formula $\text{P}(x)$ of $\mathcal{F}$ such that

$$\begin{aligned} \mathcal{F} \vdash \text{P}(\ulcorner W \urcorner) \text{ whenever } \mathcal{F} \vdash W, \text{ and} \\ \mathcal{F} \vdash \neg\text{P}(\ulcorner W \urcorner) \text{ whenever } \mathcal{F} \nvdash W \,. \end{aligned} \tag{2}$$

For a weakly representable set of theorems only the first line of (2) is required (semi-decidability), that is, negations are not necessarily "attributable" to non-derivable formulas. However, it is possible to construct, within the system $\mathcal{F}$, a Gödel sentence V^{P} relative to $\text{P}(x)$:

$$\mathcal{F} \vdash V^{\text{P}} \leftrightarrow \neg\text{P}(\ulcorner V^{\text{P}} \urcorner) \,. \tag{3}$$

A contradiction follows, and hence, at least for this sentence the strong representability does not hold, and therefore, $\mathcal{F}$ must be undecidable. Crucially, the Gödel sentence V^{P} is constructed as $V(\ulcorner V(x)\urcorner)$ for some wff $V(x)$ with a free variable, and so our central expression (3) explicitly states

$$\mathcal{F} \vdash V(\ulcorner V(x)\urcorner) \leftrightarrow \neg\text{P}(\ulcorner V(\ulcorner V(x)\urcorner)\urcorner) \,. \tag{4}$$

This perspective makes it explicit that the self-reference (or diagonalization) is used twice: inside and outside of the representative predicate $\text{P}(x)$, which is "sandwiched" between the two self-references [43].

The interrelationships played by fixed points, diagonalization, and self-reference in proofs of Gödel's first incompleteness theorem are discussed in [32], and we shall revisit these aspects in Section 3.1.

2.2. *Turing machines and the halting problem*

Turing machines were introduced as a formal model of computation, intended as an abstract general-purpose computing device which modifies symbols on an infinite tape (*data*) according to a finite set of rules (*program*). Prior to Turing's work the concept of an "effective process" had not been formalized, and so Turing's insight was to define the notion of an *algorithm*: an automated process that is able to proceed, using a set of predefined rules, through a finite number of well-defined successive states, eventually terminating at a final state and producing an output.

The infinite tape of a Turing machine (TM) is divided into discrete cells, thus implementing an unlimited memory capacity. The data are encoded, using some alphabet, as the initial input string, while the remaining cells on the tape contain blank symbols. A TM employs a tape head which can move left and right across the tape, as well as read and write symbols contained in the cell to which the head points, thereby creating strings of symbols, from an *alphabet* Γ, on the tape (a *string* over an alphabet is defined as a finite sequence of symbols from that alphabet, while a *language* over an alphabet is defined as a set of strings [44]).

These actions of the machine simulate an algorithm by following, at every given state, the rules described in its transition function, defined over a set of *internal* states Q and the alphabet Γ, as $\mu : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$. For example, if the machine is at a state q_1 and the tape head reads symbol a, then according to the machine's rules it may need to overwrite symbol a with symbol b on the tape, following which the machine switches its state to q_2 and moves to the right. Formally, this example can be expressed as $\mu(q_1, a) = (q_2, b, R)$.

The machine is able to distinguish certain predefined final states. For instance, if the machine enters the state $q_{acc} \in Q$, this indicates that the initial input is *accepted* by the machine, while entering another state $q_{rej} \in Q$ represents that the input is *rejected*. Both of these outcomes cause the machine to halt, otherwise, the machine will continue its transitions forever [44, pp. 138 – 140].

2.2.1. *Technical preliminaries*

A Turing machine, as adopted here following Sipser [44, p. 140] and Hopcroft and Ullman [45, p. 81], is a tuple

$$M = \langle Q, \Sigma, \Gamma, \mu, q_0, q_{acc}, q_{rej} \rangle$$

where Q, Σ and Γ are non-empty finite sets, and

1. Q is a set of states;
2. $q_0 \in Q$ is the start state;
3. $q_{acc} \in Q$ is the accept state;
4. $q_{rej} \in Q$ is the reject state;
5. Σ is the input alphabet not containing the blank symbol $\sqcup$;
6. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma \setminus \{\sqcup\}$;
7. $\mu : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is a partial function called the transition function, where L is left shift, and R is right shift. If μ is not defined on the current state and the current tape symbol, then the machine halts.

The transition function μ may be undefined for some arguments. Specifically, the machine M halts in the accept q_{acc} state (the initial tape contents is then said to be accepted by M) or the reject q_{rej} state (the initial input tape is said to be rejected by M). With this definition, the *output* of the computation, if it halts, is the determination whether the initial input is accepted or rejected. However, one may equivalently define a TM with just one halting state $q_{halt} \in Q$, instead of two explicit accept and reject states. In this case, if the machine halts, i.e. if it enters the state q_{halt}, then some content written on the same tape captures the actual output of the machine's computation. The precise position of such output on the tape depends in general on some *convention* and may be recognized in relation to the head position in a predesignated way, e.g., the head pointing to the cell containing the leftmost symbol of the output.

This means that in the definition of a TM with two final states q_{acc} and q_{rej}, the initial tape input represents both some initial data and some target to be verified (to be either accepted or rejected): the final content written on the tape when the machine halts at either q_{acc} or q_{rej} does not matter. On the contrary, in the alternative definition with just one halting state q_{halt}, the target is not included on the tape's initial input: instead it is expected to be found as the output on the tape when the machine halts.

In the first case, when the target is given within the input tape, the machine needs to only accept or reject this initial input. In the second case the final output needs to be explicitly generated on the tape at the end of computation. Such flexibility in embedding the target reflects the duality of the data and the program in TMs, in the sense that a part of the input data may instead be represented in the internal machinery, and vice versa. Technically, one may construct a TM working with an empty input tape, while solving a task completely embedded in the transition function over a certain set of internal states.

Given the current state q and the current content on the tape in the form uv, where two strings u and v are formed by symbols from Γ, with the head pointing to the first symbol of v, one may define a *configuration* of the TM as $u\ q\ v$ [44, p. 140]. For example, $11q_1011$ is the configuration when the tape is 11011, the current state is q_1, and the head points to 0.

A Turing machine capable of simulating *any* other TM is called a universal Turing machine (UTM) and provides a standard for comparison between various computational systems. In fact, the problems solvable by a UTM are exactly those problems solvable by an algorithm or any effective method of computation.

2.2.2. *Incomputability*

A Turing machine M recognises the language L_M if and only if the set L_M contains all the strings that machine M accepts.

In demonstrating the Halting Problem for TMs, we will show, following Sipser [44, p. 165], the undecidability of the language

$$A_{TM} = \{[M, w] \mid M \text{ is a TM and } M \text{ accepts the string } w\}\ ,$$

where strings w are formed by symbols from the alphabet Σ, that is, all strings in the set Σ^* formed by the Kleene operator, and $[\cdot]$ denotes an *encoding* of an object into a string using the alphabet Σ. Specifically, one may construct the encoding of a TM M, denoted $[M]$, into a regular string that comprises the *description* of the tuple M. If needed, the string $[M]$ may be further encoded in a binary regular form. One may also encode compound objects, for example, create an encoding $[M, w]$ of two elements M and w together, as long as there is a way to interpret such an encoding as having two components. In terms of computability, $[\cdot]$ and its partial inverse (i.e., *decoding*) must be effectively computable.

It will be crucial to deal with encodings $[M, [M]]$ so that such an input to another TM P can be decoded into two components: the description of the machine M and the input string $[M]$ into the machine M itself. The practical implementation of a decoding can vary, and one example (constructing, in fact, a universal TM simulating a machine M) separates the input data $[M]$ from the description of the machine M by three consecutive c's [45, p. 102–104], i.e., by a specific symbol sequence.

A typical approach to the proof of undecidability of language A_{TM} involves an assumption that A_{TM} is decidable leading to a contradiction. That is, we assume that there exists a decider TM P (note the analogy with the representative predicate $\mathrm{P}(x)$ used in the proof of undecidability of formal systems) such that on input $[M, w]$, where M is a TM and w is a string, the decider P halts and accepts w if M accepts w, while P halts and rejects w if M fails to accept w. Formally, the decider machine P is defined as

$$P([M, w]) = \begin{cases} \textit{accept} & \text{if } M \text{ accepts } w \\ \textit{reject} & \text{if } M \text{ does not accept } w \end{cases} \tag{5}$$

As an aside, the decider machine P is not a UTM that can simulate an arbitrary TM on arbitrary input. Unlike the decider P which rejects when M loops on w, a UTM simulating M would run forever on w if M runs forever on w. It is the assumed decidability of the universal decider P which will be refuted in the proof.

