COMPUTATION WITH OPAQUE REPRESENTATION

STEVEN T. PIANTADOSI

ABSTRACT. We define and investigate *opaque regular languages (ORLs)*, which are formal languages over the alphabet of natural numbers that can be mapped symbol-by-symbol onto regular languages, but for which the mapping is not computable by a Turing machine. We define a decidability criterion, (S, A)-decidability, for ORLs and characterize a set of regular languages that ORLs can map onto. Given any regular language r in the set, we show how to construct a Turing machine to (S, A)-decide an ORL that maps onto r. We also prove that nontrivial ORLs are not recursive sets.

1. Introduction

Many areas of science are concerned with understanding the computational properties of certain biological, physical, and mathematical systems. Such research typically proceeds by constructing an isomorphism between the system being studied and an abstract computing device such as a Turing machine [1, 2, 3]. In much previous work, it has been the case that the isomorphism between the system under study and the abstract computing device was itself computable: given any state of the system, one can compute the "equivalent" state of, for example, the Turing machine.

Here we study the case when the isomorphism is not a computable function. Namely, within the framework of formal language theory, we construct a system that performs a non-trivial computation isomorphic to one performed by a Turing machine, but for which the isomorphism cannot be computed.

These results may be relevant to the philosophy of science, as they show there exist simple and elegant, but ultimately unreachable descriptions of certain computational systems. For example, one could know the input, output, and intermediate states used in a (S, A)-deciding some word in an ORL $\mathcal{N}$, without ever being able to compute the simple explanatory correspondence between $\mathcal{N}$ and the regular language it is isomorphic to. It is possible that a more complete understanding of computation with opaque representations may show limitations on our ability to understand computation in theoretical, physical, and biological systems.

2. Preliminaries

Fix a finite or infinite alphabet Λ and let $\mathbb{N}$ be the set of natural numbers. We denote the set of all finite strings over Λ by Λ^*. A *language* $\mathcal{N}$ over Λ is any subset of Λ^*. Suppose $\phi : \mathbb{N} \to \Lambda$ and define $\phi^* : \mathbb{N}^* \to \Lambda^*$ by $\phi^*(a_1 a_2 \dots a_k) = \phi(a_1)\phi(a_2)\dots\phi(a_k)$ so that ϕ^* maps strings over $\mathbb{N}$ to strings over Λ by mapping each element of the string by ϕ.

Definition 1. Let $\mathcal{N} \subseteq \mathbb{N}^*$ and $\phi : \mathbb{N} \to \Lambda$. If ϕ is not a computable function, we say that $\mathcal{N}$ is an *opaque representation of the language* $\phi^*(\mathcal{N}) \subseteq \Lambda^*$. If $\phi^*(\mathcal{N})$ is regular, we call $\mathcal{N}$ an *opaque regular language (ORL)*.

Definition 2. A *general automaton (GA)* X is a tuple

$$X = (\Lambda, S, s_0, \Delta, A)$$

where

(1) Λ is a set of symbols
(2) S is a set of states
(3) $s_0 \in S$ is an initial state
(4) $\Delta : S \times \Lambda \to S$
(5) $A \subseteq S$ is a set of accepting states

We can view a GA $X = (\Lambda, S, s_0, \Delta, A)$ as recognizing a language over Λ. For each string $i_1 i_2 \ldots i_n \in \Lambda^*$, X begins in state s_0 and uses Δ to compute the next state from the current state and the next input symbol. After all the input symbols have been read, X *accepts* $i_1 i_2 \ldots i_n$ if the final state reached is an element of A and *rejects* $i_1 i_2 \ldots i_n$ if the final state reached is in $S \setminus A$. Formally, we define $\Delta^* : \Lambda^* \to S$ by

$$\Delta^*(i_1 i_2 \ldots i_n) = \Delta(\Delta(\Delta(s_0, i_1), i_2), \ldots, i_n)$$

so that X accepts $i_1 i_2 \ldots i_n$ if $\Delta^*(i_1 i_2 \ldots i_n) \in A$ and X rejects $i_1 i_2 \ldots i_n$ if $\Delta^*(i_1 i_2 \ldots i_n) \in S \setminus A$. Note that so long as Δ is computable, Δ^* is also computable. The set of all strings accepted by X is called the *language of* X, which is denoted $\mathcal{L}_X$.

