cryptography

MDPI

Article

How Hard Is It to Detect Surveillance? A Formal Study of Panopticons and Their Detectability Problem

Vasiliki Liagkou [1,2,*], Panayotis E. Nastou [3,4], Paul Spirakis [5,6] and Yannis C. Stamatiou [1,7]

[1] Computer Technology Institute and Press-"Diophantus", University of Patras Campus, 26504 Patras, Greece
[2] Department of Informatics and Telecommunications, University of Ioannina, 47100 Kostakioi Arta, Greece;
[3] Applied Mathematics and Mathematical Modeling Laboratory, Department of Mathematics, University of the Aegean, 83200 Samos, Greece
[4] Center for Applied Optimization, University of Florida, Gainesville, FL 32611, USA
[5] Department of Computer Science, University of Liverpool, Liverpool L69 3BX, UK; p.spirakis@liverpool.ac.uk
[6] Department of Computer Engineering and Informatics, University of Patras, 26504 Patras, Greece
[7] Department of Business Administration, University of Patras, 26504 Patras, Greece
* Correspondence: liagkou@cti.gr

Abstract: The *Panopticon* (which means "watcher of everything") is a well-known prison structure of continuous surveillance and discipline studied by Bentham in 1785. Today, where persistent, massive scale, surveillance is immensely facilitated by new technologies, the term Panopticon vaguely characterizes institutions with a power to acquire and process, undetectably, personal information. In this paper we propose a theoretical framework for studying Panopticons and their detectability status. We show, based on the *Theory of Computation*, that detecting Panopticons, modelled either as a simple Turing Machine or as an Oracle Turing Machine, is an undecidable problem. Furthermore, we show that for each sufficiently expressive formal system, we can effectively construct a Turing Machine for which it is impossible to prove, within the formal system, its Panopticon status. Finally, we discuss how Panopticons can be physically detected by the heat they dissipate each time they acquire, effortlessly, information in the form of an oracle and we investigate their detectability status with respect to a more powerful computational model than classical Turing Machines, the Infinite Time Turing Machines (ITTMs).

Keywords: formal methods; security; privacy; undecidability; panopticon; turing machine; oracle computations; irreversible computations; Infinite Time Turing Machines (ITMs)

Citation: Liagkou, V.; Nastou, P.E.; Spirakis, P.; Stamatiou, Y.C. How Hard Is It to Detect Surveillance? A Formal Study of Panopticons and Their Detectability Problem. *Cryptography* **2022**, *6*, 42. https://doi.org/10.3390/cryptography6030042

Academic Editor: Shay Gueron

Received: 30 April 2022
Accepted: 10 August 2022
Published: 20 August 2022

1. Introduction

In 1785, the English philosopher and social theorist Jeremy Bentham (see [1]) described an unprecedented institutional punishment establishment, the *Panopticon*. The architecture of this establishment consisted of a circular building dominated by an "observation tower" in the center of which a single guard was continuously watching the inmates, imprisoned in cells arranged around the circular building. Moreover, the prisoners, themselves, could never be able to see the inspector, who remained for ever "invisible" to them. In the 70s, Foucault studied, deeply, Bentham's concepts, pointing to the Panopticon as a generic *power* model impacting people's everyday life. In his own words: "a figure of political technology that may and must be detached from any specific use" [2].

Nowadays, technological achievements have given rise to new, non-physical (unlike prisons), means of constant surveillance that transcend physical boundaries. This fact necessitates a revision of the Panopticon concept to reconcile it with the current technological progress and its, virtually unlimited, surveillance potential. Thus, the Panopticon behaviour is now manifested in the countless Internet services that accumulate people's personal information, in the ubiquitous portable devices that contain massive and often

sensitive information about their owners as well as the Internet giants to which billions of people trust their digital data.

Based on this observation, in this paper we investigate the question of detecting *Panopticons*, that is malicious entities whose mission is massive surveillance and covert information acquisition. Our approach follows the paradigm of the pioneering work of Cohen (see [3,4]) in *computer viruses* and, in general, *computer malware*. The breakthrough idea of Cohen was to model the *vague*, then, concept of a "computer virus" in a *formal* way so that the question of whether computer viruses can be detected *algorithmically*, i.e., by automated detection tools, can be posed *precisely* and *unambiguously* so that it is amenable to a formal analysis and resolution. Cohen modelled computer viruses using the most general, formal, model for an entity whose behaviour is determined through the execution of a *sequence* of elementary, or mechanical, steps: the *Turing Machine* (proposed in the pioneering work of Alan Turing in [5,6]). This idea, not only allowed a formal and, thus, unambiguous, definition of what computer viruses are but allowed Cohen to deploy the rich theoretical framework of the *Theory of Computation* for the analysis of whether computer viruses can be detected algorithmically or not (see, e.g., [7] for a comprehensive treatment of Turing Machines and the fundamental results of the Theory of Computation). To the best of our knowledge, our work is the first one which targets a realistic, formal, definition of the vague concept of *Panopticons*, as well as the formal analysis of their detectability properties based on the Turing Machine formalism and the fundamental results of the Theory of Computation. In other words, although all understand, intuitively, what a Panopticon is, i.e., a massive surveillance entity, there is no, to the best of our knowledge, formal definition of Panopticons which transforms this intuition into a formal framework for the definition and analysis of their properties. Our work is a first step towards this direction.

Therefore, along the same lines of thought as Cohen, in this paper we investigate the problem of whether Panopticons can be, algorithmically, detected. We formally model Panopticons, as Cohen did with computer viruses, as special types of *Turing Machines* and we analyze their detectability properties using results from the Theory of Computation. More specifically, we provide two different, but not unrealistic, Turing Machine based theoretical models of a Panopticon and show that there is no algorithm, i.e., no systematic procedure, that can detect all Panopticons that fall under these two definitions. In other words, detecting Panopticons, at least the ones that fall under these two plausible definitions, is an *undecidable* problem, in principle.

More specifically, the first formal model we examine studies Panopticons whose Panopticon properties are manifested through the *execution* of states (in the Turing Machine terminology) or *actions* (e.g., specific programming instructions such as reading from a disk file) that belong to a specific set of states that characterizes Panopticons of this type. This model is based on the formal computer virus model proposed by Cohen in [3,4]. In some sense, since the focal point of this model is the *execution* of states or actions of a particular type, the model captures the *visible behaviour* of the Panopticon, according to the *actions* it performs, and, thus we call this model *behavioural*.

The second formal model focuses on the *impact* or *consequences* of the actions of the Panopticon and not the actions themselves. In particular, this model captures an essential characteristic of Panopticons, that of acquiring, rather, *effortlessly* information through *surveillance* and *eavesdropping*. We model this characteristic using *Oracle Turing Machines*. This concept was proposed by Turing in [5,6] to investigate the undecidability status of problems whenever they receive *free* information (in the form of an oracle) about some other undecidable problems. The *Oracle*, in the Panopticon context, models information acquired "for free" based on surveillance (observations) and eavesdropping actions, without requiring computational effort. This model is, in some sense, based on the information that a Panopticon deduces using "free" information and, thus, we call it *deductive*. Essentially, this model focuses on the *semantics* of a Turing Machine, i.e., outcomes of operation, while the first model focuses on the *syntax*, i.e., definition, of a Turing Machine. We extend a particular result from [8] (also proved in [7]) in order to show that this class of Panopticons

can, not only cannot be detected, in general, but belong to a higher undecidability class in the hierarchy of undecidable problems which has important consequence for restricted types of Panopticons. For instance, restricted (in some sense) Panopticons of the first formal model we examine are detectable while similarly restricted types of Panopticons of the second formal model remain, still, undetectable.

Furthermore, based on a fundamental result from [9], we show that for any formal system, we can construct a Turing Machine whose Panopticon status, under the second formal model, cannot be proved within the formal system. That is, no proof can be produced by the formal system that this Turing Machine is a *Panopticon* and no proof that it is not a *Panopticon*. In other words, given any formal system, one can provide a procedure that generates a Turing Machine for which it is *impossible* to decide whether it is a Panopticon or not within the formal system.

In some final, rather philosophical, considerations we discuss, based on the exposition in [10], how Panopticons' power of obtaining information *instantaneously* and *effortlessly* merely by observing, renders them *detectable* through the heat dissipated in their environment as a result of the irreversible action of obtaining this information as an oracle reply without performing any computational steps. Finally, we show how oracle-based Panopticons are easy to be detected using *hypercomputation* based on Infinite Time Turing Machines (ITTMs) based on concepts and results from [11]. Although hypercomputation is considered *unrealistic*, the concept can, nevertheless, provide an interesting context within the super-power Panopticons can be compared with super-power Turing Machines, studying how well super-power TMs fare against super-power Panopticons.

