On the undecidability of the Panopticon detection problem*

V. Liagkou[1,4], P.E. Nastou[5,6], P. Spirakis[2], and Y.C. Stamatiou[1,3]

[1] Computer Technology Institute and Press - "Diophantus", University of Patras Campus, 26504, Greece
[2] Department of Computer Science, University of Liverpool, UK and Computer Engineering and Informatics Department, University of Patras, 26504, Greece
[3] Department of Business Administration, University of Patras, 26504, Greece
[4] University of Ioannina, Department of Informatics and Telecommunications, 47100 Kostakioi Arta, Greece
[5] Department of Mathematics, University of the Aegean, Applied Mathematics and Mathematical Modeling Laboratory, Samos, Greece
[6] Center for Applied Optimization, University of Florida, Gainesville, USA
e-mails: liagkou@cti.gr, pnastou@aegean.gr, P.Spirakis@liverpool.ac.uk, stamatiu@ceid.upatras.gr

Abstract. In this paper we provide a theoretical framework for studying the detectability status of Panopticons based on two theoretical definitions. We show, using *Oracle Turing Machines*, that detecting modern day, ICT-based, Panopticons is an undecidable problem.

Keywords: Formal Methods · Security · Privacy · Undecidability · Panopticon · Turing Machine · Oracle Computations

1 Introduction

In this paper we, formally, investigate the complexity of detecting *Panopticons* (see the pioneering works of Bentham and Foucault [1, 4]), as synonyms of *massive survaillance* in modern societies, based on *Turing Machines*. We provide two different, but not unrealistic, theoretical models of a Panopticon and show that there is no algorithm that can detect, systematically, all Panopticons 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.

* 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 the first, third and fourth coauthors was partially supported by the Cyber-Sec4Europe project, funded by the European Union under the H2020 Programme Grant Agreement No. 830929.

More specifically, the first formal model we examine studies Panopticons whose Panopticon behaviour is manifested through the *execution* of states (actions) that belong to a specific set of states that characterizes Panopticon behaviour. In some sense, since the focal point of this model is the *execution* of states 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: acquiring, rather, *effortlessly* information through *surveillance* and *eavesdropping.* We model this characteristic using *Oracle Turing Machines* with the oracle having the role of 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.* The focus is 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.

2 Definitions and notation

In this section we briefly state the relevant definitions and notation that will be used in the subsequent sections. We, first, define a simple extension of a Turing Machine, following the notation in [7].

Definition 1 (Turing Machines). *A Turing Machine can be defined as 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.*

With $< M >$ we will denote the *encoding* or *code* of the Turing Machine.

One of the main outcomes of Turing's pioneering work [9] was that there exist problems that Turing machines cannot solve. The first, such, problem was the, so called, *Halting problem*:

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.* The language L_u was the first language proved to be non-recursive or undecidable by Turing in his seminal work [9].

In order to discuss Panopticons, we need an important variant of Turing machines, 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 [10] under the name *O-machine.* Post's collaboration with Kleene in [8] resulted to the definition that is used today in computability theory.

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 asks: "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.*

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

In Cohen's pioneering work (see [2, 3]) a natural, *formal*, definition of a virus is provided based on Turing machines. Specifically, Cohen defined a virus to be a program, or Turing machine, that simply copies itself to other programs, or more formally, injects its transition function into other Turing machines' 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 reasonable definitions of a Panopticon. A Panopticon is a Turing machine that when executed will demonstrate a specific, recognizable, behaviour particular to Panopticons manifested by the *execution* (not simply the existence) of a sequence of actions, e.g. it will publish secret information about an entity, it will download information illegally etc., actions that can be reflected by reaching, during its operation, particular states in a set Q_{pan}.

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

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

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 $Q_{pan} \subset Q$, $Q_{pan} \cap F = \emptyset$, 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.

Beyond displayed behaviour, however, 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 behaviour 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, in a sense, it deduces further information, perhaps expending some computational effort this time. We model the characteristic Panopticon action, i.e. *observation* or *surveillance*, using oracle Turing Machines, where the freely acquired information is modeled 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 behaviour 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 characteristicc is the *deduction* of new information from already known information.

Definition 4. *(Deductive Panopticons) A Panopticon is a Turing Machine that either by itself (language S_1') or based on observed or stolen information and, thus, acquired without expending computational effort to deduce or produce it (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 Panopticon Detection Problem 1
Input: A description of a Turing machine (program).
Output: If the input Turing machine behaves 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. Then (we omit the proof due to lack of space) the following can be proved:

Theorem 1. *(Impossibility of detecting behavioural Panopticons) The language L_b is undecidable.*

The Panopticon Detection Problem 2
Input: A description of a Turing machine (program).
Output: If the input Turing machine behaves like a Panopticon according to Definition 4 output True. Otherwise, output False (essentially, this problem 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 [5] 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, as we discussed before. 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 [5, 7]):

Theorem 2. *The membership problem for TM's without oracles is equivalent to S_1 (i.e. L_u is equivalent to S_1).*

Theorem 3. *The problem of deciding whether $L(M) = \Sigma^*$ is equivalent to S_2.*

Our first contribution is to propose a plausible Panopticon model which incorporates the *information gathering and deduction* element of its behaviour (see Definition 4). 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 machines that, correspondingly, accept these languages. These Turing machines and their corresponding languages model the fixed, finitely many, 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 $S_1' = \{< M > | L(M) \text{ is disjoint from } N_1\}$ ($< M >$ is the encoding of Turing machine M), and, recursively, in analogy with [5, 7], 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 4 we prove two theorems analogous to Theorems 2 and 3 on the undecidability of the problem of detecting a deductive Panopticon. The first one, 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.

