On Computability and Learnability of the Pumping Lemma Function*

Dariusz Kalociński†

March 2014

Abstract

On the basis of the well known pumping lemma for regular languages we define such a partial function $f : \mathbb{N} \to \mathbb{N}$ that for every e it yields the least pumping constant for the language W_e. We ask whether f is computable. Not surprisingly f turns out to be non-computable. Then we check whether f is algorithmically learnable. This is also proved not to be the case. Further we investigate how powerful oracle is necessary to actually learn f. We prove that f is learnable in $0'$. We also prove some facts relating f to arithmetical hierarchy.

Keywords: pumping lemma, computability, algorithmic learning, arithmetical hierarchy, reducibility

1 The Pumping Lemma Function

From automata theory one knows the pumping lemma for regular languages [4]. Before we formulate the lemma, let us fix some terminology. Let W_e denote the domain of the partial function computed by the Turing machine

*This is a draft of: D. Kalociński. On Computability and Learnability of the Pumping Lemma Function. In Adrian-Horia Dediu, Carlos Martín-Vide, José-Luis Sierra-Rodríguez, and Bianca Truthe, editors, *Language and Automata Theory and Applications*, volume 8370 of *Lecture Notes in Computer Science*, pages 433-440. Springer International Publishing, 2014.

†The author is a PhD student at the Department of Logic, Institute of Philosophy, University of Warsaw, Poland.

with Gödel number e. By $R(e,c)$ we understand the following statement: *for each word* $\omega \in W_e$*, if* $|\omega| > c$*, then there are words* α,β, γ *such that* $\omega = \alpha\beta\gamma$, $\beta \neq \varepsilon$, $|\alpha\beta| \leq c$ *and for all* $i \in \mathbb{N}$ $\alpha\beta^i\gamma \in W_e$. We may now formulate the pumping lemma for regular languages easily: for each e such that W_e is a regular language there is a positive integer c such that $R(e,c)$. On the basis of this familiar result, one can define a partial function $f : \mathbb{N} \to \mathbb{N}$ (here called the pumping lemma function) such that for each $e \in \mathbb{N}$ yields the least c such that $R(e,c)$, if any such c exists, and otherwise is undefined. A natural question arises: is f computable? The solution to this question is negative. Another problem which arises is as follows: is the complement of the graph of f recursively enumerable? This is also proved not to be the case. Further we investigate algorithmic learnability of f. We prove that f is not algorithmically learnable. Then we ask how powerful oracle is necessary to actually learn f. We prove that f is algorithmically learnable with the halting problem in oracle. We finish the article by relating f to arithmetical hierarchy.

2 Terminology and Notation

In this section we provide terminology and notation used throughout the article. Notions used locally are defined when they are needed. For further details on computability consult [1, 6].

In Sect. 1 we have already introduced some notation, namely W_e and $R(e,c)$. Sometimes we use R for the binary relation expressed by the formula $R(e,c)$. The appropriate meaning of R shall be clear from the context. By G_h we denote the graph of the (possibly partial) function h. The operation of taking the complement of a relation $S \subseteq \mathbb{N}^k$ is defined as follows: $S' = \mathbb{N}^k - S$. Inputs of algorithms are words or numbers. We can assume that algorithms are Turing machines that work with words over binary alphabet. Binary words are easily coded as numbers. We do not make any explicit distinction between numeric inputs and string inputs - a particular usage will be clear from context. The length of the word x is denoted by $lh(x)$ or $|x|$. By $\leq_{\mathrm{bl}}$ we denote bounded lexicographical order on strings. Let x, y be words. We say x is less or equal to y with respect to bounded lexicographical order (in symbols $x \leq_{\mathrm{bl}} y$), if $|x| < |y|$ or both $|x| = |y|$ and x is lexicographically less or equal to y. The characteristic function of the set $A \subseteq \mathbb{N}$ is denoted by c_{A}. By coding of pairs we mean some fixed reasonable coding, for example Cantor