Then we construct another machine V that is able to (i) interpret its input $[M]$ as the encoding of some TM M, (ii) invoke, as a subroutine, the decider machine P with input $[M, [M]]$, and (iii) once the decider P halts with either accept or reject (which is ensured by the assumption that P must halt on any input $[M, w]$), the machine V inverts the outcome of P. That is, the machine V accepts the input $[M]$ if $P([M, [M]])$ rejects its compound input (which happens, by definition of P, if M does not accept $[M]$), and rejects if $P([M, [M]])$ accepts (that is, if M accepts $[M]$). Formally, the inverter machine V, which includes three distinct steps, is defined as follows:

$$V([M]) = \begin{cases} \textit{reject} & \text{if } M \text{ accepts } [M] \\ \textit{accept} & \text{if } M \text{ does not accept } [M] \end{cases} \tag{6}$$

In creating the input $[M, [M]]$ for the decider machine P we forced the machine M to run on the input representing its own description $[M]$. This is a manifestation of self-reference (similar to the "inside" self-reference used in construction of the Gödel sentence).

It is also important to realise that the input to the inverter machine V is given by the encoding $[M]$ and not by the compound object $[M, [M]]$ which is constructed by V before calling the "sandwiched" decider subroutine P. This construction is possible because both the encoding and decoding are effectively computable (again we draw an analogy with the encoding and decoding utilized by Gödel numbering scheme $\ulcorner\ W\ \urcorner = \mathcal{G}(W)$).

The final step is to run the inverter machine V on itself, that is, to consider $V([V])$ (in analogy to the "outside" self-reference in Gödel's proof):

$$V([V]) = \begin{cases} \textit{reject} & \text{if } V \text{ accepts } [V] \\ \textit{accept} & \text{if } V \text{ does not accept } [V] \end{cases} \tag{7}$$

This is, of course, a contradiction analogous to the Liar's Paradox (or the inconsistency shown by the Gödel sentence in formal systems): the inverter machine V rejects its input $[V]$ whenever V accepts $[V]$. This contradiction shows the impossibility of the decider TM P, and hence, the undecidability of language A_{TM}. One corollary is that the language A_{TM} is TM recognisable but not decidable

As the proof shows, the undecidability arises due to the self-referential ability of a TM to interpret and run an input which encodes its own description, reflecting the program-data duality. The program-data duality, allowing programs to interpret other programs (sets of rules) as data (encoded strings), makes it possible for TMs to answer questions about, and ultimately completely emulate, the behaviour of other TMs. It is this implicit self-referential ability that results from the program-data duality that leads to the undecidability and The Halting Problem.

2.3. Cellular Automata and the edge of chaos

2.3.1. Technical preliminaries

A Cellular Automaton (CA) is a discrete dynamical system C [9] defined on a d-dimensional lattice c. Each lattice site (cell) c_i takes a value from a finite alphabet A_C, i.e., $c_i \in A_C$, where the indexing reflects the dimensionality and geometry of the lattice [46]. For example, for a 1-dimensional CA ($d = 1$), index $i \in \mathbb{Z}$, the set of integers. A configuration c of cells in the lattice is a bi-infinite sequence of specific cell values c_i, that is, $c = (\ldots, c_{-2}, c_{-1}, c_0, c_1, c_2, \ldots)$, for instance, in a 1-dimensional CA with a binary alphabet $A_C = \{0, 1\}$ a configuration may look like $(\ldots, 0, 1, 1, 0, 1, \ldots)$. Most applied work with CAs considers finite automata, but infinity is necessary to generate undecidable dynamics, similarly to the infinite tape in TMs.

Each cell is updated in discrete time steps t according to a deterministic *local* rule ϕ_C involving values of r neighbouring cells, and by convention a cell is included in its neighbourhood:

$$\phi_C : A_C^{(2r+1)^d} \to A_C \tag{8}$$

so that the value of the i-th cell at time t is updated as follows:

$$c_i^t = \phi_C(c_{i-r}^{t-1}, c_{i-r+1}^{t-1}, \ldots, c_{i+r}^{t-1}) \tag{9}$$

The set of all configurations will be denoted as $\Psi_C = A_C^{\mathbb{Z}^d}$. This local rule yields a global mapping (global rule) setting temporal dynamics on the lattice:

$$\Phi_C : \Psi_C \to \Psi_C \tag{10}$$

The configuration c at time t is completely determined by the preceding configuration:

$$c^t = \Phi_C(c^{t-1}) \ , \tag{11}$$

while the initial configuration c^0 is a sequence of cells in the lattice at time $t = 0$.

Formally, a CA C is a tuple:

$$C = \langle A_C, d, \phi_C \rangle \ , \tag{12}$$

and in order to specify its dynamics we shall use the notation $C(c^0)$ for the initial configuration c^0.

For example, a one-dimensional ($d = 1$) CA C with a binary alphabet $A_C = \{0, 1\}$ may use a local update rule ϕ_C defined for a neighbourhood with 3 cells (i.e., $r = 1$), setting dynamic updates as:

$$c_i^t = \phi_C(c_{i-1}^{t-1}, c_i^{t-1}, c_{i+1}^{t-1}) \tag{13}$$

There are $8 = 2^3$ permutations of inputs into the local rule ϕ_C, and consequently, $256 = 2^8$ local rules in total. This type of CA with two possible values for each cell and local update rules defined only on the current state of the cell and its two nearest neighbors is called Elementary Cellular Automata (ECAs). A scheme, known as the Wolfram code, assigns each ECA rule a number from 0 to 255 as follows: the resulting states for each possible input permutation (written in order $111, 110, \ldots, 001, 000$) is interpreted as the binary representation of an integer. For instance, the ordered resulting states $0, 1, 1, 0, 1, 1, 1, 0$ of ϕ_C constitute the rule 110, because the integer 110 has a binary representation of 01101110 [11]. The rule 110 is of particular interest because it is the only one-dimensional CA which has been proven to have the same computational power as a UTM [47], and therefore, can generate undecidable dynamics.

Another well-studied example is *Conway's Game of Life* [48]: a two-dimensional ($d = 2$) CA G with a binary alphabet $A_G = \{0, 1\}$ and a specific local update rule ϕ_G defined for the Moore neighbourhood with 9 cells (i.e., $r = 1$): $\phi_G : A_G^{3^2} \to A_G$, such that

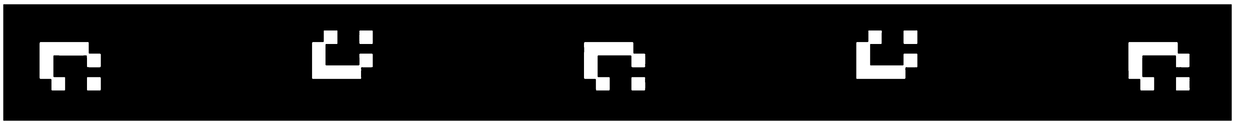

Fig. 1. The Game of Life: gliders ("lightweight spaceships" (LWSS)). Snapshot of dynamics "Life in Life" by Phillip Bradbury: https://www.youtube.com/watch?time_continue=4&v=xP5-iIeKXE8, used under CC BY license.

1. Deaths. Any live cell with fewer than two or more than three live neighbours dies.
2. Survivals. Any live cell with two or three live neighbours lives on to the next generation.
3. Births. Any dead cell with exactly three live neighbours becomes a live cell.

It is well-known that Game of Life is also undecidable, having the same computational power as a universal Turing machine [49].

Both one-dimensional rule 110 and two-dimensional Game of Life produce gliders: coherent spatial patterns that move across the grid replicating their structure (see Fig. 1). It has been demonstrated that gliders fulfill the role of information transfer in distributed computation carried out by CA [50].

2.3.2. Termination condition

As our general purpose is to study analogies and equivalencies between CAs and other systems which compute or prove specific outcomes, we need to adopt a convention determining when the desired output has occurred, i.e., trace the dynamics of the input configuration until some "halting" condition applies [28]. For example, the end of computation may be indicated by reaching an (attractive) fixed-point or by reaching a temporal cycle of length two: this can be determined by comparing configurations at different time steps [51]. Importantly, as pointed out by Sutner [28], this condition must be primitive recursively decidable, but a precise mechanism may vary: for example, a termination condition may check if a particular predesignated cell reaches a special state, or if an arbitrary cell or a set of cells reach a special predefined state(s), or if the configuration is a fixed point or a limit cycle.

Importantly, we distinguish among attractors, i.e. limit cycles (including fixed points which are limit cycles of length 1) c^* by arbitrarily designating some of those as "accepted" and the rest as "rejected" outcomes (to stay closer to the intuition behind the proof of undecidability for TMs presented in section 2.2.2). Illustrating this for fixed points, this can be done by arbitrarily partitioning the set of all configurations Ψ_C into two sets, Ψ_C^+ and $\Psi_C^- = \Psi_C \setminus \Psi_C^+$, so that an attractive fixed-point configuration $c^t \in \Psi_C^+$ can be interpreted as an accepted outcome, and a fixed-point configuration $c^t \in \Psi_C^-$ would correspond to a rejected outcome. This partitioning is formally described by function $\pi_C : \Psi_C \times \Psi_C \times \mathbb{N} \to \{1, 0\}$ such that, at time $t \in \mathbb{N}$, $\pi_C(c^t) = 1$ if and only if $c^t = c^{t-1}$, and $c^t \in \Psi_C^+$, while $\pi_C(c^t) = 0$ if and only if $c^t = c^{t-1}$ and $c^t \in \Psi_C^-$. In order to make concrete this arbitrary partition of the configuration space, we may choose any single cell, e.g., c_{42}^t, then select a specific symbol $\alpha \in A_C$, and then, for a fixed-point c^t, assign $\pi_C(c^t) = 1$ if and only if $c_{42}^t = \alpha$, and $\pi_C(c^t) = 0$ if and only if $c_{42}^t \neq \alpha$.