2. Definitions and Notation

Since our theoretical model for the Panopticon considers it as a computational entity which is what actually happens in practice in active massive surveillance based on information technologies, we briefly state the relevant definitions and notation from the computation theory that will be used in the subsequent sections (see [7]).

Definition 1 (Turing Machine(s))**.** *A Turing machine is a septuple*

$$M{=}(Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where Q is a finite set of normal operation states, Γ is a finite set called the tape alphabet, where Γ contains a special symbol B that represents a blank, Σ is a subset of $\Gamma - \{B\}$ called the input alphabet, δ is a partial function from $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R\}$ called the transition function, $q_0 \in Q$ is a distinguished state called the start state, $F \subset Q$ is a set of final states.

Notation-wise, given M we denote by $< M >$ its *code*, i.e., an encoding of its description elements as stated in Definition 1 using any fixed alphabet, usually the alphabet $\{0, 1\}$ (binary system). The details can be found in, e.g., [7,12] but they are inessential for our arguments.

One of the main outcomes of Turing's pioneering work [5] was that there exist problems that Turing Machine(s) cannot solve. The first, such, problem was the, so called, *Halting problem* (see, also, [13] for an excellent historic account):

The Halting Problem
Input: A string $x =< M, w >$ which is actually the encoding (description) of a Turing machine <M> and its input w.
Output: If the input Turing M machine halts on w, output True. Otherwise, output False.

The language corresponding to the Halting problem is $L_u = \{< M, w > | w \in L(M)\}$. In other words, the language L_u contains all possible *Turing machine-input* pair encodings $< M, w >$ such that w is accepted by M. This is why L_u is also called *universal language* since the problem of deciding whether a given Turing machine M accepts a given input w

is equivalent to deciding whether $< M, w > \in L_u$. The language L_u was the first language proved to be non-recursive or undecidable by Turing in his seminal work [5].

In order to discuss Panopticons, we need an important variant of Turing Machine(s), called *oracle* Turing Machines. Such a machine has a special tape on which it can write queries to which they obtain the answer instantaneously in one step, no matter what query it is. This type of Turing Machines was, first, discussed, briefly, by Turing himself in [6] under the name *O-machine*. Post developed further this concept in a series of papers [14–16] and his collaboration with Kleene in [17] resulted to the definition that is used today in the Theory of Computation.

Below, we give a formal definition of an Oracle Turing Machine:

Definition 2 (Oracle Turing Machine). *Let A be a language, $A \subseteq \Sigma^*$. A Turing machine with oracle A is a single-tape Turing machine with three special states $q_?, q_y$ and q_n. The special state $q_?$ is used to ask whether a string is in the set A. When the Turing machine enters state $q_?$ it requests an answer to the question: "Is the string of non-blank symbols to the right of the tape head in A?'' The answer is provided by having the state of the Turing machine change on the next move to one of the states q_y or q_n. The computation proceeds normally until the next time $q_?$ is reached, at which point the Turing machine requests another answer from the oracle.*

With respect to notation, we denote by M^A the Turing machine M with oracle A. Also, a set (language) L is recursive with respect to A if $L = L(M^A)$ for some Turing machine M^A that *always* halts while two oracle sets (languages) are called *equivalent* if each of them is recursive in the other (see [7]).

3. The Panopticon Detection Problem and Our Approach

Formal proofs about the impossibility of detecting malicious computation-based entities already exist for a long time for a very important category of such entities, the *computer viruses* or *malware* in general.

In Cohen's pioneering work (see [3,4]) a natural, *formal*, definition of a virus is provided based on Turing Machine(s). Specifically, Cohen defined a virus to be a Turing machine that simply injects its transition function into other Turing Machine(s)' transition functions (see Definition 1) replicating, thus, itself indefinitely. Then, he proves that L_u reduces to the problem of deciding whether a given Turing Machine behaves in this way proving that detecting viruses is an undecidable problem.

Following Cohen's paradigm, we will propose two definitions that precisely describe the operation of a computation-based Panopticon system. In the context of the first model, a Panopticon is a Turing machine that when executed will *demonstrate* a specific, recognizable, behaviour particular to Panopticons manifested by the *execution* of a sequence of actions. For instance, it will publish secret information about an entity, it will download information illegally etc., that is it will take actions that are reflected by *actually reaching*, during its operation, particular states in a set Q_{pan} which contains malware behaviour states.

However, an objection to this rationale can be posed on the grounds that one can scan the transition function (i.e., program code) of a Turing Machine and if a malware state is detected on the right-hand side of δ then the Turing Machine can be declared as *malware* or *Panopticon* in our case. That is, one can propose that the mere existence of malware states, but not their *actual* execution, is sufficient to declare a particular Turing Machine as malware or Panopticon. There, is, however a plausible reply to this objection. First of all, the set Q_{pan} may be *unknown* and, thus, no particular state of the Turing Machine can be identified as belonging in this (unknown) set of states. In other words, it may not be known which subset of the states of a Turing Machine description actually belong in Q_{pan} until they are *invoked* and manifest, themselves, as malicious *actions* through their execution and actual negative *impact* on the Turing Machine's environment. On the other hand, let us assume that Q_{pan} *is* known. Then one can now, indeed, locate a Panopticon state (i.e., a state in the *known*, set Q_{pan}) in the right-hand side of δ of a Turing Machine, if it exists.

However, this Turing Machine *is not known* whether it will *ever* operate as a Panopticon and perform the threatening actions represented by the states in Q_{pan}. The argument of simply locating states in Q_{pan} to detect Panopticons is similar to an argument of declaring a Turing machine as halting simply because on the right-hand side of its transition function it contains a halting state. To be characterized as "halting", the Turing Machine must, actually, *invoke* a halting state.

Now, based on the discussion above, we define formally the *behavioural Panopticons*:

Definition 3 (Behavioural Panopticons)**.** *A Behavioural Panopticon is an octuple*

$$M{=}(Q, Q_{\text{pan}}, \Sigma, \Gamma, \delta, q_0, B, F)$$

where Q is a finite set of normal operation states, Γ is a finite set called the tape alphabet, where Γ contains a special symbol B that represents a blank, Σ is a subset of $\Gamma - \{B\}$ called the input alphabet, δ is a partial function from $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R\}$ called the transition function, $q_0 \in Q$ is a distinguished state called the start state, $F \subset Q$ is a set of final states, and $Q_{\text{pan}} \subset Q$, $Q_{\text{pan}} \cap F = \emptyset$, is a distinguished set of states linked to Panopticon behaviour. We assume that transitions from states in Q_{pan} do not change the Turing Machine's tape contents, i.e., they are purely interactions with the external environment of the Turing Machine and can affect only the environment.

This is much like Cohen's definition of a virus since it characterizes Panopticons according to their *displayed* or *manifested* behaviour. We stress the word *interactions* in order to preclude situations where a false alarm is raised for "Panopticons" which merely list actions that are characteristic of Panopticon behaviour without *ever* actually invoking them during their operation. Instead, they operate normally without any actions taking place that manifest Panopticon behaviour.

Beyond displayed behaviour, Panopticons can be reasonably assumed to also possess *deductive* powers, not directly visible or measurable. In other words, one type of such Panopticons may operate by gathering or computing totally new information, distinct from the information already known to it. We model this type of Panopticon with the language S_1' defined later in this Section. Moreover, another type of Panopticons can take advantage of *easily* acquired, or even *stolen*, freely provided (in some sense) information. In other words, based on information the Panopticon acquires for free by eavesdropping or data breaching, for instance, it deduces further information, perhaps expending some computational effort this time. We model the characteristic Panopticon action, i.e., *observation*, *surveillance* and *illicit information gathering*, using *oracle Turing Machines*, where the freely acquired or eavesdropped information for particular targets is modelled by the oracle set of the machine. Based on this information, the Turing machine deduces, through its normal computation steps, *further* information about its targets. This type of Panopticon is modelled with the language S_2' defined later in this Section. Below, we describe both types of Panopticons, the ones based on S_1' and the ones based on S_2' since their common characteristic is the *deduction* of new information from already known information.