pairing function. By π_i, $i = 1, 2$, we mean the canonical projection of a pair on thie i-th coordinate. The symbol $\leq_{\mathrm{m}}$ refers to the relation of many-one reducibility. We write $h : A \leq_{\mathrm{m}} B$ to express the fact that h is total recursive and $x \in A \Leftrightarrow h(x) \in B$, for all $x \in \mathbb{N}$. We use $T(e, x, c)$ for the Kleene predicate, where e is a Gödel number of a Turing machine, x stands for an input and c for a computation. $U(c)$ refers to the output of a computation c. By EMPTY we denote the emptiness problem, i.e the set $\{e \in \mathbb{N} : W_e = \emptyset\}$. $\mathsf{NOTEMPTY}$ stands for the non-emptiness problem, i.e. the set $\mathbb{N} - \mathsf{EMPTY}$. TOT denotes the totality problem, i.e. $\{e : \forall\, x \exists\, c T(e, x, c)\}$. The halting problem $\{(e, x) \in \mathbb{N}^2 : \exists\, c T(e, x, c)\}$ is denoted by HALT. We use the standard notation Σ^0_k, Π^0_k, Δ^0_k for the classes of sets in arithmetical hierarchy.

3 Non-computability Results

Lemma 1. $\mathsf{EMPTY} \leq_m R$.

Proof. We define the function $r(e) = (\sigma(e), 1)$, where σ is the total computable function obtained through *smn* theorem from $g(e, x)$ which is computed as follows: We examine whether W_e is empty. If it is empty, the computation goes on forever; otherwise emptiness checking procedure stops. In that case we measure the length of x and if it is even we return 1. Otherwise we loop forever.

Therefore if $W_e = \emptyset$, then $W_{\sigma(e)} = \emptyset$ and $(\sigma(e), 1) \in R$. If $W_e \neq \emptyset$ then $W_{\sigma(e)}$ contains all words of even length. In this case clearly $(\sigma(e), 1) \notin R$ (otherwise W_e would have contained words of odd length). □

Lemma 2. *If* $R(e, c)$ *then* $(\forall\, d > c)\, R(e, d)$.

Proof. Directly from the definition of $R(e, c)$. □

Theorem 1. *f is not computable.*

Proof. Suppose for the sake of contradiction that f is computable. Then of course R is recursively enumerable, that is Σ^0_1 (use the fact that the graph of recursive function is r.e. and, bearing in mind the Lemma 2, devise an algorithm for enumerating R). Taking into account that EMPTY is Π^0_1-complete, we have $A \leq_{\mathrm{m}} \mathsf{EMPTY} \leq_{\mathrm{m}} R \in \Sigma^0_1$ for all $A \in \Pi^0_1$. It follows that $\Pi^0_1 \subseteq \Sigma^0_1$, which is impossible, because it is well known that $\Pi^0_1 - \Sigma^0_1 \neq \emptyset$. □

Lemma 3. $\mathsf{NOTEMPTY} \leq_m R$.

Proof. We define the function $r(e) = (\sigma(e), 1)$, where σ is the total computable function obtained through *smn* theorem from $g(e, x)$ which is computed as follows. If the length of x is even, then stop. Otherwise start checking, whether $W_e \neq \emptyset$. If $W_e \neq \emptyset$ then - when a word α is found such that $\alpha \in W_e$ - stop. Otherwise (if $W_e = \emptyset$) we loop forever.