Example 1. *When S and Λ are finite, X is also known as a finite automaton, and $\mathcal{L}_X$ is a regular language.*

Note also that Δ^* "answers" whether w is in $\mathcal{L}_X$ by returning either a member of A or a member of $S \setminus A$. When A is a recursive set, we can view Δ^* as returning a yes or no. However, when A is not recursive, it is not possible to compute if $\Delta^*(w) \in A$ for any given w, even though, in a sense, Δ^* has still "answered" the question.

Definition 3. Let S be any set and $A \subseteq S$. Suppose $\Delta^* : \mathbb{N}^* \to S$ is a computable function and $\mathcal{L}_X \subset \mathbb{N}^*$ is a language. If $\Delta^*(w) \in A$ if and only if $w \in \mathcal{L}_X$, we say that Δ^* (S, A)-*decides* $\mathcal{L}_X$.

Thus, if Δ is a computable function, the GA $X = (\Lambda, S, s_0, \Delta, A)$ defines a computable function Δ^* which (S, A)-decides the language $\mathcal{L}_X$.

In this paper we show that there exists a certain class of ORLs that are (S, A)-decidable by computable functions for a non-recursive set A. Given a regular language $\mathcal{L}_X$ on the alphabet $\{0, 1\}$ that meets certain conditions, we next show how to construct a general automaton $Y = (\mathbb{N}, S, s_0, \Theta, A)$ with the following properties:

(1) Θ is a computable function such that Θ^* (S, A)-decides $\mathcal{L}_Y$.
(2) There exists a function $\phi : \mathbb{N} \to \{0, 1\}$ such that $\phi^*(w) \in \mathcal{L}_X$ if and only if $w \in \mathcal{L}_Y$.
(3) ϕ is not computable.

In other words, the following diagram commutes:

$$\begin{array}{ccc} \mathbb{N}^* & \xrightarrow{\Theta^*} & \mathbb{N} \\ \phi^* \big\downarrow & & \phi \big\downarrow \\ \{0,1\}^* & \xrightarrow{\Delta^*} & \{0,1\} \end{array}$$

This is a counterintuitive result since it shows that given $w \in \mathbb{N}^*$, a Turing machine can (S, A)-decide if $\phi^*(w)$ is in some given regular language over $\{0, 1\}$ without being able to compute what the binary string $\phi^*(w)$ is. Thus, one can perform computations on strings of natural numbers as though they were strings of binary digits, without being able to compute what bits the natural numbers represent.

3. Construction of an ORL (S, A)-decider

Let $T_1, T_2, T_3 \dots$ be some enumeration of Turing machines and let $\mathbf{H} = \{n \mid T_n \text{ halts}\}$ [5]. Define $\phi : \mathbb{N} \to \{0,1\}$ by

$$\phi(n) = \begin{cases} 1 \text{ if } n \in \mathbf{H} \\ 0 \text{ if } n \notin \mathbf{H}. \end{cases}$$

We also require two related functions, ϕ^* and $\vec{\phi}_k$. $\phi^* : \mathbb{N}^* \to \{0,1\}^*$ is defined as above by $\phi^*(n_1 n_2 \dots n_m) = \phi(n_1)\phi(n_2)\dots\phi(n_m)$. For each $k = 1, 2, \dots$ define $\vec{\phi}_k : \mathbb{N}^k \to \{0,1\}^k$ by $\vec{\phi}_k(a_1, a_2, \dots, a_k) = (\phi(a_1), \phi(a_2), \dots, \phi(a_k))$. Note that ϕ, ϕ^*, and $\vec{\phi}_k$ are not computable.

We now show how to perform certain computable functions on the natural numbers that "mimic" logical functions. Define $\hat{\wedge} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ by $\hat{\wedge}(n, m) = r$ such that T_r halts if and only if $n \in \mathbf{H}$ and $m \in \mathbf{H}$. We will typically write $\hat{\wedge}(n, m)$ as $n \,\hat{\wedge}\, m$. Note that $\hat{\wedge}$ is computable since we can construct a Turing machine that takes n and m as input and outputs the number r of a Turing machine such that T_r simulates T_n and T_m and T_r halts only if both T_n and T_m halt. Similarly, define $\hat{\vee} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ by $\hat{\vee}(n, m) = r$ such that T_r halts if and only if T_n or T_m halt. $\hat{\vee}(n, m)$ will also be written $n \,\hat{\vee}\, m$.