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.

4 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]), the proofs we give below provide more insightful information as they place S'_2 in a *higher* undecidability level than S'_1.

Theorem 4. *The Halting Problem for Turing machines 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 $M^{S'_1}$ 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 yes, i.e. $L(M^{'}) = \emptyset$, then M does not accept w. If 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 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 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 $M_1,\ M_2,\ldots,M_k$ accepts a particular input, this input is recorded on $M^{'}$'s second tape. Each time M_{k+1} accepts an input, 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 $M_1,\ M_2,\ldots,M_k$. If no, the process continues. If yes, $M^{'}$ stops the simulation and accepts its own input. Thus, $< M > \in S_1^{\prime}$ if $L(M^{'}) = \emptyset$ since this means that $L(M)$ is disjoint from N_1 while $< M > \notin S_1^{\prime}$ if $L(M^{'}) = \varSigma^*$, 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 $< M > \in S_1^{\prime}$, otherwise it accepts $< M >$ i.e. $< M > \notin S_1^{\prime}$. Thus, $S_1^{\prime}$ is recognizable. □

Theorem 5. *The problem of deciding whether $L(M) = \varSigma^*$ is equivalent to $S_2^{\prime}$.*

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

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

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

Now $\hat{M}^{'S_2^{\prime}}$ asks its oracle $S_2^{\prime}$ whether $< \hat{M}^{S_1^{\prime}} > \in S_2^{\prime}$, i.e. whether $L(\hat{M}^{S_1^{\prime}})$ is disjoint from all sets in N_2. If the answer is yes, then $L(\hat{M}^{S_1^{\prime}}) = \emptyset$ and, thus, $L(M) = \varSigma^*$. If no, then $L(\hat{M}^{S_1^{\prime}}) = \varSigma^*$ and, thus, $L(M) \neq \varSigma^*$. Thus, deciding whether $L(M) = \varSigma^*$ is recursive in $S_2^{\prime}$. We show that $S_2^{\prime}$ is recursive in the problem of whether $L(M) = \varSigma^*$. If L_* contains the codes of the Turing machines accepting 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_*, recognizing $S_2^{\prime}$.

Given a Turing machine $M^{S_1^{\prime}}$, we define the notion of a *valid computation* of $M^{S_1^{\prime}}$ using oracle $S_1^{\prime}$ in a way similar to the notion defined in [5, 7]. 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 [5, 7] for details).

We describe the operation of $\hat{M}''^{L_*}$ with $< M^{S_1'} >$ as input. Given $M^{S_1'}$, $\hat{M}''^{L_*}$ constructs 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, M' first checks if the string is malformed (i.e. not of correct format) 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 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

- (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.
- (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. $\square$

5 Conclusions

In this paper we addressed the problem of detecting Panopticons and their activity based on *Oracle Turing Machines.* Comparing Theorems 1, 4, and 5, Theorem 1 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 machines (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 machines (Panopticons). For example, the L_u language is decidable for Context-free Grammars (i.e. for Turing machines modeling 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.

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.

References

1. J. Bentham. *Panopticon or The Inspection House.* Written as a series of letters in 1787.
2. F. Cohen. *Computer Viruses.* PhD thesis, University of Southern California, 1985.
3. F. Cohen. Computer Viruses: Theory and Experiments. *Computers & Security*, Volume 6, Issue 1, pp. 22–35, 1987.
4. M. Foucault. *Discipline and Punish: The Birth of the Prison.* New York: Random House, 1977.
5. J. Hartmanis and J.E. Hopcroft. *Structure of undecidable problems in automata theory.* In *Proc. 9th Annual IEEE Symposium on Switching and Automata Theory (SWAT 1968)*, pp. 327–333, 1968.
6. J. Hartmanis and J.E. Hopcroft. Independence results in computer science. *ACM SIGACT News,* Volume 8, Issue 40, pp. 13–24, October-December 1976.
7. J. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley series in Computer Science, 1979.
8. S.C. Kleene and E.L. Post. The upper semi-lattice of degrees of recursive unsolvability. *Ann. of Math.* **59**, 379–407, 1954.
9. A.M. Turing. On Computable Numbers, with an Application to the Entscheidungs problem. *Proceedings of the London Mathematical Society* **2**, 230–265, 1936–7.
10. A.M. Turing. Systems of logic based on ordinals. *Proc. London Math. Soc.* **45**, Part 3, pp. 161–228, 1939.