Therefore if $W_e = \emptyset$ then $W_{\sigma(e)}$ contains all words of even length, so $(\sigma(e), 1) \notin R$. If $W_e \neq \emptyset$, then $W_{\sigma(e)}$ contains all words and $(\sigma(e), 1) \in R$. □

Lemma 4. *If G'_f is r.e., then R' is r.e.*

Proof. Let $p : \mathbb{N}^2 \to \mathbb{N}$ be the recursive partial characteristic function of G'_f, i.e.

$$p(x, y) = \begin{cases} 1 & \text{if } (x, y) \notin G_f \\ \text{undefined} & \text{otherwise} \end{cases} . \tag{1}$$

We define $h : \mathbb{N}^2 \to \mathbb{N}$ - the recursive partial characteristic function of R':

$$h(x, y) = \Pi_{i=0}^{y} p(x, i) \ . \tag{2}$$

Assume $(x, y) \in R'$. Then it must be the case that $(\forall\, i \leq y)\,(x, i) \in R'$. Therefore $(\forall\, i \leq y)\,(x, i) \notin G_f$ and thus $(\forall\, i \leq y)\, p(x, i) = 1$.

Now assume $(x, y) \notin R'$. Then $(x, y) \in R$. So it must be the case that $f(x) \leq y$. Therefore $\Pi_{i=0}^{y} p(x, i)$ is undefined, since $p(x, f(x))$ is undefined and $0 \leq f(x) \leq y$. □

Theorem 2. *G'_f is not r.e.*

Proof. Suppose on the contrary that G'_f is r.e. Then by Lemma 4 R' is r.e. We use the following fact from recursion theory: $A \leq_{\mathrm{m}} B \Leftrightarrow A' \leq_{\mathrm{m}} B'$. We apply it to $\mathsf{NOTEMPTY} \leq_{\mathrm{m}} R$ and obtain $\mathsf{EMPTY} \leq_{\mathrm{m}} R'$. Then the reasoning is analogous to the proof the Theorem 1. □

4 Non-learnability Result

We established some lower bounds on the complexity of the pumping lemma function: we know that f is not recursive and that G'_f is not r.e. In this section we show that f is not algorithmically learnable which means that G_f is not Σ_2^0.

Definition 1. *Let $f : \mathbb{N}^k \to \mathbb{N}$ be a (possibly partial) function. We say that f is algorithmically learnable (shortly: learnable) if there is a total computable function $g_t(\overline{x})$[1] such that for all $\overline{x} \in \mathbb{N}^k$:*

$$lim_{t\to\infty} g_t(\overline{x}) = f(\overline{x}) \ , \tag{3}$$

which means that neither $f(\overline{x})$ nor $lim_{t\to\infty} g_t(\overline{x})$ exist or - alternatively - both $f(\overline{x})$ and $lim_{t\to\infty} g_t(\overline{x})$ exist and are equal.

The following lemma is a familiar result from the algorithmic learning theory. Classical papers related to the subject are [2, 3, 5].

Lemma 5. *Let $f : \mathbb{N}^k \to \mathbb{N}$. f is learnable if and only if G_f is Σ^0_2.*

Proof. ($\Rightarrow$) Let f be learnable and $g_t(\overline{x})$ be total computable function satisfying the Equation (3). We observe that $G_f(\overline{x}, y) \Leftrightarrow (\exists\, t \in \mathbb{N}\,)(\forall\, k > t)\, y = g_k(\overline{x})$. The formula $y = g_k(\overline{x})$ defines a recursive ternary relation, because g is total computable.

($\Leftarrow$) Let f be such that G_f is Σ^0_2. Choose a recursive relation $A \subseteq \mathbb{N}^{k+3}$ such that $G_f(\overline{x}, y) \Leftrightarrow \exists z \forall w A(\overline{x}, y, z, w)$. We define an infinite procedure $G(\overline{x})$ (Alg. 1) that is easily convertible to the appropriate definition of total computable learning function $g_t(\overline{x})$ satisfying Equation 3.

The procedure simply searches for y satisfying $\exists z \forall w A(\overline{x}, y, z, w)$ by enumerating all possible pairs (y, z). If there is such y, the procedure will finally spot it together with the relevant witness z. Therefore $f(\overline{x})$ will be finally stored in the variable y and from that point the contents of y will never change. On the other hand, if there is no such y, the contents of the variable y will continue to change ad infinitum. □

Lemma 6. $\mathsf{TOT} \leq_m R$.