The functions $\hat{\wedge}$ and $\hat{\vee}$ make it possible to perform logical operations on the set $\mathbb{N}$ such that $\phi(n \,\hat{\wedge}\, m) = \phi(n) \wedge \phi(m)$ and $\phi(n \,\hat{\vee}\, m) = \phi(n) \vee \phi(m)$. That is, using $\hat{\wedge}$ and $\hat{\vee}$ we can compute boolean logical operations on the natural numbers, where the truth value of a natural number is determined by its membership in $\mathbf{H}$. Interestingly, $\hat{\wedge}$ and $\hat{\vee}$ perform these logical operations without "knowing" what boolean values their arguments represent.

We now define a special class of binary functions $\mathbb{F}_k$ for $k = 1, 2, \dots$. For the sake of brevity, we avoid a completely rigorous definition of $\mathbb{F}_k$ since Example (2) should make it clear which functions are in $\mathbb{F}_k$.

Definition 4. Let $\mathbb{F}_k$ be the class of binary functions $f : \{0,1\}^{k+1} \to \{0,1\}^k$ which can be written using only conjunction ($\wedge$) of arguments, disjunction ($\vee$) of arguments, and parentheses. These are sometimes called *monotone* binary functions.

Example 2. *Let $f : \{0,1\}^4 \to \{0,1\}^3$ be defined by*

$$f(a_1, a_2, a_3, a_4) = (a_4, (a_1 \wedge a_2) \vee (a_2 \wedge a_3), a_1 \vee (a_2 \wedge a_3)).$$

Then $f \in \mathbb{F}_3$.

Lemma 1. *If $f \in \mathbb{F}_k$, there exists a function $\Theta : \mathbb{N}^{k+1} \to \mathbb{N}^k$ such that $\vec{\phi}_k \circ \Theta = f \circ \vec{\phi}_{k+1}$.*

Proof. This should be evident from the fact that we can define Θ analogously to f, but with $\wedge$ replaced by $\hat{\wedge}$ and $\vee$ replaced by $\hat{\vee}$. The fact that $\phi(n \,\hat{\wedge}\, m) = \phi(n) \wedge \phi(m)$ and $\phi(n \,\hat{\vee}\, m) = \phi(n) \vee \phi(m)$ implies $\vec{\phi}_k \circ \Theta = f \circ \vec{\phi}_{k+1}$. □

Example 3. *Let $\Theta : \mathbb{N}^4 \to \mathbb{N}^3$ be defined by*

$$\Theta(a_1, a_2, a_3, a_4) = (a_4, (a_1 \,\hat{\wedge}\, a_2) \,\hat{\vee}\, (a_2 \,\hat{\wedge}\, a_3), a_1 \,\hat{\vee}\, (a_2 \,\hat{\wedge}\, a_3)).$$

Then $\vec{\phi}_3 \circ \Theta = f \circ \vec{\phi}_4$ where f is the function from Example (2).

Theorem 1. *Assume $Q \subseteq \{0,1\}^k$ for some k and $\Delta \in F_k$. If $X = (\{0,1\}, Q, s_0, \Delta, R)$ is a finite automaton, then there exists a general automaton $Y = (\mathbb{N}, S, q_0, \Theta, A)$ such that Θ is computable and $\phi^*(w) \in \mathcal{L}_X$ if and only if $w \in \mathcal{L}_Y$.*