In demonstrating that rule 110 is computationally equivalent to a UTM, Cook developed a concrete algorithm for compiling a Turing machine showing that the dynamics of rule 110 will eventually produce the bit sequence 01101001101000 if and only if the corresponding Turing machine halts [52]. Such a termination condition can be expressed in terms of temporal rather than spatial sequences: "it is also the case that the sequence 110101010111111 will be produced over time by a single cell if and only if the Turing machine halts" [52]. Specifically, these sequences are produced by a designated glider configuration (glider F), chosen to occur only if the corresponding algorithm halts. Similarly, one may designate appearance of a specific two-dimensional configuration in the Game of Life — glider, still-life (a non-changing pattern), or oscillator (a pattern returning to its original state, in the same orientation and position, after a finite number of generations) — as the "accepted" termination condition. Analogously, another glider, still-life, or oscillator configuration may be chosen to indicate the opposite "rejected" termination outcome. We stress that, in order to achieve the computational equivalence with Turing machines, such termination conditions are necessary to specify in addition to setting the automaton's rule table and an initial configuration.

Therefore, in general, one may extend the definition of a CA C to include a termination condition π_C:

$$C = \langle A_C, d, \phi_C, \pi_C \rangle \tag{14}$$

so that $C(c^0)$ specifies the CA dynamics starting from initial configuration c^0.

The inequality $c^t \neq c^{t-1}$ is always computable. However, due to the finitary nature of all computations, the equality is not decidable in type-2 computability [28] (the framework of Type-2 Theory of Effectivity allows for computability over sets of a cardinality up to continuum [53]), and so there is no guarantee that the termination condition can be effectively checked for any given pair c^t and c^{t-1}, because the lattice is itself infinite. As we shall see in subsection 2.3.3, one may restrict the space of possible CA configurations to certain subspaces within which the termination condition can always be checked in a primitive recursively decidable manner. Henceforth we follow the approach which restricts the space of possible CA configurations to only those subspaces over which a recursively decidable test of termination conditions is possible. As pointed out by Sutner [28], all of these subspaces are closed under the application of a global map Φ_C, ensuring that the dynamics stay within the restricted space.

We will abbreviate the case when a CA C terminates at a configuration $c^t \in \Psi_C^+$ as follows $C : c^0 \to c^+$, and the case terminating at $c^t \in \Psi_C^-$ as $C : c^0 \to c^-$. It is worth pointing out that membership $c^t \in \Psi_C^+$ or $c^t \in \Psi_C^-$ is computable within the restricted subspace of possible CA configurations, i.e., a recursively decidable test of membership is ensured.

Our choice of the distinction between the attractors in Ψ_C^+ and Ψ_C^- as opposite outcomes of the computation carried by the dynamics is somewhat arbitrary. Importantly, any such distinction needs to be encodable into a regular string, for example, the determination that an attractor satisfies the requirement of being effectively computable during CA run-time (i.e., it is intrinsic to CA dynamics), and all that needs to be encoded is the assignment of "accept" or "reject" labels to the chosen binary outcomes.

With such a termination condition it is possible to frame a question on decidability of CA dynamics directly, without tasking an algorithm external to the CA to check whether the CA dynamics do or do not ever reach the given target configuration.

Others have shown that there is a way to simulate a TM M with a one-dimensional CA C, by creating the alphabet A_C as the union of the set of states Q and the tape alphabet Γ of M, and constructing the local update rule ϕ_C out of the transition function μ by smartly interleaving state symbols $q \in Q$ and tape symbols $\gamma \in \Gamma$ [6, p. 121]. For example, the transition resulting in the move of the machine's head to the right corresponds to these two local CA updates by ϕ_C:

$$\text{if } \mu(q_1, \gamma_1) = (q_2, \gamma_2, R), \quad \text{then } \phi_C(*, *, q_1, \gamma_1, *) = \gamma_2 \ \text{ and } \ \phi_C(*, q_1, \gamma_1, *, *) = q_2 \ ,$$

where * matches any state. One may see a parallel here with one-dimensional configurations of TMs 2.2.1. As a result, the computation carried out by a TM, updating over the set of states Q and the tape alphabet Γ, i.e. over $Q \times \Gamma$, can be made equivalent to dynamics of the corresponding automaton which modifies its configurations $c^t \in A_C^{\mathbb{Z}}$. Consequently, there is a correspondence between the combination of the TM's start state $q_0 \in Q$ and its initial tape pattern formed by symbols from Σ, on the one hand, and the initial configuration $c^0 \in A_C^{\mathbb{Z}}$ of the CA, on the other hand.

Finally, the role of the machine's accept and reject states $q_{acc} \in Q$ and $q_{rej} \in Q$ may be played by the termination condition π_C checking whether configurations are attractors in Ψ_C^+ or Ψ_C^-.

2.3.3. *Classifications of Cellular Automata*

The repeated application of a global rule Φ_C, starting from the initial configuration c^0, produces an evolution of configurations c^t over time. In classifying global CA rules according to its long-term asymptotic dynamics, the following qualitative taxonomy is typically employed [54]:

- class I (evolution leads to a homogeneous state);
- class II (evolution leads to periodic configurations);
- class III (evolution leads to chaotic patterns);
- class IV (evolution leads to complex localized structures over long transients).

In other words, class I consists of CAs that, after a finite number of time steps, produce a unique, homogeneous state (analogous to "fixed point" dynamics). Class II contains automata which generate a set of either stable or periodic structures (typically having small periods — analogous to "limit cycle" dynamics) — each region of the final configuration depends only on a finite region of the initial configuration. Class III includes CAs producing aperiodic ("chaotic") spatiotemporal patterns from almost all possible initial states — the effects of changes in the initial configuration almost always propagate forever, and a particular region of the final configuration depends on a region of the initial configuration of an ever-increasing size (analogous to "chaotic attractors"). Class IV includes CAs that generate patterns continuously changing over an unbounded transient, and some of these CAs have been shown to be capable of universal computation [11,47,54].

It is important to distinguish between (i) (possibly undecidable) questions about CA dynamics on all possible initial configurations, and therefore, about the CAs classification, and (ii) (possibly undecidable) questions whether the CA dynamics can ever reach a target configuration for a given initial configuration. An extensive analysis of the classification problem and its undecidability for a broad range of cases has been provided by Sutner [8,28] and others [6,7]. The important insight in dealing with the classification problem is a restriction of the space of possible configurations to certain subspaces, which include, for example, configurations with finite support, or spatially periodic configurations, or almost periodic configuration, or in the most general case, recursive configurations, where a cell state is assigned by a computable function, so that such a restriction produces an "effective dynamical system" [28].

2.3.4. *Universal Cellular Automata*

A universal CA is a CA which can emulate any CA. One of the simplest universal CAs has been shown to be the rule 110 ECA with just 2 states which happen to be sufficient for producing universality in a 1-dimensional CA [47]. A universal CA has the same power as a UTM, and can, therefore, generate undecidable dynamics. For example, whether an initial state will ever reach a quiescent state can be seen as the CAs equivalent of the undecidable Halting Problem [46,55]. The undecidability of CA dynamics and the role played by self-reference will be discussed in subsection 2.3.5, and here we point out several aspects that are important in constructing universal CAs.

First of all, in constructing universal CAs one must derive a way to encode any simulated CA and its initial configuration, as data, in the form that can be used by the universal CA. Without loss of generality, we can assume that such an encoding $[C, c^0]$ can be produced in a primitive recursively decidable way, as one only needs to encode the initial configuration c^0 from the suitably restricted subspace (e.g., recursive configurations) and the local rule ϕ_C defined for finite neighbourhoods. The encoding of a CA which has been extended with a termination condition π_C needs only to include in addition the distinction between attractors in Ψ_C^+ and Ψ_C^-. Such a distinction can be determined by the state of a single designated cell.

Another technique employed in simulating CAs uses the coarse-graining of the CA dynamics, by grouping neighboring cells into a *supercell* according to some specified convention (this essentially follows a renormalization scheme) [46]. A supercell is created by projecting the states of a block of cells of one CA C into a single cell of the coarse-grained CA C'. The update rule $\phi_{C'}$ is constructed from the update of ϕ_C by projecting its arguments and outcome, subject to certain commutativity conditions [46], to the arguments and outcomes defined for supercells. Such a coarse-grained emulation of C may or may not be carried out without loss of relevant dynamic information, but a universal coarse-grained CA C' ensures that all dynamics can be preserved.