Definition 4. *(Deductive Panopticons) A Deductive Panopticon is a Turing Machine that either by itself (language S_1') or based on observed or stolen information acquired without expending any computational effort (language S_2'), deduces (perhaps with computational effort) further information about entities.*

In the definition above, the Panopticon operating by itself, i.e., without oracles (language S_1'), is weaker (as we will show in what follows) than the one with oracles (language S_2') since the latter is allowed to obtain free advice or information, in the form of an oracle.

Naturally, many other deductive Panopticon definitions would be reasonable or realistic. Our main motivation behind the ones stated above was a balance of theoretical simplicity and plausibility in order to spark interest on the study on formal properties of Panopticons as well as the difficulty of detecting them algorithmically.

Based on the two Panopticon definitions we gave above, we can define the corresponding Panopticon detection problems. The aim of a Panopticon detection algorithm or Turing machine, is to take as input the encoding of another Turing machine and decide whether it is Panopticon or not based on the formal definition.

The Deductive Panopticon Detection Problem 1
Input: A description of a Turing machine (program).
Output: If the input Turing machine operates like a Panopticon according to Definition 3 output True. Otherwise, output False.

More formally, if by L_b we denote the language consisting of Turing machine encodings $< M >$ which are Panopticons according to Definition 3, then we want to decide L_b, i.e., to design a Turing machine that, given $< M >$, decides whether $< M >$ belongs in L_b or not.

The Deductive Panopticon Detection Problem 2
Input: A description of a Turing machine (program).
Output: If the input Turing machine operates like a Panopticon according to Definition 4 output True. Otherwise, output False.

Essentially, the deductive Panopticon detection problem 2 asks to decide the languages S_1' and S_2'.

Our approach is different for each of the two Panopticon models we propose since they are of a different nature, i.e syntactic (for the behavioural model) vs. semantic (for the deductive model). For the behavioural model, we provide a simple adaptation of Cohen's pioneering formal model of a *virus* and prove a Panopticon detection impossibility result much like Cohen's result for virus detection. For the deductive model, we follow a completely different approach using Oracle Turing Machines and a technique that can be applied to prove undecidabililty results for this type of machines.

More specifically, in Chapter 8 of [7] a technique from [8] is presented that establishes a hierarchy of undecidable problems for Oracle Turing Machines. In particular, the technique targets the oracle set $S_1=\{<M> \,|L(M)=\emptyset\}$, with $<M>$ denoting the encoding of Turing machine M. Then, the sets $S_{i+1}=\left\{<M> \,|L(M^{S_i})=\emptyset\right\}$ can be, recursively, defined and the following can be proved (see [7,8]):

Theorem 1. *The membership problem for TM's without oracles is equivalent to S_1 (i.e., L_u is equivalent to S_1).*

Theorem 2. *The problem of deciding whether $L(M) = \Sigma^*$ is equivalent to*

$$S_2=\left\{<M> \,|L(M^{S_1})=\emptyset\right\}.$$

Our first contribution is to propose a plausible Panopticon model which incorporates the *information deduction* characteristic of a Panopticon (see Definition 4). Information deduction takes place whenever the Turing machine under scrutiny produces a *completely new* information set given a set of fixed, finitely many, already *known* information sets. This set models the information that the Panopticon *already knows* through surveillance and observation, without (usually) expending considerable effort since it, merely, intercepts or eavesdrops information.

More formally, let $N_i = \left\{L_1^i, L_2^i, \ldots, L_k^i\right\}$ be a set of recursively enumerable languages, for some fixed integer $k \geq 1$, such that $\emptyset \notin N_i$ for all i. Also, let $M_1^i, M_2^i, \ldots, M_k^i$ the Turing Machine(s) that, correspondingly, accept these languages. These Turing Machine(s) and their corresponding languages model the fixed information sets already known to the Panopticon. We, also, say that a set is *disjoint* from a collection of sets

if it is disjoint from all the sets in the collection. We will, now, define the oracle set $S'_1 = \{< M > | L(M) \text{ is disjoint from } N_1\}$, with $< M >$ denoting the encoding of Turing machine M, and, recursively, in analogy with [7,8], the sets $S'_{i+1} = \left\{< M > | L(M^{S'_i}) \text{ is disjoint from } N_{i+1}\right\}$. The sets S'_1 and

$$S'_2 = \left\{< M > | L(M^{S'_1}) \text{ is disjoint from } N_2\right\}$$

in particular, are central to our approach.

Based on this framework, in Section 5 we prove two theorems analogous to Theorems 1 and 2 on the undecidability of the problem of detecting a deductive Panopticon. Theorem 4 is focused on the weaker form of the deductive Panopticons, related to the set S'_1, while the more powerful one, based on oracle computation for "free" information gathering, related to the set S'_2, is handled by Theorem 5. In particular, in Theorem 4 we prove that L_u is equivalent to S'_1 and in Theorem 5 we prove that the problem of whether $L(M) = \Sigma^*$ is equivalent to S'_2.

Finally, in Section 6 we show that for any sufficiently expressive formal system $\mathcal{F}$, such as Set Theory, we can effectively construct a Turing Machine which is impossible to classify it as a Panopticon or non-Panopticon within $\mathcal{F}$. In other words no formal system is powerful enough so that given any Turing Machine, it can provide either a proof that it is a Panopticon or a proof that it is not a Panopticon.

Before continuing, we should remark that the essential element of the proposed definition of deductive Panopticons is that the oracle consultations model the "effortless", through surveillance, interception or eavesdropping, information gathering by Internet surveillance agencies and organizations. In this context, the sets S'_{i+1} define an infinite *hierarchy* of deductive Panopticons in which a Panopticon whose accepted language belongs in S'_{i+1} operates by consulting a (weaker) lower-level Panopticon whose language belongs in S'_i, with the weakest Panopticons being the ones whose accepted languages belong in S'_1. These last level Panopticons do not have oracle consultations or effortless information gathering capabilities.

4. Impossibility of Detecting Behavioural Panopticons

We will show below that L_u is recursive in L_b. This implies that if we had a decision procedure for L_b then this procedure could also be used for deciding L_u which is undecidable. Thus, no decision procedure exists for L_b too. Our proof is similar to Cohen's proof about the impossibility of detecting viruses.

Theorem 3. *The language L_b is undecidable.*

Proof. Let $< M, w >$ be an instance of the Halting problem. We will show how we can decide whether $< M, w >$ belongs in L_u or not using a hypothetical decision procedure (Turing machine) for the language L_b. In other words, we will show that L_u is recursive in L_b.

Given $< M, w >$, a set Q_{pan} of states, $Q_{pan} \cap Q = \emptyset$, where each state is related with actions that manifest Panopticon behaviour, is embedded into M. We design a Turing machine M_{con} that it gets M as input and modifies the transition function (see Definition 1) of M so as when a final state is reached (i.e., a state in the set F of M) a transition takes place that essentially starts the execution of the actions related to some state of Q_{pan}. In a sense, M_{con} produces now a new Turing machine M' containing the actions of M followed by actions (any of them) corresponding to the states in Q_{pan}. Now, M' is given as input the input of M, i.e., w, and operates as described above.

Let us assume that there exists a Turing machine M_b that decides L_b and suppose that M_b answers that $M' \in L_b$ when M' is given as input to M_b. Since a state in Q_{pan} was finally activated, as M_b decided, this implies that M halted on w since M' initially simulated M on w. Then we are certain that M halts on w.

Assume, now, that M_b decides that M' is not a Panopticon, i.e., $M' \notin L_b$. Then a state in Q_{pan} was never reached, which implies that no halting state is reached by M on w since

a state of Q_{pan} in M' is reached only from halting states of M, which is simulated by M'. Thus, M does not halt on w. It appears that M' is a Panopticon if and only if M halts on w and, thus, we have shown that L_u is recursive in L_b.

However, there is a catch that invalidates this reasoning: if M *itself* can exhibit the Panopticon behaviour, i.e., it already contains Q_{pan} and it can reach a state in Q_{pan} before reaching a final state. This means that Panopticon behaviour can be manifested without ever M reaching a final state that would lead M' to invoke a Panopticon state in Q_{pan}, by its construction. This situation can be solved if we create a new set of dummy ("harmless" or "no-operation") "Panopticon" states Q'_{pan} which contains a new state for each of the states in Q_{pan}. Then we replace the states from Q_{pan} that appear in the transition function of M with the corresponding states in Q'_{pan}. This new version of M is given to M_{con} to produce M' as mentioned previously. Actually, this transformation removes from a potential Panopticon the actions that *if* executed would manifest a Panopticon. We stress, again, the fact the mere existence of Panopticon actions is not considered Panopticon *behaviour*.