Proof. Define $S = \phi_k^{-1}(Q)$ and $A = \phi_k^{-1}(R)$, where ϕ_k^{-1} denotes the set-theoretic inverse of ϕ_k. Set q_0 to any value such that $\vec{\phi}_k(q_0) = s_0$. Choosing such a q_0 is not problematic because we can always compute some set of Turing machines that do not halt and some set that do. Since $\Delta \in \mathbb{F}_k$, Lemma (1) implies we can choose Θ such that $\vec{\phi}_k \circ \Theta = \Delta \circ \vec{\phi}_{k+1}$. A word $i_1 i_2 \ldots i_{n-1} i_n \in \mathcal{L}_Y$ if and only if $\Theta^*(i_1 i_2 \ldots i_{n-1} i_n) \in A$ or equivalently $\vec{\phi}_k(\Theta^*(i_1 i_2 \ldots i_{n-1} i_n)) \in R$. But

$$
\begin{aligned}
&\vec{\phi}_k(\Theta^*(i_1 i_2 \ldots i_{n-1} i_n))\\
&\qquad = \vec{\phi}_k \circ \Theta(\Theta(\Theta(\ldots\Theta(\Theta(q_0, i_1), i_2), \ldots), i_{n-1}), i_n)\\
&\qquad = \Delta \circ \vec{\phi}_{k+1}(\Theta(\Theta(\ldots\Theta(\Theta(q_0, i_1), i_2), \ldots), i_{n-1}), i_n)\\
&\qquad = \Delta(\vec{\phi}_k \circ \Theta(\Theta(\ldots\Theta(\Theta(q_0, i_1), i_2), \ldots), i_{n-1}), \phi(i_n))\\
&\qquad = \Delta(\Delta(\vec{\phi}_k \circ \Theta(\ldots\Theta(\Theta(q_0, i_1), i_2), \ldots), \phi(i_{n-1})), \phi(i_n))\\
&\qquad \vdots\\
&\qquad = \Delta(\Delta(\Delta(\ldots\Delta(\Delta(q_0, \phi(i_1)), \phi(i_2)), \ldots), \phi(i_{n-1})), \phi(i_n))\\
&\qquad = \Delta^*(\phi^*(i_1 i_2 \ldots i_{n-1} i_n))
\end{aligned}
$$

which is an element of R if and only if $\phi^*(i_1 i_2 \ldots i_{n-1} i_n) \in \mathcal{L}_X$. This shows that $i_1 i_2 \ldots i_{n-1} i_n \in \mathcal{L}_Y$ if and only if $\phi^*(i_1 i_2 \ldots i_{n-1} i_n) \in \mathcal{L}_X$. Θ^* is a computable function on $\mathbb{N}^k$ since $\hat{\wedge}$ and $\hat{\vee}$ are computable. ☐

Thus, Θ^* is a computable function that (S, A)-decides $\mathcal{L}_Y$.

4. An example of an ORL

We begin by defining the finite automaton $X = (\{0,1\}, Q, s_0, \Delta, R)$ by setting $Q = \{0,1\}^2$, $R = \{(1,0),(1,1)\}$, and $s_0 = (0,1)$. Define $\Delta : \{0,1\}^2 \times \{0,1\} \to \{0,1\}^2$ by

$$\Delta(\overbrace{a_1, a_2}^{\text{state}}, \overbrace{\;i\;}^{\text{input}}) = (\overbrace{i \vee (a_1 \wedge a_2), a_2 \wedge i}^{\text{new state}}).$$

Note that $\Delta \in \mathbb{F}_2$ and Δ determines the following transitions mapping $Q \times \{0,1\} \to Q$:

$$
\begin{aligned}
\Delta(0,0,0) &= (0,0) & \qquad \Delta(0,0,1) &= (1,0)\\
\Delta(0,1,0) &= (0,0) & \qquad \Delta(0,1,1) &= (1,1)\\
\Delta(1,0,0) &= (0,0) & \qquad \Delta(1,0,1) &= (1,0)\\
\Delta(1,1,0) &= (1,0) & \qquad \Delta(1,1,1) &= (1,1)
\end{aligned}
$$

A graph representing X is shown in Figure 1.

To define $Y = (\mathbb{N}, S, q_0, \Theta, A)$, we begin by setting $S = \phi_2^{-1}(Q) = \mathbb{N}^2$ and $A = \phi_2^{-1}(R)$. That is, A consists of all ordered pairs $(n, m) \in \mathbb{N}^2$ such that $(\phi(n), \phi(m)) \in \{(1,0),(1,1)\}$. Set q_0 to any tuple (n, m) such that $(\phi(n), \phi(m)) = (0,1) = s_0$. We then define $\Theta : \mathbb{N}^2 \times \mathbb{N} \to \mathbb{N}^2$ by replacing every $\wedge$ with $\hat{\wedge}$ and $\vee$ with $\hat{\vee}$ in the definition of Δ:

$$\Theta(a_1, a_2, i) = (i \,\hat{\vee}\, (a_1 \,\hat{\wedge}\, a_2), a_2 \,\hat{\wedge}\, i).$$

Thus, Θ^* is a computable function that (S, A)-decides an opaque representation of the regular language shown in Figure 1.