An important building block used in constructing universal two-dimensional CAs is a *unit cell*: a rectangular or square subset of the configuration space (e.g., the Game of Life plane) that tiles over the space. In general, a unit cell has a fixed number of distinct patterns, essentially forming a meta-level alphabet — for example, two distinct patterns, the ON and OFF cells, are needed to simulate the binary Game of Life. Each tile can assume one of the patterns, aiming to simulate a cellular automaton in a coarse-grained but fully preserving way. For example, the *Outer Totalistic Cellular Automata metapixel* (*OCTA metapixel*), a 2048 × 2048 unit cell, was designed by Brice Due in 2006 to reproduce the Game of Life and any Life-like CA [56] in a "Life in Life" simulation. The period of OCTA metapixel is 35328 cycles, needed to change between the ON and OFF metapixel states (see Fig. 2 showing emergence of meta-level states during one period). The meta-level ON and OFF cells are particularly easily distinguishable in a simulation of the Game of Life by OCTA metapixel, as shown in Fig. 3.

It is important to point out that the unit cell's states, observed at the meta-level, emerge as a result of the dynamics produced by the underlying CA, and not by any direct interaction between metapixels. That is, the distributed computation itself is still carried out at the underlying level (e.g., the level of the original Game of Life), but the "Life in Life" dynamics, which are recognized with respect to the OCTA metapixels' states, are simulated at the emergent level. The emergence in this case is understood not only as pattern formation, but also in the broader sense related to the efficiency of prediction [57].

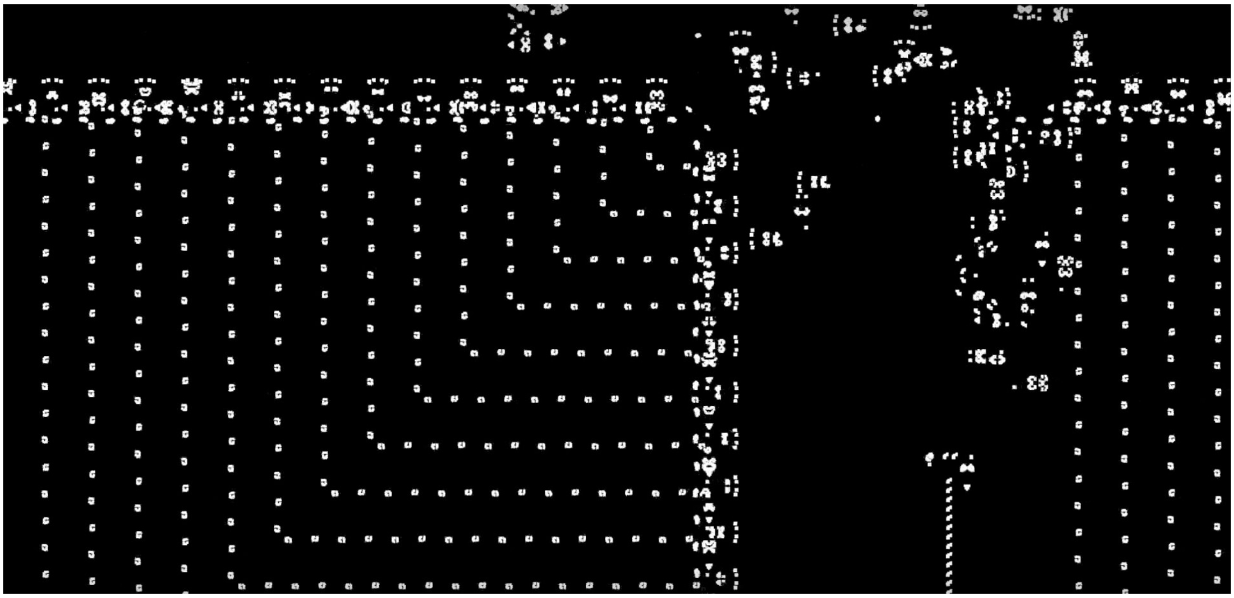

Fig. 2. The Game of Life simulated in OCTA metapixel: emergence of meta-level states within unit cells, which are being filled by a series of gliders formed by the underlying dynamics. Snapshot of dynamics "Life in Life" by Phillip Bradbury: https://www.youtube.com/watch?time_continue=4&v=xP5-iIeKXE8, used under CC BY license.

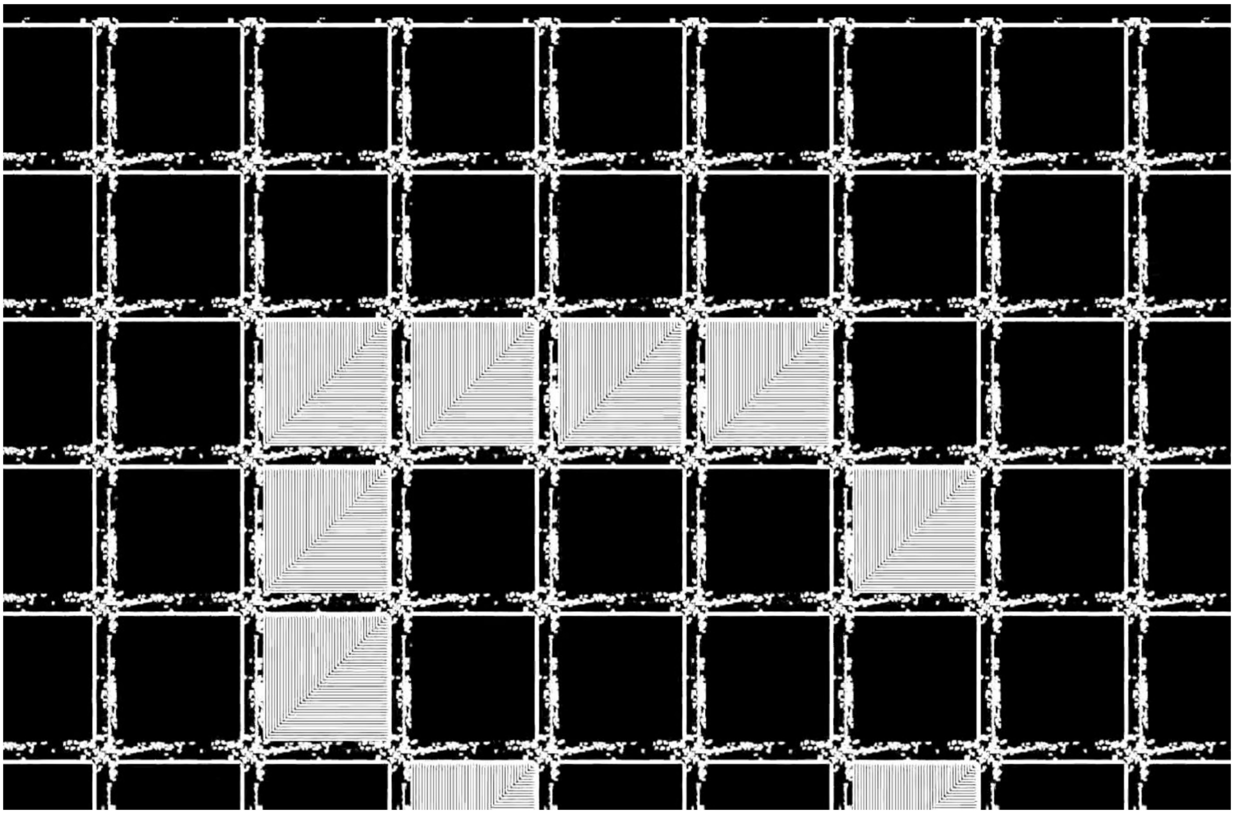

Fig. 3. The Game of Life simulated in OCTA metapixel: emergence of a meta-level LWSS glider configuration. Snapshot of dynamics "Life in Life" by Phillip Bradbury: https://www.youtube.com/watch?time_continue=4&v=xP5-iIeKXE8, used under CC BY license.

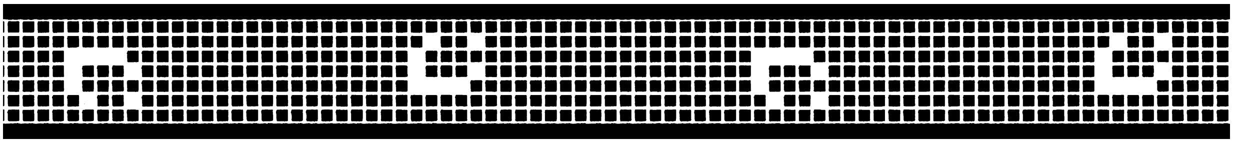

Fig. 4. The Game of Life simulated in OCTA metapixel: emergence of meta-level LWSS gliders. Snapshot of dynamics "Life in Life" by Phillip Bradbury: https://www.youtube.com/watch?time_continue=4&v=xP5-iIeKXE8, used under CC BY license.

The dynamics of the underlying universal CA simulate the "Life in Life" CA, completely reconstructing itself at the meta-level: see, for example, the emerging glider configuration shown by Fig. 3, and a series of gliders shown in Fig. 4. Therefore, any termination condition specified at the underlying level may also be utilized at the meta-level, with respect to the emergent pattern(s) defined in terms of metapixel states.