With this last transformation, M' is a Panopticon if and only if M halts on w and, thus, L_u is recursive in L_b. □

5. Impossibility of Detecting Deductive Panopticons

In the following two theorems, we prove the undecidability of S'_1 and S'_2. Although their undecidability follows, directly, from *Rice's Theorem* (see [7,18]), the proofs we give below provide more insightful information as they place S'_2 in a *higher* undecidability level than S'_1 (see, also, the discussion about the *Arithmetical* and the *Analytical Hierarchies* in Section 7.3).

We, first, prove the undecidability of S'_1, i.e., the impossibility of deciding for a given Turing machine (its encoding, to be precise) whether it accepts a language disjoint from a given, fixed, finite set of languages. In other words, it is impossible to detect Turing Machine(s) that decide, perhaps with effort, new information sets given some known ones.

Theorem 4. *The Halting Problem for Turing Machine(s) without oracles, i.e., L_u, is equivalent to S'_1.*

Proof. We first prove that given an oracle for S'_1 we can recognize L_u. We construct a Turing machine $M^{S'_1}$ such that given $\langle M, w\rangle$ constructs a Turing machine $M^{'}$ which operates as follows. It ignores its input and simulates, internally, M on w. $M^{'}$ accepts its input if M accepts w which means that $L(M^{'}) = \Sigma^*$ otherwise, i.e., if M does not accept w then $M^{'}$ does not accept its input and $L(M^{'}) = \emptyset$. Then, $M^{S'_1}$ asks the oracle whether $< M^{'} > \in S'_1$. If the answer is yes, i.e., $L(M^{'}) = \emptyset$, then M does not accept w. If the answer is no, then $L(M^{'}) = \Sigma^*$ and, thus, M accepts w. We, thus, can recognize L_u.

Now, we show that given an oracle for L_u we can recognize S'_1. We will construct a Turing machine $M^{''}$ such that, given M, it constructs another Turing machine $M^{'}$ that operates as follows. $M^{'}$ ignores its own input initially, and uses a generator of triples (i, j, l), $1 \leq l \leq k+1$, for simulating the lth Turing machine, M_l, with $M_{k+1} = M$, on the ith string lexicographically constructed for j steps (the Turing machine M_l runs on the ith string for j steps i.e., it applies the transition function j times). The triples are generated in increasing order of the sum $n = i + j + l$ of their components and for triples of equal component sum, in increasing i, then in increasing j (if the i components are equal), and finally in increasing l (if the i and j components are equal). Each time one of the Turing Machine(s) $M_1,\ M_2, \ldots, M_k$ accepts a particular input, this input is recorded on $M^{'}$'s second tape. Each time M_{k+1} accepts an input (i.e., M accepts an input), it is also recorded on $M^{'}$'s second tape separately from the inputs accepted by $M_1,\ M_2, \ldots, M_k$. Then, $M^{'}$ checks (using the recorded inputs stored on its second tape) whether this M_{k+1} input, or one accepted previously by M_{k+1}, has been accepted by one of the $M_1,\ M_2, \ldots, M_k$. If no, the process continues. If yes, $M^{'}$ stops the simulation and accepts its own input that was initially ignored. Thus, $< M > \in S'_1$ if $L(M^{'}) = \emptyset$ since this means that $L(M)$ is disjoint from

N_1 while $< M > \notin S_1'$ if $L(M^{'}) = \Sigma^*$, i.e., $M^{'}$ accepts all its inputs, ε in particular. Then, $M^{''L_u}$ may query its oracle set L_u for $\left\langle M^{'}, \varepsilon \right\rangle$. If the answer is yes then $M^{''}$ rejects $< M >$ which means that $< M > \notin S_1'$, otherwise it accepts $< M >$ i.e., $< M > \in S_1'$. Thus, S_1' is recognizable. □

Theorem 5. *The problem of deciding whether $L(M) = \Sigma^*$ is equivalent to S_2'.*

Proof. We first show that deciding whether $L(M) = \Sigma^*$ *is recursive in* S_2'. We construct a Turing machine $\hat{M}^{'S_2'}$ that takes as input a Turing machine M and constructs from it a Turing machine $\hat{M}^{S_1'}$, that is a Turing machine with oracle set S_1', that operates in the following way. It enumerates strings x over the alphabet Σ, and for each such string it uses oracle S_1' in order to decide whether M accepts x. This can be accomplished by constructing M' which ignores its input and simulates M on x. If M accepts x then M' accepts its input which means that $L(M') = \Sigma^*$ while $L(M') = \emptyset$ if M does not accept *any* x. Then, $\hat{M}^{S_1'}$ asks the oracle whether $< M' > \in S_1'$. If the answer is yes, which means that M does not accept *any* x, then $\hat{M}^{S_1'}$ accepts its input.

Thus, $\hat{M}^{S_1'}$ accepts its own input if and only if there is a string x *not* accepted by M. Consequently,

$$L(\hat{M}^{S_1'}) = \begin{cases} \emptyset, \text{ if } L(M) = \Sigma^* \\ \Sigma^* \text{ otherwise.} \end{cases}$$

Now $\hat{M}^{'S_2'}$ asks its oracle S_2' whether $< \hat{M}^{S_1'} > \in S_2'$, i.e., whether $L(\hat{M}^{S_1'})$ is disjoint from all sets in N_2. If the answer is yes, then $L(\hat{M}^{S_1'}) = \emptyset$ and consequently, $L(M) = \Sigma^*$. If the answer is no, on the other hand, then $L(\hat{M}^{S_1'}) = \Sigma^*$ and, thus, $L(M) \neq \Sigma^*$. Thus, deciding whether $L(M) = \Sigma^*$ is recursive in S_2'.

We now turn to showing that S_2' is recursive in the problem of whether $L(M) = \Sigma^*$. In other words, if by L_* we denote the codes of the Turing Machine(s) which accept all their inputs, then we will prove that there exists a Turing machine $\hat{M}^{''L_*}$, i.e., a Turing machine with oracle set L_*, that recognizes S_2'.

Given a Turing machine $M^{S_1'}$, we define the notion of a *valid computation* of $M^{S_1'}$ using oracle S_1' in a way similar to the notion defined in [7,8]. A valid computation is a sequence of Turing Machine step descriptions, called *Instantaneous Descriptions* or *ID*, such that the next one follows from the current one after a computational (*not* oracle query) step, according to the internal operation details (i.e., transition function or program) of the Turing machine. Roughly, an ID describes fully the status of a Turing Machine computation at each time step, containing information such as tape contents, head position, and current state.

However, if a query step is taken, i.e., the Turing machine $M^{S_1'}$ enters state $q_?$, and the next state is q_n, this means that $M^{S_1'}$ submitted a query to the oracle S_1' with respect to whether some given Turing machine, say T, belongs to the set S_1', receiving the answer no. In other words, the oracle replied that $< T > \notin S_1'$ or, equivalently, $L(T)$ is not disjoint from *all* sets in N_1. As evidence for the correctness of this reply from the oracle, we substitute the query step with a valid computation of the ordinary (i.e., with no oracle) Turing machine T that shows that a *particular* string from a language in N_1 is, also, accepted by T. If, however, after $q_?$ the state q_y follows, no computation is inserted. Intuitively, such a computation would be infinite. By definition, all valid computations conclude in a *halting*, i.e., acceptance state (see [7,8] for details).

We, now, describe the operation of $\hat{M}^{''L_*}$ with $< M^{S_1'} >$ as input. Given $M^{S_1'}$, $\hat{M}^{''L_*}$ constructs a Turing machine $M^{'}$ which accepts all computations of $M^{S_1'}$ which show that they are not a Panopticon. We call these computations *non-Panopticon* computations and they are of two disjoint types: (i) invalid computations, i.e., computations which contain invalid successions of IDs, and (ii) unsuccessful computations, i.e., computations which, although not invalid, they demonstrate that $M^{S_1'}$ is *not* a Panopticon.

$M^{'}$ interprets its inputs as computations of $M^{S'_1}$. Given such an input string, $M^{'}$ first checks if the string is malformed (i.e., it does not follow the format of a computation of a Turing Machine) or when one step does not follow from the previous one according to the internals of the Turing machine $M^{S'_1}$, or when the inserted, non-oracle, computation in a $q_?$-q_n step is not valid. In all these cases $M^{'}$ accepts the input string as an invalid computation.