Proof. Let $H(x, m, t)$ mean $(\exists\, c < t\,)T(x, m, c)$. Define a relation $S(x, y)$:

$$S(x, y) \Leftrightarrow_{\mathrm{df}} 2 \,|\, lh(y) \vee (\exists\, t\,)(\forall\, m \leq_{\mathrm{bl}} y)\, H(x, m, t) \ . \tag{4}$$

Of course, S is r.e. By $p_{\mathrm{S}}(x, y)$ we denote recursive partial characteristic function of S. By *smn* theorem, there is a total computable function σ such that $\{\sigma(x)\}(y) \simeq p_{\mathrm{S}}(x, y)$, where $\{\cdot\}$ refers to the function computed by

[1]Expression $g_t(\overline{x})$ shall be read as $g(t, \overline{x})$. In precise terms, $g_t(\overline{x})$ stands for a sequence of functions. We use indexed t to distinguish the discrete time parameter from the input.

Data: $\overline{x}$
Result: y contains hypothesized value $f(\overline{x})$
$p, w, y, z \leftarrow 0$;
while *true* **do**
  $y \leftarrow \pi_1(p)$;
  $z \leftarrow \pi_2(p)$;
  if $c_A(\overline{x}, y, z, w) = 1$ **then**
    $w \leftarrow w + 1$;
  else
    $p \leftarrow p + 1$;
    $w \leftarrow 0$;
  end
end

Algorithm 1: The infinite procedure $G(\overline{x})$.

Turing machine having Gödel number $\cdot$. Define $r(x) =_{\mathrm{df}} (\sigma(x), 1)$. We prove that $r : \mathsf{TOT} \leq_{\mathrm{m}} R$.

($\Rightarrow$) Let $x' \in \mathsf{TOT}$. We begin by showing that $S(x', y)$ holds for all y. Fix y'. If $2 \,|\, lh(y')$, then obviously $S(x', y')$. Assume that $2 \nmid lh(y')$. Because $x' \in \mathsf{TOT}$, we choose a finite sequence of numbers $(t_\omega)_{\omega \leq_{\mathrm{bl}} y'}$ such that $H(x', \omega, t_\omega)$ holds for $\omega \leq_{\mathrm{bl}} y'$. Let $T = max\{t_\omega : \omega \leq_{\mathrm{bl}} y'\}$. Then we have $(\forall\, \omega \leq_{\mathrm{bl}} y')\, H(x', \omega, T)$, and - what follows - $(\exists\, t\,)(\forall\, \omega \leq_{\mathrm{bl}} y')\, H(x', \omega, t)$. Therefore $S(x', y')$.

The following are equivalent:
$(\forall\, y)\, S(x', y)$,
$(\forall\, y)\, p_{\mathrm{S}}(x', y) = 1$,
$(\forall\, y)\, \{\sigma(x')\}(y) = 1$,
$\sigma(x') \in \mathsf{TOT}$.

It remains to show that $r(x') = (\sigma(x'), 1) \in R$, which is trivially true.

($\Leftarrow$) Let $x' \notin \mathsf{TOT}$. Choose y_0 to be the smallest word with respect to $\leq_{\mathrm{bl}}$ such that $\neg(\exists\, c\,)T(x', y_0, c)$. Observe that $(\forall\, y)\, [y_0 \leq_{\mathrm{bl}} y \wedge 2 \nmid lh(y) \Rightarrow \neg S(x', y)]$. For let $y_0 \leq_{\mathrm{bl}} y$, $2 \nmid lh(y)$ and suppose $S(x', y)$. It follows, that $(\exists\, t\,)H(x', y_0, t)$ which is a contradiction.