2.3.5. *Undecidable dynamics*

In this subsection we sketch a proof of the undecidability of CA dynamics, following the steps used in the proof of the undecidability of language A_{TM}, which demonstrated the Halting Problem for TMs, as well as the undecidability of formal systems. The traditional approaches typically establish an equivalence between CAs and TMs *per se*, and thus impute universality and undecidability of CAs based on these properties in TMs. Our purpose is more specific insofar as we aim to explicitly reconstruct the Halting Problem within the undecidable dynamics of CAs, exposing the Liar's Paradox analogy within this framework.

If CA dynamics were decidable, then there would have existed a decider CA with a binary alphabet $P = \langle (A_P = \{0, 1\}), d, \phi_P, \pi_P \rangle$ capable of simulating any other CA $M = \langle A_M, d, \phi_M, \pi_M \rangle$ starting from the initial configuration m^0 (again we note the analogy with the representative predicate P(x) used in the proof of undecidability of formal systems, and the decider TM P). As we have seen, a universal CA with a binary alphabet can be constructed, and it is the decidability of the dynamics created by a universal CA which we shall refute in the proof. The input of decider P is given as $p^0 = [M, m^0]$, while the termination condition π_P are specified in such a way that only two decidable outcomes are possible, being constrained as follows:

$$\begin{cases} P : p^0 \to p^+ & \text{whenever } M : m^0 \to m^+ \\ P : p^0 \to p^- & \text{whenever } M : m^0 \to m^- \text{ or runs forever} \end{cases} \tag{15}$$

In other words, the dynamics of P terminate at some attractor configuration p^t whenever the dynamics of M terminate at some attractor configuration m^t. More importantly, whenever the dynamics of M reach an attractor in the complement set Ψ_M^- or simply run forever, the dynamics of P are assumed to necessarily reach an attractor in the complement set Ψ_P^-. The ability to specify such a definitive termination condition for P is, in fact, the main assumption behind the decidability of CA dynamics, to be refuted by the proof that follows.

The universal CA P that we shall use to illustrate the proof is the "Life in Life" CA, based on the OCTA metapixel. As mentioned already, this CA is universal and the aspect to be refuted is the decidability of the dynamics created by the "Life in Life" CA — in other words, we shall show that this CA is not a decider CA. In doing so, we specify the termination condition π_P for the "Life in Life" CA, set to capture the two decidable outcomes 15, in a way replicating the termination condition π_M of the CA M, but expressed in the alphabet of the CA P. For example, the termination condition may be set with respect to observing specific Game of Life configurations, i.e., if a designated oscillator configuration, F^+, is observed at the meta-level within the lattice configuration c^t, then $\pi_P(c^t) = 1$, while appearance of another specifically chosen oscillator configuration F^-, or the determination that the CA M runs forever, would yield $\pi_P(c^t) = 0$. Since, by the to-be-refuted assumption, P is a decider CA, both of these outcomes must be decidable.

Having assumed that P exists, we construct another inverter CA $V = \langle A_V, d, \phi_V, \pi_V \rangle$, running from the initial configuration $v^0 = [M]$. This intends to match the idea of a Gödel sentence in formal systems, as well as the inverter TM V. Using suitable encoding and decoding in producing $[M, [M]]$ from $[M]$ is the first required step. For example, in simulating "Life in Life", the initial configuration v^0 of CA V must match the initial configuration m^0 of the CA M, and hence, must be encoded in a way ensuring that the initial metapixels form the ON and OFF states identical to the binary states of the initial configuration m^0. Similar to the inverter TM V described by (6), the CA V will simulate M

running on $[M]$. The crucial step in creating the inverter CA is, however, the inversion of the attractor outcomes, so that the termination condition π_V matches the following:

$$\begin{cases} V : [M] \to v^- & \text{whenever } M : [M] \to m^+ \\ V : [M] \to v^+ & \text{whenever } M : [M] \to m^- \text{ or runs forever} \end{cases} \tag{16}$$

It is important to point out that this inversion occurs by simply changing the interpretation of the Game of Life configurations designated to indicate the termination outcomes. Formally, if the designated oscillator configuration F^- is observed at the meta-level within the lattice configuration c^t, or it is determined that the CA M runs forever, then $\pi_V(c^t) = 1$. On the contrary, if the designated oscillator configuration F^+ is observed at the meta-level within the configuration c^t, then $\pi_V(c^t) = 0$. We stress that the inversion of the termination conditions is confined to (re-)setting π_V, outside of the specifications of the CA's rule table and initial configuration. Thus, the "Life in Life" CAs P and V simulate the CA M in exactly the same way, but the interpretations of the observed oscillators F^+ and F^- are inverted in V. We again point out the analogy with the "inside" self-reference in formal systems visible here in the CA M running on an encoding of itself.

Finally, we consider dynamics of the inverter V running with the initial configuration $v^0 = [V]$ (this is, of course, similar to the construction of the "external" self-reference in formal systems), which corresponds to the following constraint, resulting from substituting the elements of V for the elements of M in expression (16):

$$\begin{cases} V : [V] \to v^- & \text{whenever } V : [V] \to v^+ \\ V : [V] \to v^+ & \text{whenever } V : [V] \to v^- \text{ or runs forever} \end{cases} \tag{17}$$

The result is again a contradiction in the style of the Liar's Paradox: the CA V reaches an attractor in the subset Ψ_V^- whenever it reaches an attractor in the complement subset Ψ_V^+. This contradiction shows the impossibility of the existence of a decider CA P, and therefore, the undecidability of CA dynamics. We note that the inverter CA V was running on the input representing its own description $[V]$, while employing the decider CA P "sandwiched" between the self-referencing V and the self-referencing M.

Continuing with the "Life in Life" V example, we can express this contradiction through the meta-level dynamics reaching the configuration that corresponds to the "accepted" outcome, being in Ψ_V^+, but at the underlying level of the CA V itself this configuration indicates the "rejected" outcome, being in Ψ_V^-. This forms a contradiction only because the CA V simulates itself. We must note that a key step leading to the contradiction is the inversion of the termination condition which occurred *outside* of the system *per se*. Thus, it can be argued that this contradiction is empowered not only by the ability to represent programs as data (via suitable encodings) and the ability to design universal CAs, but also by the capacity to assign a negative meaning to the observed configurations. This is, in fact, the same mechanism that was employed in Gödel's proof where the Self-reference lemma was applied to the *negated* provability predicate $\neg\text{Provable}_{\mathcal{F}}(x)$.

We re-iterate that universal CAs are definitely constructable and as we pointed out, the CA rule 110 and "Life in Life" have been shown to be capable of universal computation [47,56]. What is actually impossible is a specification of a definitive termination condition assigning binary outcomes for any possible CA M that is being simulated, as in (15).

3. Results

3.1. Diagonalization and self-reference

To illustrate the diagonalization argument employed in the undecidability proof(s) in various frameworks, we follow the expositions offered by Buldt [32] and Gaifman [43] in the context of formal systems, adapted for our purposes.

In Step 1, the (at most countable) set of all first-order expressions with the free variable x is considered:

$$\mathcal{A} = \{W_0(x), W_1(x), W_2(x), \ldots\}.$$

In Step 2, the set of all of their Gödel numbers is formed:

$$\mathcal{B} = \{\ulcorner W_0(x) \urcorner, \ulcorner W_1(x) \urcorner, \ulcorner W_2(x) \urcorner, \ldots\}.$$

Table 1
First diagonalization (i.e., "internal" self-reference) for a formal system.

	$\ulcorner W_0(x) \urcorner$	$\ulcorner W_1(x) \urcorner$	$\ulcorner W_2(x) \urcorner$	$\cdots$
$W_0(x)$	W_{00}	W_{01}	W_{02}	
$W_1(x)$	W_{10}	W_{11}	W_{12}	$\cdots$
$W_2(x)$	W_{20}	W_{21}	W_{22}	
$\vdots$		$\vdots$		$\ddots$

Table 2
Second diagonalization (i.e., "external" self-reference) for a formal system.

	$\ulcorner W_0(x) \urcorner$	$\ulcorner W_1(x) \urcorner$	$\ulcorner W_2(x) \urcorner$	$\cdots$	$\ulcorner W_k(x) \urcorner$	$\cdots$
$W_0(x)$	W_{00}	W_{01}	W_{02}		W_{0k}	
$W_1(x)$	W_{10}	W_{11}	W_{12}	$\cdots$	W_{1k}	$\cdots$
$W_2(x)$	W_{20}	W_{21}	W_{22}		W_{2k}	
$\vdots$		$\vdots$		$\ddots$		
$W_k(x)$	W_{k0}	W_{k1}	W_{k2}		$W_{kk} = \gamma$	
$\vdots$		$\vdots$				$\ddots$

In Step 3, all members of set $\mathcal{B}$ are used in place of the free variables of all members of the set $\mathcal{A}$. Denoting $W_{ij} = W_i(\ulcorner W_j(x) \urcorner)$, a matrix is constructed as shown in Table 1.

The diagonal sequence $\{W_{jj}\}$ corresponds to the "first diagonalization" (i.e., first, or "internal", self-reference [32,43]).