However, there is some difficulty in the $q_?$-q_y cases since, as we stated above, there is no obvious *finite* computation evidence for the correctness or not of the reply. Now the Turing machine $M^{'}$ must decide on its own whether the reply to each $q_?$-q_y query is correct. Let us assume there are $t \geq 1$ such queries in the examined computation (otherwise there are no $q_?$-q_y cases to check). Let, also, w be the input string to the computation of $M^{S'_1}$ that is checked by $M^{'}$ whether it is invalid, so as to accept it.

In particular, the reply q_y to the ith, $1 \leq i \leq t$, query means that the language recognized by the queried Turing machine, say T_i, is disjoint from all the sets in N_1, i.e., $< T_i > \in S'_1$. Using a *round robin* technique similar to the *triples generation* technique described in the proof of Theorem 4, $M^{'}$ cycles, concurrently (in a *time sharing* fashion)

- (Simulation A) over all the t $q_?$-q_y queries in the examined computation of $M^{S'_1}$, trying to locate a string accepted by a queried Turing Machine T_i and one of the Turing Machines $M^1_1, M^1_2, \ldots, M^1_k$ in S'_1, and
- (Simulation B) over $M^{S'_1}$ and the Turing Machines $M^2_1, M^2_2, \ldots, M^2_k$ in S'_2, with the same input w, trying to discover whether w, which is accepted by the examined (by $M^{'}$) computation of $M^{S'_1}$, if valid, is, also, accepted by one of the Turing Machines $M^2_1, M^2_2, \ldots, M^2_k$ in S'_2.

As long as none of the above simulations concludes, $M^{'}$ continues the search. If one of them concludes, then $M^{'}$ stops the simulation and accepts its input string (which represents a computation of $M^{S'_1}$) since the computation it represents was either *invalid* (Simulation A concludes) or *unsuccessful* (Simulation B concludes). In other words, the computation was *a non-Panopticon computation*.

Based on the above, $L(M^{'}) = \Sigma^*$ if and only if $< M^{S'_1} > \notin S'_2$. Thus, $\hat{M}^{''L_*}$ can, now, ask its oracle whether $L(M^{'}) = \Sigma^*$ or not, deciding in this way S'_2 and, thus, detecting deductive Panopticons. □

6. Impossibility of Proving Panopticon Status within Formal Systems

Based on the Recursion Theorem, the following, central to our approach in this section, theorem is proved in [7] contained, among other similar results, in [9]:

Theorem 6. *Given a formal system $\mathcal{F}$, we can construct a Turing Machine $M_{\mathcal{F}}$ for which no proof exists in $\mathcal{F}$ that it either halts or does not halt on a particular input.*

Based on Definition 4 of a Panopticon (see, also, Sections 3 and 5 for the formal details of the definition), Theorem 6 and the Turing Machine $M_{\mathcal{F}}$, we now prove the following:

Theorem 7. *Let $\mathcal{F}$ be a consistent formal system. Then we can construct a Turing Machine for which there is no proof in $\mathcal{F}$ that it behaves as a Panopticon and no proof that it does not behave as a Panopticon.*

Proof. We will reduce the problem of deciding whether the Turing Machine $M_{\mathcal{F}}$ halts on a given input w to the problem of deciding whether a given Turing Machine M is a Panopticon.

For some fixed k, we define a set $N = \{L_1, L_2, \ldots, L_k\}$ of recursively enumerable languages and a set of corresponding Turing Machines $M_1, M_2, \ldots, M_k$ which recognize them correspondingly. We also assume there exists a recursively enumerable language L, recognized by a Turing Machine M, which is disjoint from N.

These elements can be effectively constructed. For instance, for $k = 2$, we can set $L_1 = \{$The multiples of 2, except 2 itself$\}$, $L_2 = \{$The multiples of 3, except 3 itself$\}$, and $L = \{$All primes$\}$. For each of these languages (in particular L) we may construct a

corresponding Turing Machine that recognizes it. Also, note that L is disjoint from the set $\{L_1, L_2\}$ and, thus, the Turing Machines that accept this language are Panopticons.

Given these elements as well as the Turing Machine $M_{\mathcal{F}}$ and an input w on which we want to decide whether $M_{\mathcal{F}}$ halts or not, we proceed as follows. We construct a Turing Machine $M_?$ which, given w' as input, simulates M on w' and $M_{\mathcal{F}}$ on w, alternating between them. Then $M_?$ accepts if either of them accepts at some step of the simulation process.

We, now, observe that $L(M_?) = L(M)$, if $M_{\mathcal{F}}$ does not halt on w while $L(M_?) = \Sigma^*$ if $M_{\mathcal{F}}$ halts on w. Then, according to the definition of a deductive Panopticon, $M_?$ is a Panopticon if and only if $M_{\mathcal{F}}$ *does not* halt on w.

But, now, if a proof existed in the formal system $\mathcal{F}$ that $M_?$ either is a Panopticon or it is not a Panopticon then the same proof could be used to prove that $M_{\mathcal{F}}$ either halts or does not halt, correspondingly, contradicting, thus, Theorem 6. □

7. Philosophical Remarks on Panopticon Detectability

In this final section, we consider the problem of detecting oracle-based Panopticons through their *physical* properties and by *Hypercomputation* computational models, focusing on the *Infinite Time Turing Machine (ITTM)* model.

With respect to the *physical* detection, we discuss how the action of receiving information "for free" through the oracle reply, forces the Panopticon to perform an *irreversible* operation which, necessarily, dissipates heat to the environment, manifesting the Panopticon's presence. A reversible operation of a Turing Machine implies that one can trace or rewind the computation back to the beginning, from any step of the evolution of computation or, equivalently, if the Turing Machine is found at a particular state, then there is only a *unique* predecessor state from which this state may have resulted. We base our discussion on the theory of the *thermodynamics of computation* and the fact that irreversible computational operations, necessarily, consume energy and dissipate heat to the environment (see [10,19]).

Moving to *hypercomputation*, we consider the power of *Infinite Time Turing Machines* with respect to detecting oracle-based Panopticons. We conclude that since the hypercomputation decidability powers transcend the *Arithmetical Hierarchy* and reach the first level of the *Analytical Hierarchy* (see [11]), simple oracle-based Panopticons *are* detectable by ITTMs. Then we consider the question of defining ITTM-based Panopticons that evade the detectability power of ITTMs.

7.1. Detectability of Panopticons by Their Thermal Emissions

In the preceding sections, we proved that, under a *plausible* formal definition of Panopticons as *Oracle Turing Machines*, detecting Panopticons by "normal" Turing Machines, i.e., Turing Machines *without* oracles, fully realizable and algorithmic in operation, is an undecidable problem. However, as we discuss in this section, it may still be possible to detect Panopticons or, more generally, Turing Machines deploying Oracles, based on *non-algorithmic* approaches, namely the theory of the *thermodynamics of computation*.

It is a well-known fact that any (non-oracle) Turing Machine can be transformed so as to operate, in a physical implementation, in a way that does not dissipate energy, i.e., it can be transformed so as to operate *reversibly* (see [19]). Reversibility in computation is important due to the fact that only operations that are irreversible *necessarily* consume energy and *dissipate* heat to their environment. Such irreversible operations include, for instance, the logical AND of two bits or resetting to a *default* state (e.g., to '0') a 1-bit memory. However, as we stated above, in principle any Turing Machine, which normally operates in an irreversible way, can be transformed into an equivalent, as far as the computation is concerned, *reversible* Turing Machine (see [19] for the formal details of the transformation).

On the other hand, *Oracle Turing Machines* and *Panopticons* as we defined them in this paper, for that matter, include a step which cannot be performed *reversibly*. This step results from the consultation of the oracle when the Oracle Turing Machine enters the oracle query state. The oracle consultation cannot be replaced, in general, by the computation of a Turing

Machine that always halts and supplies the correct answer to the oracle query since the oracle may correspond to an undecidable problem such as the Halting Problem, as it is in our case. Moreover, this step is irreversible, by its nature since to a particular oracle query response there may correspond more than one (perhaps infinitely many) strings for which the response is correct. Thus, it is not possible to infer, uniquely, the query string, given the query response. More precisely, the oracle reply, *Yes* or *No*, is stored in a 1-bit memory as a 1 or 0 respectively. Since there is no connection with the query to which the reply was given, the memory can be considered as containing a *random* or *unknown* bit just before the reply is given to the next query. However, setting a previously unknown bit to either 0 or 1 is an irreversible operation since given the set value, it is impossible to infer the previous one. According to the thermodynamics of information, this operation consumes energy equal to $k_B T \ln 2$, where k_B is Boltzmann's constant and T the environment temperature, and, thus, dissipates heat to the Panopticon's environment (see, e.g., [10]). Therefore, this oracle consultation step, which is unavoidably irreversible, dissipates heat to the environment of the Panopticon and, thus, provides evidence of its existence and operation (in the form of *increasing temperature* in its environment), even though detecting Panopticons *algorithmically*, i.e., by non-Oracle Turing Machines, is an undecidable problem.