Suppose, for example, that $\{T_{h_1}, T_{h_2}, T_{h_3}, \ldots\}$ were a set of machines that halted and $\{T_{r_1}, T_{r_2}, T_{r_3}, \ldots\}$ were a set of Turing machines that did not halt. Then we

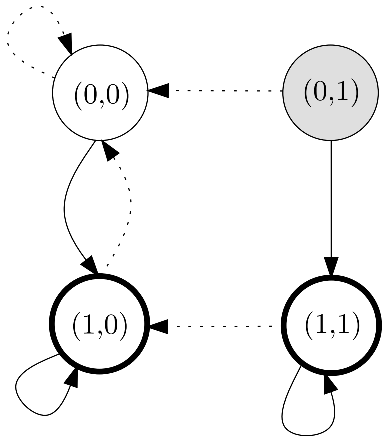

FIGURE 1. A finite automaton corresponding to $\mathcal{L}_X$. Here states are labeled with elements of $\{0,1\}^2$, the gray state is the start state, and bold states are accepting states. Solid arrows correspond to an input of 1 and dotted arrows correspond to an input of 0.

would have,

$$\begin{aligned}
h_1h_1 \in \mathcal{L}_Y \text{ since } \phi^*(h_1h_1) &= 11 &\in \mathcal{L}_X\\
h_1h_2 \in \mathcal{L}_Y \text{ since } \phi^*(h_1h_2) &= 11 &\in \mathcal{L}_X\\
h_1h_2r_1 \in \mathcal{L}_Y \text{ since } \phi^*(h_1h_2r_1) &= 110 &\in \mathcal{L}_X\\
h_1h_2r_1r_2h_3h_4 \in \mathcal{L}_Y \text{ since } \phi^*(h_1h_2r_1r_2h_3h_4) &= 110011 &\in \mathcal{L}_X
\end{aligned}$$

but

$$\begin{aligned}
r_1 \notin \mathcal{L}_Y \text{ since } \phi^*(r_1) &= 0 &\notin \mathcal{L}_X\\
r_1h_1r_2h_2r_3 \notin \mathcal{L}_Y \text{ since } \phi^*(r_1h_1r_2h_2r_3) &= 01010 &\notin \mathcal{L}_X\\
h_1h_2r_1r_2r_3r_4 \notin \mathcal{L}_Y \text{ since } \phi^*(h_1h_2r_1r_2r_3r_4) &= 110000 &\notin \mathcal{L}_X.
\end{aligned}$$

5. Limitations on Deciding ORLs

When $\phi^{-1}(\{0\})$ or $\phi^{-1}(\{1\})$ is recursively enumerable, as above, (S,A)-deciding an ORL $\mathcal{L}_Y$ for a non-recursive set A is, in a sense, the "best" we can do: we now show that if A were recursive and $\phi^*(\mathcal{L}_Y)$ were nontrivial, there is no computable function Δ^* that would (S,A)-decide $\mathcal{L}_Y$. In particular, this shows that with ϕ as above, ORLs that map onto non-trivial regular languages are not recursive sets.

Definition 5. A language $\mathcal{N} \subset \{0,1\}^*$ is *symbol dependent* if there exists words $x \in \mathcal{N}$ and $y \notin \mathcal{N}$ such that $x = a_1a_2\dots a_n\alpha a_{n+1}\dots a_m$ and $y = a_1a_2\dots a_n\beta a_{n+1}\dots a_m$ with $\alpha,\beta \in \{0,1\}$.

That is, $\mathcal{N}$ is symbol dependent if membership in $\mathcal{N}$ depends on the symbols in words rather than just the length of words. For example, $\mathcal{N} = \{w \in \Sigma^* : |w| = 0 \mod 2\}$ is not symbol dependent because for $n = 1,2,\dots$, every word of length $2n$ is in $\mathcal{N}$. If a language $\mathcal{N}$ is not symbol dependent, it is, for our purposes, "trivial" since the values of the symbols in a word have no influence on the word's membership in $\mathcal{N}$.