The next step is to consider the row of the table, with an index k, corresponding to the predicate

$$W_k(x) \equiv \neg\text{Provable}_{\mathcal{F}}(\text{diag}(x)) ,$$

where the term $\text{diag}(x)$ corresponds to a function $diag(x)$ that maps the Gödel number of a wff $W(x)$ to the Gödel number of the self-referential wff $W(\ulcorner W(x) \urcorner)$, that is:

$$diag(\mathcal{G}(W(x))) \equiv \mathcal{G}(W(\ulcorner W(x) \urcorner))$$

and

$$\text{diag}(\ulcorner W(x) \urcorner) = \ulcorner W(\ulcorner W(x) \urcorner) \urcorner .$$

As pointed out by Gaifman, it does not matter how the function $diag(x)$ is defined on numbers that are not Gödel numbers [43]. The elements of the k'th row are formed, as any other elements of the table, by using all members of set $\mathcal{B}$, i.e., the numerals $\ulcorner W_j(x) \urcorner$, in place of the free variable of the predicate $W_k(x)$:

$$W_{kj} = \neg\text{Provable}_{\mathcal{F}}(\ulcorner W_j(\ulcorner W_j(x) \urcorner) \urcorner) .$$

In the style of Cantor's diagonalization method, we can informally say that the k'th row of the table "inverts" the diagonal entities $W_{jj} = W_j(\ulcorner W_j(x) \urcorner)$, by applying $\neg\text{Provable}_{\mathcal{F}}$ to numerals of their Gödel numbers. Importantly, the predicate $W_k(x) \equiv \neg\text{Provable}_{\mathcal{F}}(\text{diag}(x))$ is itself a member of the set $\mathcal{A}$, by construction being distinct from other members $W_j(x)$, see Table 2.

The crux of the argument is the element $W_{kk} = \neg\text{Provable}_{\mathcal{F}}(\ulcorner W_k(\ulcorner W_k(x) \urcorner) \urcorner)$ which was also technically formed, at Step 3 above, as $W_{kk} = W_k(\ulcorner W_k(x) \urcorner)$. Finally we arrive at a Gödel sentence $\gamma = W_k(\ulcorner W_k(x) \urcorner)$ which is neither provable nor disprovable in $\mathcal{F}$, cf. key expressions (1) and (4) re-expressed in terms of γ:

$$\mathcal{F} \vdash \gamma \leftrightarrow \neg\text{Provable}_{\mathcal{F}}(\ulcorner \gamma \urcorner) . \tag{18}$$

Again, in forming the diagonal element W_{kk}, the Gödel sentence γ is self-referencing: this is the second diagonalization [32] or second, "external" use of self-reference [43].

Table 3
The cell i, j is ‘accept’ if M_i accepts $[M_j]$, (or for the CAs: $M_i : [M_j] \to m^+$).

	$[M_1]$	$[M_2]$	$[M_3]$	$\cdots$
M_1	accept		accept	
M_2	accept	accept	accept	$\cdots$
M_3		accept		
$\vdots$		$\vdots$		$\ddots$

Table 4
The cell i, j is the outcome of running P on $[M_i[M_j]]$.

	$[M_1]$	$[M_2]$	$[M_3]$	$\cdots$
M_1	accept	reject	accept	
M_2	accept	accept	accept	$\cdots$
M_3	reject	accept	reject	
$\vdots$		$\vdots$		$\ddots$

The diagonalization argument presented above can be seen almost as a template, including the first diagonalization $W_{ij} = W_i(\ulcorner\ W_j(x)\ \urcorner)$, then the “inversion” $\neg\text{Provable}_{\mathcal{F}}$ applied to numerals of Gödel numbers of the diagonal elements, and finally the second diagonalization where we construct the Gödel sentence $\gamma = W_k(\ulcorner\ W_k(x)\ \urcorner)$ used in expression (18).

Using this template we can now apply the diagonalization argument to show the undecidability of both TMs, (5)–(7), and CAs, (15)–(17).

In the Table 3 the rows correspond to all TMs (or CAs) $M_1, M_2, \ldots, M_j, \ldots$, and the columns correspond to the encodings of these objects $[M_1], [M_2], \ldots, [M_j], \ldots$. Each element of the table is ‘accept’ if the machine accepts the input but is blank if it rejects or loops on that input, cf. expression (5) [44]. In case of CAs, ‘accept’ represents the outcome $M_i : [M_j] \to m^+$, and blank then represents the outcome $M_i : [M_j] \to m^-$ or runs forever, corresponding to expression (15).

The assumption that there exists a decider TM P (or decider CA P) corresponds to “filling” the table with ‘reject’ entries in place of the blanks, as every program-data combination is assumed to be decidable, shown in Table 4 [44]. For example, if M_3 does not accept the input $[M_1]$, the entry $(3, 1)$ is now ‘reject’ because the decider machine or decider CA P rejects the input $[M_3[M_1]]$, cf. expressions (5) and (15).

The diagonal sequence is a result of the first diagonalization (first self-reference), and we can now invert the diagonal elements in order to populate the row representing the inverter TM (CA) V, analogously to the construction of Table 2 for formal systems, and matching the expressions (6) and (16). The result of including the inverter machine (CA) $V = M_k$, for some k, is shown in Table 5, where the element (k, k) is an analogue of the Gödel sentence $\gamma = W_k(\ulcorner\ W_k(x)\ \urcorner)$: the inverter machine (CA) $V = M_k$ runs on $[M_k[M_k]]$, which is the second diagonalization. Neither ‘accept’ nor ‘reject’ in place of the element (k, k) would avoid a logical contradiction. This refutes the assumption of the existence of the decider machine (CA) P.

One instructive comparison is that the decoding and encoding sub-steps used in creating $[M_k[M_k]]$ are analogous to the function $diag(x)$ that maps the Gödel number of a formula $W_k(x)$ to the Gödel number of the self-referential formula $\ulcorner\ W_k(\ulcorner\ W_k(x)\ \urcorner)\ \urcorner$. That is, given $[M_k]$ or $\ulcorner\ W_k(x)\ \urcorner$ one may choose to decode into M_k or $W_k(x)$, and then run the decoded machine (CA) or use the formula $W_k(x)$ on itself, constructing the final self-referential input $[M_k[M_k]]$ or $\ulcorner\ W_k(\ulcorner\ W_k(x)\ \urcorner)\ \urcorner$.

3.2. Comparative analysis

We have considered three computational frameworks (formal systems, Turing machines and Cellular Automata), focussing on self-reference, diagonalization and undecidability manifested on a fundamental level. In this section we offer a detailed comparative analysis across specific structural elements utilized in these frameworks. In doing so, we

Table 5
The cell (k, j) is the outcome of running $V = M_k$ (the inverter of P) on $[M_k[M_j]]$. A contradiction occurs at cell (k, k).

	$[M_1]$	$[M_2]$	$[M_3]$	$\cdots$	$[M_k]$	$\cdots$
M_1	$\underline{\text{accept}}$	reject	accept		accept	
M_2	accept	$\underline{\text{accept}}$	accept	$\cdots$	reject	$\cdots$
M_3	reject	accept	$\underline{\text{reject}}$		reject	
$\vdots$		$\vdots$		$\ddots$		
M_k	reject	reject	accept		$\underline{?}$	
$\vdots$		$\vdots$				$\ddots$

Table 6
State-space comparison across three computational frameworks.

Formal systems	Turing machines	Cellular Automata
alphabet $\mathcal{A}_\mathcal{F}$	alphabets: input Σ and tape Γ	alphabet A_C
symbol strings in $\mathcal{A}^*_\mathcal{F}$	tape strings in Σ^*	configurations in state-space $\Psi_C = A_C^{\mathbb{Z}^d}$
grammar $\langle \mathcal{A}_\mathcal{F}, \mathcal{N}_\mathcal{F}, \mathcal{P}_\mathcal{F}, \mathcal{S}_\mathcal{F} \rangle$	admissible syntax, given Γ, e.g., blank symbol	constraints on state-space Ψ_C, e.g., by recursive configurations
well-formed formula in $\mathcal{A}^*_\mathcal{F}$, restricted by grammar	recognizable tape pattern in Σ^*, given Γ	primitive recursively decidable configuration in restricted subset of Ψ_C
infinite language	infinite tape	infinite lattice

separately analyze different ways to structure the state-space, define the problem, and evolve the system's dynamics, culminating with a comparison of the mechanics of undecidability. While some of these comparisons are well-noted in the literature at a high level [1,2,30], the rest, we believe, reveals the deeper formal analogies unifying the frameworks at a much more detailed level.

3.2.1. State-space

The three computational frameworks that we considered define their state-space in different but analogous terms, and Table 6 explicitly contrasts the corresponding formal descriptions.

The three row elements describing grammar/syntax/restriction must ensure that the well-formed formulas, tape patterns and CA configurations are effectively computable. In formal systems this guarantees that a decision procedure for deciding whether a formula is well-formed or not does exist; the tape patterns of a TM are recognizable; and in CAs an "effective dynamical system" is maintained.