In some sense, an Oracle Turing Machine can be viewed as a type of *Maxwell's Demon*. The reader is urged to consult the excellent collection of seminal papers in [20] on its history as well as impact on philosophy, information theory, and thermodynamics of computation. Similarly to Maxwell's Demon, the Oracle Turing Machine involves itself in a cyclic process of (i) normal computation followed by an *observation*, in the form of *submitting* a query string to its oracle, which results to an instantaneous "observation" result, i.e., the reply of the oracle, and (ii) Storing the result of the observation (i.e., query reply, "YES" or "No") on an 1-bit memory. As in the resolution of the tantalizing operation of Maxwell's Demon which appears to perform useful work without consuming energy, apparently violating the *Second Thermodynamics Law*. In the final resolution of the "paradox", after many failed attempts, Bennett showed (see [10,19]) that it was not the observation the process that should, necessarily, consume energy (it can be performed reversibly) but the operation of resetting of the 1-bit memory from an unknown value, to a specific value that reflects the result of the observation, 1 or 0 (i.e., "YES" or "NO", as oracle replies in our case). Thus, *information theory* was linked, tightly, to *thermodynamics*. Likewise, the Panopticon operating as an Oracle Turing Machine does not betray its presence through the process of "observation", i.e., invoking the oracle, bur by manipulating the 1-bit memory that holds the oracle reply.

In summary, it appears that the very strength of Panopticons (defined as Oracle Turing Machines), which is their ability to "observe" everything and acquire knowledge without computational effort, can be their very weakness. This is due to the fact that this ability relies on irreversible operations (i.e., setting to either 0 or 1 an 1-bit memory which holds a *random* value that holds the result of the observation, or, query to the oracle) which dissipate energy and emit heat to the Panopticon's environment revealing, in this way, the Panopticon's existence and operation.

7.2. Infinite Time Turing Machine Hypercomputation Model

The Oracle Turing Machines (see Definition 2), or O-machines as defined by Turing, is an instance of a general class of theoretical computation models named *Hypercomputation* or *Super-Turing* (see [21,22]). Within the context of all these models, problems such as the Halting Problem can be solved, which cannot by solved by ordinary Turing Machines and, thus, are not confined by the Church-Turing Thesis.

In this section we focus on *Infinite Time Turing Machine* computational model or *ITTM* for short. A machine in this model operates much like an ordinary Turing Machine in the sense of Definition 1, as follows: each computational step is executed, in succession, after the previous one through the application of the transition function based on the contents of each tape cell and the current machine state. In this way, each configuration of

the machine at computation step $t+1$ results from the configuration of the machine at step t according to the tape contents and the actions of the transition function, as it is the case for ordinary Turing Machines under Definition 1. The critical difference is that in contrast with Turing Machines which are required to produce the correct computation result after a *finite* number of steps (otherwise the computation proceeds indefinitely without producing any result) the machines in the ITTM model are allowed to compute *and* produce results using an infinity of steps. To avoid some important technicalities (see [11] for the details) with respect to how a result can be obtained within an infinite number of steps, the theory of *ordinals* is employed to define specific computation limit steps upon which results are obtained from a machine in the ITTM model. Before we continue, we should remark that this computational model has received harsh criticism (see, for instance, [13,23,24]). However, this model can be, nevertheless, proved to be a promising theoretical tool for studying the limits of Panopticism, at least when Panopticons are based on some formal theoretical computational model.

Ordinals can be used as "labels" of the order of the elements of a *well-ordered* set, i.e., a set in which every pair of its elements is comparable. The labeling is as follows: the smallest element of the set is labeled 0, the one coming after the smallest element is labeled 1, the next element 2 etc. The size of the set is equal to the *least* ordinal that does not constitute a label for a set element. This size is the *order type* of the set.

An ordinal can be, inductively, defined as the set of the preceding ordinals. For instance, the *finite* ordinal 20 is the order type (or size) of the ordinals that precede it, i.e., the finite ordinals 0 up to 19 which is the immediate predecessor of 20. Thus, the ordinal 20 is identified as the set $\{0, 1, \ldots, 19\}$. Conversely, any well-ordered set S such as for each ordinal $\alpha \in S$ and any $\beta < \alpha$, it also holds that $\beta \in S$, can be identified with an ordinal.

Except the finite ordinals, which can be identified with the elements of the set $\mathbb{N}$ of natural numbers, there are also *infinite* ordinals, the smallest one being ω, the order type of the natural numbers (which are the finite ordinals). The ordinal ω, thus, can be identified with the set $\mathbb{N}$ of natural numbers.

In the terminology of ordinals, if an ITTM reaches a halting state within a number of steps equal to a *finite* ordinal, then the halting procedure is the one followed by an ordinary Turing Machine. However, whereas the computation may be impossible for an ordinary Turing Machine, which implies that for some input(s) it fails to halt within a finite number of steps, as it happens for the Halting Problem of any undecidable problem, for machine in ITTM the computation is considered to step on *all* finite ordinals, until it reaches the first infinite ordinal ω which is the first *infinite* halting step. Thus, after all finite ordinals, comes the first *infinite* ordinal ω. Next, in succession, come the infinite ordinals $\omega+1$, $\omega+2$ etc. until we reach $\omega+\omega$ or $\omega \cdot 2$. Then come $\omega \cdot 2+1$, $\omega \cdot 2+2$ etc. until we reach $\omega \cdot 3$ which is $\omega+\omega+\omega$. In general, the infinite ordinals are written as $\omega \cdot n+m$, with n, m natural numbers (or finite ordinals). With this set of ordinals we associate the ordinal ω^{ω}. Likewise, some ordinals which come next are $\omega \cdot 4$, and then later on $\omega \cdot 5$. Analogously, we have ordinals such as ω^3, ω^4, ω^5, ω^{ω}, $\omega^{\omega^{\omega}}$ and at some point the ordinal ϵ_0. This can be continued infinitely, enumerating the *countable* ordinals until we reach the smallest *uncountable* ordinal ω_1. Loosely speaking, much as we say an ordinary Turing Machine *falls in an infinite loop* and, thus, does not terminate if the number of steps is infinite, or we may say ω in the language of ordinals, we may, likewise, say that a machine in the ITTM model falls in an infinite loop if the computation steps over all countable ordinals reaches ω_1.

The key element in the computation of a machine of the ITTM model is to *define* its state at steps that go over the countable set of the *limit* ordinals such as ω, which is the first infinite ordinal and, thus, the first "infinite step" of a machine operation after the *finite* steps (ordinals) which are the natural numbers $\mathbb{N}$ and which form the numbers of steps that a computation of an ordinary Turing Machine is allowed to take.

Moreover, the most important thing in the operation and halting process of a machine of the ITTM is to define a type of *limit* for an infinite computation to which the computation will be said to *converge* providing the correct computation result. Thus, we need to define

the machine's state or configuration at limit steps ω, $\omega \cdot 2$, $\omega \cdot 3$ etc., i.e., all *limit ordinals* of the *hypercomputation*. As described in [11], at limit states, the machine tape head is rewound to the first cell and the machine is set at a distinguished state, the *limit state* much like the halting state of an ordinary Turing Machine. At this point, the limit value of all tape cells is taken according to the following rule: if the value appearing in a cell has converged, i.e., they have reached the value 0 or 1 *before* the limit step we examine, then the cell keeps this value *at* the limit step. If convergence has not taken place before the limit step, that is if the values of the cell were alternating between 0 and 1, *unboundedly often*, then the cell is set at the limit value 1. Equivalently, the cell's limit value is set to the lim sup of the cell values appearing before the limit step we examine.

This discussion (see [11]) describes, precisely, the configuration of a machine of the ITTM model at all limit steps β, i.e., all limit infinite ordinals, the first of which is ω. Between limit steps, i.e., $\beta + 1$, $\beta + 2, \ldots$ before reaching the next limit step $\beta + \omega$, the computation proceeds normally, as in ordinary Turing Machines. If at any computation step between limit steps the machine reaches the *halting* state, the computation stops the the cell value is the value already existing there. As long as the *halting state* is not reached, the computation proceeds indefinitely, over the limit ordinals and the steps (natural numbers) between them.