Let k be any number satisfying $lh(y_0) < 2k$. Observe that $1^{2k} \in W_{\sigma(x')}$, because $2 \,|\, lh(1^{2k})$. The only possible division of 1^{2k} into $\alpha\beta\gamma$ satisfying conditions $lh(\alpha\beta) \leq 1$, $\beta \neq \varepsilon$ is $\alpha = \varepsilon$, $\beta = 1$, $\gamma = 1^{2k-1}$. Consider $\alpha\beta^0\gamma = 1^{2k-1}$. Clearly $y_0 \leq_{\mathrm{bl}} 1^{2k-1}$. Therefore $\neg S(x', 1^{2k-1})$ and thus

$1^{2k-1} \notin W_{\sigma(x')}$. We may then conclude that $r(x') = (\sigma(x'), 1) \notin R$. □

We are ready to prove the main non-learnability theorem.

Theorem 3. *f is not learnable.*

Proof. Suppose for the sake of contradiction that f is learnable. By the Lemma 5, G_f is Σ_2^0. R can be defined as follows: $R(x, y) \Leftrightarrow \exists c(G_f(x, c) \wedge c \leq y)$. The right side of the equivalence is easily convertible to a Σ_2^0-formula. Thus R is Σ_2^0. Due to the Lemma 6, TOT is Σ_2^0. However, TOT is not Σ_2^0. Contradiction. □

5 Learnability Result

So far we have proved only negative results concerning computability or learnability of f. The question now rises how much we would need to strengthen our computational capabilities to turn this task into something learnable. In terms of recursion theory: how complex oracle we need to make f learnable? In this section we prove that f is learnable in $0'$.

Theorem 4. *f is learnable in $0'$.*

Proof. We choose $\mathsf{HALT} = \{(e, x) \in \mathbb{N}^2 : (\exists\, c)T(e, x, c)\}$ for the oracle. By $\phi(e, x)$ we denote the formula provided below. The formula $\phi(e, x)$ expresses the fact that $R(e, x)$:

$$x > 0 \ \wedge\ (\forall\, \omega)\, \{\overbrace{[(e, \omega) \in \mathsf{HALT} \ \wedge\ lh(\omega) \geq x]}^{\sigma(x,e,\omega)} \Rightarrow (\exists\, \alpha, \beta, \gamma \leq_{\mathrm{bl}} \omega\,) \\ [\underbrace{\alpha\beta\gamma = \omega \ \wedge\ lh(\alpha\beta) \leq x \ \wedge\ \beta \neq \varepsilon}_{\theta_1(x,\omega,\alpha,\beta,\gamma)} \wedge (\forall\, i)\ \underbrace{(e, \alpha\beta^i\gamma) \in \mathsf{HALT})}_{\theta_2(e,\alpha,\beta,\gamma,i)}]\} \ . \tag{5}$$

Observe that the relation $\{(x, \omega, e, \alpha, \beta, \gamma, i) : \theta_1 \wedge \theta_2\}$ is recursive in HALT. There is a relation η, recursive in HALT, such that $(\exists\, \alpha, \beta, \gamma \leq_{\mathrm{bl}} \omega\,)(\forall\, i)\,(\theta_1 \wedge \theta_2) \Leftrightarrow (\forall\, j)\,\eta(x, e, \omega, j)$.

The following are equivalent:

$\phi(e, x)$,

$x > 0 \wedge (\forall\, \omega)\,\{\sigma \Rightarrow (\exists\, \alpha, \beta, \gamma \leq_{\mathrm{bl}} \omega\,)[\theta_1 \wedge (\forall\, i)\,\theta_2]\}$,

$(\forall\, \omega)\,\{x > 0 \wedge [\sigma \Rightarrow (\exists\, \alpha, \beta, \gamma \leq_{\mathrm{bl}} \omega\,)(\forall\, i)\,(\theta_1 \wedge \theta_2)]\}$,