3.2.2. Problem definition and dynamics

The adopted definition of a TM used two final states q_{acc} and q_{rej} to distinguish whether the initial input (which includes a target problem to be solved) is accepted or rejected. To re-iterate, according to this definition, denoted (‡), the initial tape input includes the target to be verified (to be either accepted or rejected), and therefore, the final content of the tape, upon halting at either q_{acc} or q_{rej}, does not matter. As mentioned, an equivalent definition of a TM, denoted (†), may have just one halting state q_{halt}, in which case the target is not included on the tape's initial input, but when the machine halts, the content written on the tape represents the actual output of the computation.

The elements describing axioms and initial inputs/configurations, shown in Table 7, leave some room in the initial conditions to also include the target statement: a theorem (in a formal system), a target to be verified (by a TM), or a target configuration (of a CA extended with a termination condition).

Having considered the elements that define the problem and drive the system's "evolution", that is, the inference process within a formal system, the computation by a TM, or the CA dynamics, we now turn our attention to the mechanics employed by the different proofs of undecidability in our three computational frameworks.

Table 7
Problem definition and inferences/computation/dynamics in three computational frameworks.

Formal systems	Turing machines	Cellular Automata
axioms $X_{\mathcal{F}}$	(a part of) initial tape	(a part of) initial configuration
(‡) target: well-formed formula (wff) to be proven	(‡) target: string as part of initial tape	(‡) target: subset of initial configuration
(‡) proving or disproving a target wff	(‡) final states q_{acc} and q_{rej}	(‡) termination condition testing against Ψ_C^+ or Ψ_C^-
rules of inference $R_{\mathcal{F}}$	transition function μ	local update rule ϕ_C
proof: derivation sequence	sequence of tape patterns and machine states	dynamics: evolution of configurations
(†) an external criterion distinguishing a wff in a proof	(†) final state q_{halt}	(†) termination condition testing for fixed points or limit cycles
(†) theorem: the last wff in a proof	(†) final output written on the tape	(†) the attractor configuration(s)

Table 8
Proving undecidability in three computational frameworks.

Formal systems	Turing machines	Cellular Automata
weakly representative predicate (“a mocking bird”)	universal TM	universal CA
($\nexists$) representative predicate $\mathrm{P}(x)$	($\nexists$) decider UTM P	($\nexists$) universal decider CA P
Gödel number of $W_j(x)$, denoted $\mathcal{G}(W(x))$, such that $\ulcorner W \urcorner = \underline{\mathcal{G}(W)}$	encoding of TM M_j, denoted $[M_j]$	encoding of CA M_j, denoted $[M_j]$
unique decoding of $W(x)$ from Gödel number $\mathcal{G}(W(x))$	unique decoding of TM M from $[M]$	unique decoding of CA M from $[M]$
first diagonalization, internal self-referencing: $W_j(\ulcorner W_j(x) \urcorner)$	first diagonalization, “internal” self-referencing: $M_j[M_j]$	first diagonalization, “internal” self-referencing: $M_j[M_j]$
diagonalization term for $W(x)$: $\mathrm{diag}(\ulcorner W(x) \urcorner) = \ulcorner W(\ulcorner W(x) \urcorner) \urcorner$	compound encoding of TM M, as $[M[M]]$	compound encoding of CA M, as $[M[M]]$
“inverted” predicate $V^{\mathrm{P}}(x) \equiv \neg \mathrm{P}_{\mathcal{F}}(\mathrm{diag}(x))$	inverter TM $V([M])$	inverter CA V running on $[M]$
Gödel sentence $V^{\mathrm{P}}(x) = V(\ulcorner V(x) \urcorner)$	self-referencing inverter TM $V([V])$	inverter CA V running on $[V]$
second diagonalization, external self-referencing: $\mathcal{F} \overset{?}{\vdash} V^{\mathrm{P}} \leftrightarrow \neg \mathrm{P}(\ulcorner V^{\mathrm{P}} \urcorner)$	second diagonalization, external self-referencing: $V([V]) = ?$	second diagonalization, external self-referencing: $V : [V] \rightarrow v^?$
Gödel Incompleteness Theorem, leading to undecidability	The Halting Problem	Undecidable dynamics and the “Edge of Chaos”

3.2.3. *Undecidable dynamics*

Table 8 traces the key steps of the diagonalization argument. The existence of some elements in the table have only been assumed for the purposes of proof by contradiction, and we denote these lines by $\nexists$.

Importantly, each of the proofs ends up in a contradiction. In formal systems, the proof constructs a Gödel sentence which yields a contradiction, expressed as an inability to resolve the question: $\mathcal{F} \overset{?}{\vdash} V^{\mathrm{P}} \leftrightarrow \neg \mathrm{P}(\ulcorner V^{\mathrm{P}} \urcorner)$. This results in the Gödel Incompleteness Theorem that leads to undecidability. In TMs, the contradiction comes from trying to answer the halting question about the inverter machine running on the encoding of itself: $V([V]) = ?$ which is the core issue of The Halting Problem. And in Cellular Automata, the conundrum manifests itself as the question of whether the inverter CA would reach a termination condition if presented with an initial condition that encodes its own description, $V : [V] \rightarrow v^?$. This, in our opinion, captures undecidable dynamics at the “edge of chaos”.

4. Discussion

It is important to point out that in all considered computational frameworks the undecidable “dynamics” are possible even with perfect knowledge of the initial / boundary conditions of the system [3–5]. As mentioned in Introduction, this distinguishes undecidable dynamics from chaotic dynamics. Interestingly, undecidable dynamics can also be distinguished from unprestatable functions and their evolution [58]: when a dynamical system (e.g., a chemical reaction system) alters its own boundary condition, we cannot deduce the actual behavior of the system even from the same initial and boundary conditions. Unprestatable dynamics resulting from such, possibly iterative, modifications of the boundary conditions is also unlike standard chaos. However, the class of systems with dynamically altering boundaries and hence, unprestatable dynamics, is distinct from the systems with undecidable dynamics which evolve from fixed initial conditions.

Therefore, one may be justified in defining a complex system as a dynamical system with at least undecidable dynamics, and possibly unprestatable dynamics.

As has been previously pointed out [59–62], undecidability may be fundamentally related to computational novelty, and so a mechanism producing novelty may need to be capable of universal computation. For example, Markose [62] recently argued that the issue of novelty production and “thinking outside the box” by digital agents must be immediately related to their capacity to encode a Gödel sentence in order to exit from known listable sets (e.g., actions, technologies, phenotypes) and produce new structured objects. This formalism follows Binmore [63] in highlighting the fundamental aspects of novelty generation through the lens of game-theory, and considering a strategic game with adversarial (contrarian) agents which act as the Liar by negating what it can predict or compute [62]. It has also been recently argued that evolutionary strategies in iterated games, in which the same economic interaction is repeatedly played between the same agents, can be seen as processes capable of universal computation [64]. In other words, undecidable dynamics is the necessity for creativity and innovation.

As we have shown, the capacity to generate undecidable dynamics is based upon three underlying factors: (i) the program-data duality; (ii) the potential to access an infinite computational medium; and (iii) the ability to implement negation. It is interesting to note parallels between these principles and Markose’s ingredients for novelty generation by digital agents, underpinned by Gödel – Turing – Post approach [62]: (1) agents can operate on encoded information and store codes; (2) agents can do offline simulations that involve self-referential meta-calculations, i.e., deal with Gödel meta-mathematics; and (3) agents can record negation and, therefore, “can process the logical archetype of the Liar in a fixed point setting”. These considerations emphasize once more the self-referential basis of undecidable dynamics, not only providing foundations for the most general computational frameworks, but also revealing paths for implementing complex adaptive systems.

Conflict of interest statement

Authors declare no competing financial interests.

Author contributions statement

M.P. and F.B. conceived the idea of the paper and carried out initial analysis; M.P., M.H. and P.P. carried out analysis for section 2.1; M.P., M.H., J.L. and F.B. carried out analysis for section 2.2; M.P., M.H. and J.L. carried out analysis for section 2.3; S.K. discussed the role of unprestatable functions vs undecidable dynamics. M.P and M.H. wrote the manuscript. All authors reviewed the manuscript.

References

[1] Casti JL. Chaos, Gödel and truth. In: Casti JL, Karlqvist A, editors. Beyond belief: randomness, prediction, and explanation in science. CRC Press; 1991.
[2] Ilachinski A. Cellular Automata: a discrete universe. Singapore: World Scientific; 2001.
[3] Bennett CH. Undecidable dynamics. Nature 1990;346:606–7.
[4] Moore C. Unpredictability and undecidability in dynamical systems. Phys Rev Lett 1990;64(20):2354–7.
[5] Moore C. Generalized shifts: unpredictability and undecidability in dynamical systems. Nonlinearity 1991;4(2):199–230.
[6] Durand B, Formenti E, Varouchas G. On undecidability of equicontinuity classification for Cellular Automata. In: DMCS. AB of discrete mathematics and theoretical computer science proceedings, DMTCS; 2003. p. 117–28.
[7] Kari J. Decidability and undecidability in Cellular Automata. Int J Gen Syst 2012;41(6):539–54.
[8] Sutner K. Computational classification of Cellular Automata. Int J Gen Syst 2012;41(6):595–607.
[9] Wolfram S. Computation theory of Cellular Automata. Commun Math Phys 1984;96(1):15–57.
[10] Wolfram S. Twenty problems in the theory of Cellular Automata. Phys Scr 1985;1985(T9):170.