7.3. Detecting Deductive Panopticons with Ittm Computation Model

Based on the above, it is easy to see that the Halting Problem is decidable by a machine of the ITTM class. Given an ordinary Turing Machine M and an input w to it, we provide $< M, w >$ to a Universal Machine of the class ITTM. This universal machine writes 0 at a designated output cell, meaning "NO", i..e. the Turing Machine has not halted yet. Then it simulates M on w if at any step the Turing machine halts, the universal machine writes "YES" on the output cell and halts, signifying that M halted on w. Otherwise, the first limit step ω will be reached, the limit value 0 will be computed on the output cell (actually, the same as the initial value - no alternation between values took place), and the universal machine will halt.

Definition 5. *The i-th level of the* arithmetical hierarchy *contains* 3 *classes* Σ_i^0, Π_i^0, *and* Δ_i^0. *The class* Σ_i^0 *is the set of languages defined as*

$$L = \{x \in \{0,1\}^* : \exists y_1 \forall y_2 \exists y_3 \cdots Q\, y_i\, R(x, y_1, \cdots, y_i) = 1\}$$

for some total Turing machine R, where Q is the quantifier $\forall$ *when i is even, or* $\exists$ *when i is odd. The class* Π_i^0 *is the complement of* Σ_i^0, *and* $\Delta_i^0 = \Sigma_i^0 \cap \Pi_i^0$.

Thus, the arithmetical hierarchy includes sets definable by first order logic or arithmetic statements. If we allow quantification over sets instead of single elements, we obtain the *analytical hierarchy* with analogous definition and a notation in which the superscript 0 replaced by 1 (see [25]).

In particular the first level of the arithmetical hierarchy corresponds to the classes $\Sigma_1^0 = \mathsf{RE}$, $\Pi_1^0 = \mathsf{coRE}$, and Δ_1^0 the set of decidable languages $\mathsf{RE} \cap \mathsf{coRE}$.

We can study the set of Turing Machine computable functions within the context of oracles, as we discussed in Section 3. That is, we can consider Oracle Turing Machines with oracle set the empty set $\emptyset$. We can iterate this process much like it is done with the sets S_i defined in Section 3. It is easy to see that O-machines at a specific iteration level i cannot solve the Halting Problem for themselves *but only* for lower levels, acting as oracles with the oracle set $\emptyset$. Thus, as i increases, we obtain increasingly more powerful classes of Turing Machines.

The iteration step for obtaining the *Halting Problem* for a class of Oracle Turing Machines i with oracle set X in general (X is the empty set in our discussion but it can be any set enumerable by an Oracle Turing Machine) is written X' which is known as the *jump operator*. In this notation, for $X = \emptyset$, the Halting Problem (and all Recursively Enumerable

functions) can be seen to be recursive in $\emptyset'$, the Halting Problem for Oracle Turing Machines with oracle set $\emptyset'$ is recursive in $\emptyset''$ etc.

Based on the jump operator, the following can be proved:

1. *The representation theorem:* For any relation R, R is in the arithmetical hierarchy iff R is definable in elementary arithmetic.
2. *The strong hierarchy theorem:*
   - $R \in \Delta^0_{n+1}$ iff R is recursive in $\emptyset^{(n)}$ (i.e., the nth jump of the empty set).
   - $R \in \Sigma^0_{n+1}$ iff R is recursively enumerable in $\emptyset^{(n)}$.
   - $R \in \Pi^0_{n+1}$ iff R is recursively enumerable in $\emptyset^{(n)}$.

The Arithmetical and Analytical Hierarchies, separated by the jump operator (appearing as 0 instead of $\emptyset$) are shown in Figure 1.

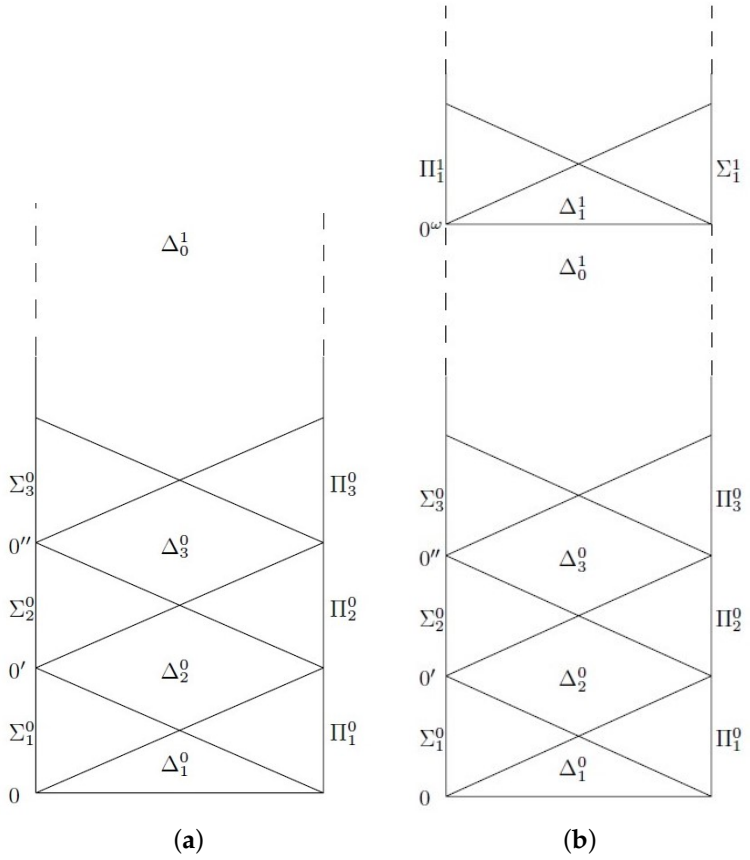

Figure 1. The Arithmetical and the Analytical Hierarchies. (**a**) The Arithmetical Hierarchy; (**b**) The Analytical Hierarchy.

With respect to the power of machines of the ITTM class, in [11] it is shown that the class of the ITTM-decidable sets lie between $\Sigma^1_1 \cup \Pi^1_1$ and Δ^1_2 and extends beyond the arithmetical hierarchy into the analytical hierarchy of sets. The ITTM-decidable sets, thus, certainly include L_u (the Halting Problem), which is placed low in the arithmetic hierarchy in Σ^0_1 (actually all sets in Σ^0_1 many-one reduce to L_u) and the language consisting of encodings of Turing Machines who accept Σ^*, which is placed, also, low (but higher than L_u) in the arithmetical hierarchy in $\Pi^0_2 - \Pi^0_1$.

Let us, now, examine our two *deductive* Panopticon detection problems in the context of the ITTM computation model. We have defined two such problems, one represented by the set S'_1 and one by the set S'_2. With respect to S'_1, we proved in Theorem 4 that the problem of deciding this language is *equivalent* to L_u, which represents the Halting Problem. In view of the discussion above, a machine from the ITTM model decides S'_1 since it decides L_u, i.e., solves the Halting Problem, in ω steps.

With respect to the second Panopticon problem, i.e., deciding S'_2, we proved in Theorem 5 that this problem is equivalent to the problem of deciding the problem of whether, for a given Turing Machine M, $L(M) = \Sigma^*$. This problem belongs to the $\Pi^0_2 - \Pi^0_1$ level of the arithmetical hierarchy (see [25]) and, thus, deciding S'_2, i.e., solving the second Panopticon detection problem, is also in the $\Pi^0_2 - \Pi^0_1$ level.

8. Discussion and Directions for Future Research

Theorems 3–5 in Section 3 show that, even for Panopticons with the simple behaviours described in Definitions 3 and 4, it is impossible, in principle, to detect them. Potential Panopticons, naturally, can have any imaginable, complex, behaviour but then the problem of detecting them may become harder compared to our definitions.

Comparing, now, Theorems 3–5, Theorem 3 examines the detection of Panopticons based on the execution of *specific* visible or detectable actions, i.e., on a *behavioural level*, such as connecting to a server and sending eavesdropped information or sending an email to the unlawful recipient. Theorems 4 and 5 examine Panopticon detection not based on their visible behaviour but from what languages they may recognize, without having any visible clue of behaviour or actions, only their *descriptions* as Turing Machine(s) (i.e., programs or systems). These theorems, that is, examine the detection of Panopticons at a *"metabehavioural" level*.