Theorem 2. *Suppose $\mathcal{N} \subseteq \mathbb{N}^*$ is an ORL and $\phi : \mathbb{N} \to \{0,1\}$ is such that either $\phi^{-1}(\{1\})$ or $\phi^{-1}(\{0\})$ is recursively enumerable. If $\phi^*(\mathcal{N})$ is symbol dependent and A is a recursive set, there is no computable function Δ^* that (S,A)-decides $\mathcal{N}$.*

Proof. Suppose such a Δ^* did exist. We will show how to compute ϕ, which is impossible. Suppose $f : S \to \{0,1\}$ is a computable function that decides A with $f(s) = 1$ if $s \in A$ and $f(s) = 0$ if $s \notin A$.

Since $\phi^*(\mathcal{N})$ is symbol dependent, there exist $x \in \phi^*(\mathcal{N})$ and $y \notin \phi^*(\mathcal{N})$ such that $x = a_1 a_2 \ldots a_n \alpha a_{n+1} \ldots a_m$ and $y = a_1 a_2 \ldots a_n \beta a_{n+1} \ldots a_m$ with $\alpha, \beta \in \{0,1\}$.

Suppose we are given some number $\gamma \in \mathbb{N}$. It is possible to recursively enumerate some subset of Turing machines that halt and some subset that do not halt. Therefore, we can construct a word $w_\gamma = b_1 b_2 \ldots b_n \gamma b_{n+1} \ldots b_m$ such that $\phi(b_i) = a_i$ for each i. We can then compute $f(\Delta^*(w_\gamma))$ and if $f(\Delta^*(w_\gamma)) = 1$, we know $\phi(\gamma) = \alpha$. If $f(\Delta^*(w_\gamma)) = 0$, we know $\phi(\gamma) = \beta$. □

This theorem implies that if we wish to study any interesting opaque languages, with ϕ defined as above, we must restrict ourselves to (S, A)-decidable languages where A is not recursive. The theorem might be paraphrased as "ask an opaque question, get an opaque answer."

6. Further directions

It is possible to perform similar computations with other logical gates if ϕ is chosen differently. For example, let $B : \mathbb{N} \to \mathbb{N}$ be the Busy Beaver function [4] and denote the k'th prime by p_k. If $n = p_{a_1} p_{a_2} \ldots p_{a_k}$ is the prime factorization of n, define

$$\phi(n) = \begin{cases} 1 \text{ if the cardinality of } \{i \mid a_i \in B(\mathbb{N})\} \text{ is odd} \\ 0 \text{ otherwise.} \end{cases}$$

Clearly ϕ is not computable, and with this choice of ϕ it is easy to construct an XOR gate on $\mathbb{N}$: define $XOR(n,m) = n \cdot m$. If $\oplus$ denotes the usual boolean exclusive-or function, then $\phi(XOR(n,m)) = \phi(n) \oplus \phi(m)$ and one could carry out a construction similar to that in Section 3. However, with this choice of ϕ there is no way of computing AND and OR. Determining precisely which logical gates allow computation with opaque representations is an open problem. Solving this problem would help to characterize which languages are decidable in opaque representations. Indeed, it is not clear if there exist opaque languages that can map onto any more complex class of languages than regular languages.

References

[1] Berger, R.: The undecidability of the Domino Problem. Memoirs of the American Mathematical Society **66**, 1-72 (1966)

[2] Block, Ned.: "The mind as software of the brain." In *An Invitation to Cognitive Science v. 3: Thinking*, ed. by Smith, E., Osherson, D., (MIT Press, Cambridge Massachusetts 1995) pp 377-425

[3] Cook, M.: Universality in elementary cellular automata. Complex Systems **15**, 1-40 (2004)

[4] Radó, T.: On non-computable functions. Bell Systems Tech. J. **41(3)**, 877-844 (1962)

[5] Turing, A.: On computable numbers, with an application to the Entscheidungsproblem. Proceedings of the London Mathematical Society Series 2 **42,** (1937)