$(\forall\, \omega)\,\{x > 0 \wedge [\sigma \Rightarrow (\forall\, j)\,\eta]\}$,

$$(\forall\,\omega)\,(\forall\,j)\,\{\underbrace{(x>0\ \wedge\ (\neg\sigma\vee\eta)}_{\xi(x,e,\omega,j)}\}\ .$$

Thus, the relation defined by $\phi(e,x)$ is Π^0_1 in HALT. Now we express the fact that x is the least number such that $\phi(e,x)$:

$$\phi_{\inf}(e,x) := \phi(e,x)\ \wedge\ (\forall\,y<x)\,\neg(\forall\,\omega)\,(\forall\,j)\,\xi(e,x,\omega,j)\ . \tag{6}$$

The right conjunct of $\phi_{\inf}$ is equivalent to a formula of the form $(\exists\,z\,)\zeta(e,x,z)$, where $\zeta(e,x,z)$ is recursive in HALT. Therefore we have

$$\phi_{\inf}(e,x) \Leftrightarrow (\forall\,\omega,j)\,\xi(e,x,\omega,j)\ \wedge\ (\exists\,z\,)\zeta(e,x,z)\ . \tag{7}$$

This easily leads us - by familiar first-order transformations - to the Σ^0_2 definition of the relation expressed by $\phi_{\inf}(e,x)$, in terms of relations recursive in HALT. Observe that the relation defined by $\phi_{\inf}(e,x)$ is G_f. Thus, G_f is Σ^0_2 in HALT. By the relativized version of the Lemma 5 f is learnable in HALT. □

6 Supplement

We end the article by providing supplementary results relating f to arithmetical hierarchy.

Theorem 5. *R is Π^0_2-complete.*

Proof. We have already proved that $\mathsf{TOT} \leq_{\mathrm{m}} R$ (Lemma 6). It remains to show that R is Π^0_2. Consider the following definition of R:

$$\begin{aligned} x>0\ \wedge\ (\forall\,\omega)\,\{[(\exists\,c\,)T(e,\omega,c)\ \wedge\ \overbrace{lh(\omega)\geq x}^{\sigma(x,\omega)}]\Rightarrow(\exists\,\alpha,\beta,\gamma\leq_{\mathrm{bl}}\omega\,)\\ [\underbrace{\alpha\beta\gamma=\omega\ \wedge\ lh(\alpha\beta)\leq x\ \wedge\ \beta\neq\varepsilon}_{\theta(x,\omega,\alpha,\beta,\gamma)}\wedge(\forall\,i)\,(\exists\,c\,)T(e,\alpha\beta^i\gamma,c)]\}\end{aligned} \tag{8}$$

Consider the implication enclosed in curly brackets:

$$((\exists\,c\,)T(e,\omega,c)\ \wedge\ \sigma)\Rightarrow(\exists\,\alpha,\beta,\gamma\leq_{\mathrm{bl}}\omega\,)(\theta\ \wedge\ (\forall\,i)\,(\exists\,c\,)T(e,\alpha\beta^i\gamma,c)) \tag{9}$$

We proceed by equivalent reformulations of (9):

$$(\forall\,c)\,(\neg T(e,\omega,c)\vee\neg\sigma)\vee(\exists\,\alpha,\beta,\gamma\leq_{\mathrm{bl}}\omega\,)(\theta\ \wedge\ (\forall\,i)\,(\exists\,c\,)T(e,\alpha\beta^i\gamma,c))\ \ ,$$

$$(\forall\, c)\,(\neg T(e,\omega,c) \vee \neg\sigma) \vee \underbrace{(\exists\, \alpha,\beta,\gamma \leq_{\mathrm{bl}} \omega\,)(\forall\, i)\,(\exists\, d\,)(\theta \wedge T(e,\alpha\beta^i\gamma,d))}_{\varphi}\ ,$$

$$(\forall\, c)\,(\neg T(e,\omega,c) \vee \neg\sigma) \vee \underbrace{(\forall\, i)\,(\exists\, d\,)\psi(x,e,\omega,i,d)}_{\varphi'}\ ,$$

$$(\forall\, c)\,(\forall\, i)\,(\exists\, d\,)(\neg T(e,\omega,c) \vee \neg\sigma \vee \psi(x,e,\omega,i,d))\ .$$