With respect to the difference between Theorems 4 and 5, we first observe that L_u is recursively enumerable but not recursive while the $\{< M > | L(M) = \Sigma^*\}$ language is *not* recursively enumerable (see, e.g., [7]). Although they are, both, not recursive (i.e., not decidable), their "undecidabilities" are of different levels, with the $\{< M > | L(M) = \Sigma^*\}$ language considered "more difficult" than L_u in restricted types of Turing Machine(s) (Panopticons). For example, the L_u language is decidable for Context-free Grammars (i.e., for Turing Machine(s) modelling Context-free Grammars) while the $\{< M > | L(M) = \Sigma^*\}$ language is still undecidable. Also, for regular expressions, the problem of deciding L_u is solvable efficiently (i.e., by polynomial time algorithms) while the $\{< M > | L(M) = \Sigma^*\}$ language has been shown, almost certainly, to require exponential time (in the length of the given regular expression) to solve (see, e.g., [7]). Therefore, a similar decidability complexity status is expected from S'_1 (deductive Panopticons without external advice) and S'_2 (deductive Panopticons with external advice in the form of an oracle) since they are equivalent to the languages L_u and $\{< M > | L(M) = \Sigma^*\}$ respectively. That is, when we consider more restricted definitions of Panopticons that render the detection problem decidable, then deciding which Panopticons belong in S'_1 is expected to be easier than deciding which Panopticons belong in S'_2.

Finally, Theorem 7 shows that for any formal system $\mathcal{F}$, we can, effectively, exhibit a particular Turing Machine for which there is no proof in $\mathcal{F}$, that it is either a Panopticon or it is not a Panopticon, emphasizing the difficulty of recognizing Panopticons by formal means.

As a next step, it is possible to investigate the status of the Panopticon detection problem under other definitions, either targeting the behaviour (i.e., specific actions) of the Panopticon or its information deducing capabilities (e.g., recognizing languages with specific closure properties or properties describable in some formal system such as second order logic). Our team plans to pursue further Panopticon definitions in order to investigate their detection status, especially for the decidable (and, thus, more practical) cases of suitably constrained Panopticons.

Moreover, in view of the considerations discussed in Section 7.1, it would be interesting to pursue, further, the computational undetectability of Panopticons in the form of oracle Turing Machines with their, apparent, *physical* detectability. This detectability is possible through the energy dissipation that takes place each time the oracle is consulted and the result is provided *without* any computational effort which, if it was expended by the Panopticon Turing Machine, it could be transformed into a reversible sequence of steps, so that no energy dissipation takes place which could betray the Panopticon. It appears that the mere Panopticon action, i.e., "knowing" without "toiling", deprives the Panopticon of any potential to hide its presence.

Finally, it would be interesting to study Panopticon capabilities in the context of other "unrealistic" models such as the Infinite Time Turing Machines (ITTM) which we discussed in Section 7.2. We remarked that a Turing Machine of the ITTM class decides the Panopticon detection problem under both definitions that we gave in this paper, although the two definitions lead to problems belonging in different levels of the Arithmetic Hierarchy. Actually, since the power of such a Turing Machine steps well into the first level of the *Analytical Hierarchy*, it would be interesting to provide (as natural as possible) Panopticon definitions that fall in higher levels. Also, the possibility of an *ITTM* Panopticon can be investigated in order to define its potential behaviour, its properties and power.

In conclusion, we feel that the formal study of the power and limitations of massive surveillance establishments and mechanisms of today's as well as of the future Information Society can be, significantly, benefited from fundamental concepts and deep results of computability and computational complexity theory. We hope that our work will be one step towards this direction.

Author Contributions: V.L., P.E.N., P.S. and Y.C.S. have contributed, equally, to all aspects of the paper. All authors have read and agreed to the published version of the manuscript.

Funding: The Research of P.E. Nastou has been co-financed by the European Regional Development Fund of the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH–CREATE–INNOVATE (project code:T2EDK-01862). The work of V. Liagkou, P. Spirakis and Y.C. Stamatiou was partially supported by the CyberSec4Europe project, funded by the European Union under the H2020 Programme Grant Agreement No. 830929. Finally, we would like to thank the anonymous reviewers whose comments contributed greatly to improving the presentation of our ideas and results.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Bentham, J. Panopticon or the Inspection House. Written as a Series of Letters in 1787. Available online: https://www.ics.uci.edu/~djp3/classes/2012_09_INF241/papers/PANOPTICON.pdf+ (accessed on 15 January 2022).
2. Foucault, M. *Discipline and Punish: The Birth of the Prison*; Random House: New York, NY, USA, 1977.
3. Cohen, F. Computer Viruses. Ph.D. Thesis, University of Southern California, Los Angeles, CA, USA, 1985.
4. Cohen, F. Computer Viruses: Theory and Experiments. *Comput. Secur.* **1987**, *6*, 22–35. [CrossRef]
5. Turing, A.M. On Computable Numbers, with an Application to the Entscheidungs problem. *Proc. Lond. Math. Soc.* **1936**, *2*, 230–265.
6. Turing, A.M. Systems of logic based on ordinals. *Proc. Lond. Math. Soc.* **1939**, *45*, 161–228. [CrossRef]
7. Hopcroft, J.; Ullman, J.D. *Introduction to Automata Theory, Languages, and Computation*; Addison-Wesley Series in Computer Science; Addison-Wesley: Boston, MA, USA, 1979.
8. Hartmanis, J.; Hopcroft, J.E. Structure of undecidable problems in automata theory. In Proceedings of the 9th Annual IEEE Symposium on Switching and Automata Theory (SWAT 1968), Schenedtady, NY, USA, 15–18 October 1968; pp. 327–333.
9. Hartmanis, J.; Hopcroft, J.E. Independence results in computer science. *ACM Sigact News* **1976**, *8*, 13–24. [CrossRef]
10. Bennett, C.H. The Thermodynamics of Computation—A Review. *Int. J. Theor. Phys.* **1982**, *21*, 905–940. [CrossRef]
11. Hamkins, J.D. Infinite Time Turing Machines. *Minds Mach.* **2002**, *12*, 521–539. [CrossRef]
12. Evans, D. *Introduction to Computing: Explorations in Language, Logic, and Machines*; Eleven Learning: Delhi India, 2011.

13. Davis, M. *The Universal Computer: The Road from Leibniz to Turing*, 3rd ed.; CRC Press: Boca Raton, FL, USA, 2018.
14. Post, E.L. Formal reductions of the general combinatorial decision problem. *Am. J. Math.* **1943** *65*, 197–215. [CrossRef]
15. Post, E.L. Recursively enumerable sets of positive integers and their decision problems. *Bull. Am. Math. Soc.* **1944**, 50, 284–316. [CrossRef]
16. Post, E.L. Degrees of recursive unsolvability: Preliminary report (abstract). *Bull. Am. Math. Soc.* **1948**, *54*, 641–642.
17. Kleene, S.C.; Post, E.L. The upper semi-lattice of degrees of recursive unsolvability. *Ann. Math.* **1954**, *59*, 379–407. [CrossRef]
18. Rice, H.G. Classes of Recursively Enumerable Sets and Their Decision Problems. *Trans. Am. Math. Soc.* **1953**, *74*, 358–366. [CrossRef]
19. Bennett, C.H. Logical Reversibility of Computation. *IBM J. Res. Dev.* **1973**, *17*, 525–532. [CrossRef]
20. Leff, H.; Rex, A.F.; Hilger, A. (Eds.) *Maxwell's Demon: Entropy, Information, Computing*; Princeton University Press: Princeton, NJ, USA, 1990.
21. Abbott, A.A.; Calude, C.S.; Svozil, K. On Demons and Oracles. In *Asia Pacific Mathematics Newsletter*; World Scientific: Singapore, 2012; Volume 2.
22. Aoun, M.A. Advances in Three Hypercomputation Models. *Electron. J. Theor. Phys. (EJTP)* **2016**, *13*, 169–182.
23. Davis, M. The Myth of Hypercomputation. In *Alan Turing: Life and Legacy of a Great Thinker*; Teuscher, C., Ed.; Springer: Berlin/Heidelberg, Germany, 2004; pp. 195–211.
24. Davis, M. Why there is no such discipline as hypercomputation. *Appl. Math. Comput.* **2006**, *178*, 4–7. [CrossRef]
25. Kleene, S.C. Recursive predicates and quantifiers. *Trans. Am. Math. Socl.* **1943**, *53*, 41–73. [CrossRef]