[11] Wolfram S. A new kind of science. Champaign, Ilinois, US, United States: Wolfram Media Inc.; 2002.

[12] Langton CG. Computation at the edge of chaos: phase transitions and emergent computation. Physica D 1990;42(1–3):12–37.

[13] Crutchfield JP. The calculi of emergence: computation, dynamics and induction. Physica D 1994;75(1–3):11–54.

[14] Wuensche A. Classifying Cellular Automata automatically: finding gliders, filtering, and relating space–time patterns, attractor basins, and the Z parameter. Complexity 1999;4(3):47–66.

[15] Hordijk W, Shalizi CR, Crutchfield JP. Upper bound on the products of particle interactions in Cellular Automata. Physica D 2001;154(3–4):240–58.

[16] Shalizi CR, Haslinger R, Rouquier J-B, Klinkner KL, Moore C. Automatic filters for the detection of coherent structure in spatiotemporal systems. Phys Rev E 2006;73(3):036104.

[17] Lizier JT, Prokopenko M, Zomaya AY. Local information transfer as a spatiotemporal filter for complex systems. Phys Rev E 2008;77(2):026110.

[18] Lizier JT, Prokopenko M, Zomaya AY. Local measures of information storage in complex distributed computation. Inf Sci 2012;208:39–54. https://doi.org/10.1016/j.ins.2012.04.016.

[19] Packard NH. Adaptation toward the edge of chaos. In: Kelso JAS, Mandell AJ, Shlesinger MF, editors. Dynamic patterns in complex systems. World Scientific; 1988. p. 293–301.

[20] J. P. Crutchfield, K. Young, Computation at the onset of chaos, in: The Santa Fe Institute, Westview, Press, 1988. pp. 223–269.

[21] Mitchell M, Crutchfield JP, Hraber PT. Dynamics, computation, and the "edge of chaos": a re-examination. In: Cowan G, Pines D, Melzner D, editors. Complexity: metaphors, models, and reality. Santa Fe Institute Studies in the Sciences of Complexity, vol. 19. Reading, MA: Addison-Wesley; 1994. p. 497–513.

[22] Lizier JT, Prokopenko M, Zomaya AY. The information dynamics of phase transitions in random Boolean networks. In: Bullock S, Noble J, Watson R, Bedau MA, editors. Proceedings of the eleventh international conference on the simulation and synthesis of living systems (ALife XI). Cambridge, MA: MIT Press; 2008. p. 374–81.

[23] Lizier JT, Pritam S, Prokopenko M. Information dynamics in small-world Boolean networks. Artif Life 2011;17(4):293–314. https://doi.org/10.1162/artl_a_00040.

[24] Boedecker J, Obst O, Lizier Mayer JT, Asada M. Information processing in echo state networks at the edge of chaos. Theory Biosci 2012;131(3):205–13.

[25] Siegelmann HT. Neural networks and analog computation: beyond the Turing limit. Cambridge, MA, USA: Birkhauser Boston Inc.; 1999.

[26] Delvenne J-C, Kůrka P, Blondel V. Decidability and universality in symbolic dynamical systems. Fundam Inform 2006;74(4):463–90.

[27] Martínez GJ, Mora JCST, Zenil H. Computation and universality: class IV versus class III Cellular Automata. J Cell Autom 2012;7(5–6):393–430.

[28] Sutner K. Cellular Automata, classification of artificial intelligence. In: Meyers RA, editor. Computational complexity: theory, techniques, and applications. Springer; 2012. p. 312–24.

[29] Hyötyniemi H. On the universality and undecidability in dynamic systems. Technical Report 133, Control Engineering Laboratory, Helsinki University of Technology; 2002.

[30] Ali SM. The concept of poiēsis and its application in a Heideggerian critique of computationally emergent artificiality. Ph.D. thesis, Department of Electrical & Electronic Engineering, Brunel University; 1999.

[31] Smullyan RM. Theory of formal systems. Princeton University Press; 1961.

[32] Buldt B. On fixed points, diagonalization, and self-reference. In: Freitag W, Rott H, Sturm H, Zinke A, editors. Von Rang und Namen: essays in honour of Wolfgang Spohn. Münster: Mentis; 2016. p. 47–63.

[33] Rapaport WJ. Philosophy of computer science: an introductory course. Teach Philos 2005;28(4):319–41.

[34] Chomsky N. Three models for the description of language. IRE Trans Inf Theory 1956;2(3):113–24.

[35] Nagel E, Newman JR. Gödel's proof. New York: New York University Press; 2001.

[36] Raatikainen P. Gödel's Incompleteness Theorems. In: Zalta EN, editor. The Stanford encyclopedia of philosophy. Spring 2015 edition. Metaphysics Research Lab, Stanford University; 2015.

[37] Gaifman H. Naming and diagonalization, from Cantor to Gödel to Kleene. Log J IGPL 2006;14(5):709–28.

[38] Smullyan RM. Fixed points and self-reference. Int J Math Math Sci 1984;7(2):283–9.

[39] Carnap R. Logische Syntax der Sprache. Schriften zur wissenschaftlichen Weltauffassung. In: English: the logical syntax of language. London: Routledge and Kegan Paul; 1934. 1937, 1971 printing.

[40] A. Tarski, Der Wahrheitsbegriff in den formalisierten Sprachen (1936). In English: the concept of truth in formal systems, in: Logic, semantics, metamathematics: papers from 1923 to 1938 / by A. Tarski. Translated from various languages by J.H. Woodger, Clarendon Press Oxford, 1956.

[41] Sieg W, Field C. Automated search for Gödel's proofs. Ann Pure Appl Log 2005;133(1–3):319–38.

[42] Gödel K. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I (1931). In: Feferman S, editor. Collected works. Vol. 1, Publications 1929–1936. Oxford University Press; 1986.

[43] Gaifman H. The easy way to Gödel's proof and related matters. http://www.columbia.edu/~hg17/Inc07-chap0.pdf, 2007.

[44] Sipser M. Introduction to the theory of computation. 1st edition. Thomson Publishing: International; 1996.

[45] Hopcroft JE, Ullman JD. Formal languages and their relation to automata. Reading, Massachusetts: Addison-Wesley Publishing Company; 1969.

[46] Israeli N, Goldenfeld N. Coarse-graining of Cellular Automata, emergence, and the predictability of complex systems. Phys Rev E 2006;73(2).

[47] Cook M. Universality in elementary Cellular Automata. Complex Syst 2004;15(1):1–40.

[48] Gardner M. Mathematical games: the fantastic combinations of John Conway's new solitaire game "life". Sci Am 1970;223:120–3.

[49] Berlekamp ER, Conway JH, Guy RK. What is life? Winning ways for your mathematical plays, vol. 2. London: Academic Press; 1982.

[50] Lizier JT, Prokopenko M, Zomaya AY. Coherent information structure in complex computation. Theory Biosci 2012;131:193–203. https://doi.org/10.1007/s12064-011-0145-9.

[51] Lindgren K, Nordahl MG. Universal computation in simple one-dimensional Cellular Automata. Complex Syst 1990;4(3):299–318.

[52] Cook M. A concrete view of rule 110 computation. In: Neary T, Woods D, Seda AK, Murphy N, editors. The complexity of simple programs. EPTCS, vol. 1. 2008. p. 31–55.

[53] Weihrauch K. Computable analysis: an introduction. Texts in Theoretical Computer Science. An EATCS Series. Springer; 2000.

[54] Wolfram S. Universality and complexity in Cellular Automata. Physica D 1984;10:1–35.

[55] Brady AH. The busy beaver game and the meaning of life. In: Herken R, editor. A half-century survey on the universal Turing machine. New York, NY, USA: Oxford University Press, Inc.; 1988. p. 259–77.

[56] Wikipedia contributors, http://www.conwaylife.com/w/index.php?title=OTCA_metapixelOTCA metapixel, [Online; accessed 4-November-2018] (2018). http://www.conwaylife.com/w/index.php?title=OTCA_metapixel.

[57] Prokopenko M, Boschietti F, Ryan AJ. An information-theoretic primer on complexity, self-organization, and emergence. Complexity 2009;15(1):11–28.

[58] Kauffman S. Humanity in a creative universe. New York, NY, USA: Oxford University Press; 2016.

[59] Casti JL. Complexification: explaining a paradoxical world through the science of surprise. New York, USA: Harper Collins; 1994.

[60] Markose SM. Novelty in complex adaptive systems (CAS) dynamics: a computational theory of actor innovation. Phys A, Stat Mech Appl 2004;344(1):41–9.

[61] Prokopenko M. Grand challenges for computational intelligence. Front Robot AI 2014;1:2.

[62] Markose SM. Complex type 4 structure changing dynamics of digital agents: nash equilibria of a game with arms race in innovations. J Dyn Games 2017;4(3):255–84.

[63] Binmore K. Modeling rational players: part I. Econ Philos 1987;3(2):179–214.

[64] Harré M. Utility, revealed preferences theory, and strategic ambiguity in iterated games. Entropy 2017;19(5):201.