Thus, (8) is equivalent to:

$$(\forall\, \omega)\,(\forall\, c)\,(\forall\, i)\,(\exists\, d\,)[x > 0 \wedge (\neg T(e,\omega,c) \vee \neg\sigma \vee \psi(x,e,\omega,i,d))]\ . \qquad (10)$$

We have used familiar first order tautologies and the fact that bounded quantifier prefix in φ can be somehow shifted inside, resulting in an equivalent Π^0_2-formula φ' such that its subformula ψ expresses a recursive relation. The above argument clearly shows that R is Π^0_2. □

Theorem 6. *G_f is Δ^0_3.*

Proof. Let us denote by $\phi(e,x)$ the Π^0_2-formula (10) defining R. As in the proof of the Theorem 4 define G_f in the following way:

$$\phi(e,x) \wedge (\forall\, y < x)\,\neg\phi(e,y) \qquad (11)$$

The formula (11) is equivalent to a formula of the form $(\forall\exists\ldots \wedge \exists\forall\ldots)$, where $\ldots$ stand for some formulae expressing recursive relations. Proper shifts of quantifiers lead us to Π^0_3- and Σ^0_3-formula defining G_f. This clearly shows that G_f is Δ^0_3. □

7 Remarks about Practical Significance

We addressed the problem of determining the least constant from the pumping lemma with a view to applying results to formal language learning framework.

One of the directions for further investigation can be as follows. Consider the machine placed in an unknown environment. The environment presents positive and negative examples of an unknown language L from an unknown class. Note that the environment that exhaustively presents both positive and negative examples can be viewed as an oracle for the input language L. Thus the Theorem 4 may be applied, since in such an environment the machine is equipped with an analogue of the halting problem and the only queries to the halting problem that are important for determining the least constant for L are of the form ,,$\alpha \in L$?". Suppose the machine is equipped with the

learning procedure for f as described above. The pumping lemma for regular languages gives a necessary condition for a language to be regular. Let the input language $L = W_e$, for some e. If $\neg\exists c R(e, c)$, then W_e is not regular and the learning procedure for f diverges. This fact can be used as a heuristic and the machine can hypothesize that the input language is not regular. On the other hand, if $\exists c R(e, c)$, then the learning procedure for f converges and the machine can use this fact as a heuristic and conjecture that input language is regular or at least exclude certain languages from consideration.

We can put further constraints on input languages to make the applications more practical. Since for every regular language L there is a Turing machine e that computes the characteristic function c_L in linear time, we can restrain the working time of input machine e by a suitable quadratic polynomial p, with no worry of omitting any regular language. If the computation of e on the input α does not stop after $p(|\alpha|)$ steps, the answer to ,,$\alpha \in W_e$?" is set to *no*. By including such time constraints, we in fact obtain the learning algorithm of the pumping lemma function for languages decidable in quadratic polynomial time. In this setting, the learning algorithm may be used as a supplementary heuristic for hypothesizing whether input language is regular.

References

[1] N. Cutland. *Computability, an Introduction to Recursive Function Theory*. Cambridge University Press, 1980.

[2] E. M. Gold. Limiting Recursion. *J. Symbolic Logic*, 30:28–48, 1965.

[3] E. M. Gold. Language Identification in the Limit. *Information and Control*, 10:447–474, 1967.

[4] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Cambridge, 1979.

[5] H. Putnam. Trial and Error Predicates and the Solution to a Problem of Mostowski. *J. Symbolic Logic*, 30:49–57, 1965.

[6] J. R. Schoenfield. *Recursion Theory*, volume 1 of *Lecture Notes in Logic*. Springer-Verlag, Berlin, 1993.