EasyChair Preprint

№ 7718

New Languages of Abstract Automata

Mark Burgin

April 4, 2022

New languages of abstract automata

Mark Burgin
UCLA, Los Angeles, 90095, USA

Abstract. The conventional theory of automata and algorithms associates only one language with each automaton or algorithm. Here it is demonstrated that each automaton or algorithm determines several algorithmic languages on different levels of computability. Properties of these languages and relations between them and conventional languages of automata or algorithms are studied. The obtained results made possible the discovery of a new class of algorithmic languages in addition to the well-known recursively enumerable and recursively coenumerable languages. The new languages are called recursively bienumerable comprising both recursively enumerable and recursively coenumerable languages.

Keywords: algorithmic language, finite automaton, Turing machine, recursively enumerable language, recursively coenumerable language, logic

1. Introduction

In the theory of algorithms, automata and computation, a definite language of an automaton (algorithm) is defined (cf., for example, (Cohen, 1991; Sipser, 1997; Burgin, 2005)). For an accepting automaton (algorithm) *A*, this language consists of all words accepted (recognized) by *A*. For a computing automaton (algorithm) *A*, its language consists of all words computed by *A*. These languages show what abstract automata can do defining their computing or accepting (recognizing) power.

However, a more detailed analysis of computation shows that it is natural to associate more languages with each automaton or algorithm. Indeed, given a word *w* in the alphabet of an accepting automaton (algorithm) *A*, this word can be accepted (recognized) by *A*, rejected by *A*, both accepted and rejected by *A*, and it is possible that it is undefined whether the word is accepted or rejected. This partition of words into four categories gives birth to four types of the algorithmic languages of automata and algorithms:

- The *upper* or *positive language* $L_+(A)$ of an automaton (algorithm) A consists of all words accepted (recognized) by A
- The *lower* or *negative language* $L_-(A)$ of an automaton (algorithm) A consists of all words rejected (negatively recognized) by A
- The *impartial language* $L_0(A)$ of an automaton (algorithm) A consists of all words computationally undefined by A
- The *double language* $L_\pm(A)$ of an automaton (algorithm) A consists of all words both accepted (recognized) and rejected (unrecognized) by A

Two latter cases contradict classical European logic, according to which each statement can be either true or false, that is, in our case, a word can be either accepted or rejected (cf., for example, (Church, 1956)). In the case of automaton languages, it will give two languages – upper (positive) and lower (negative), from which only upper languages have been studied in the theory of automata and computation.

Contemporary intuitionistic logic added one more truth value. As the result, intuitionistic logic uses three truth values: true, false and undefined (Dummett, 1973). This correlates with computations of Turing machines when a word can be accepted, rejected or neither of this is true (cf., for example, (Cohen, 1991)). Going even further, different directions of ancient Indian logic allow four types of truth values. Namely, a statement can be: (1) true, (2) false, (3) neither true or false, and (4) both true and false (Ganeri, 2001; 2004; Chi, 1969).

For instance, one of the most prominent Buddhist philosophers Nagarjuna (ca. 150-250), who founded the madhyamaka school of the Buddhist philosophy, maintained that paradox in the following terms: "all phenomena are empty and so ultimately have no nature. But emptiness is, therefore, the ultimate nature of things. So [phenomena] both have and lack an ultimate nature" (Garfield, 2002). This means that the statement that all phenomena have no nature is true and false at the same time being *transconsistent*.

In this paper, we study the described types of algorithmic languages for different classes of algorithms and automata, relations between them, and connections between the languages, algorithms and automata. In section 2, finite automata and their algorithmic languages are analyzed. In section 3, Turing machines and their algorithmic languages are explored. An interesting phenomenon is that nondeterministic Turing machines determine even more than four

considered above types of algorithmic languages. In Conclusion, we analyze the obtained results and suggest directions of future research.

2. Languages of finite automata

In what follows, we assume that all automata and algorithms work with words in the alphabet Σ while Σ^* denotes the set of all words in this alphabet.

Let us consider algorithmic languages of accepting finite automata.

When A is a deterministic finite automaton, then given a word w as the input to A, it consumes any word w ending either in a final state or not in a final state (Cohen, 1991; Sipser, 1997). In the first case, w is accepted while in the second case, w is rejected. This gives us the following result.

Theorem 2.1. For any deterministic finite automaton A, we have $L_0(A) = L_{\pm}(A) = \varnothing$ and $L_+(A) \cup L_-(A) = \Sigma^*$.

The conventional approach tells that the standard language $L(A)$ of a deterministic finite automaton A is $L_+(A)$ and it is a regular language (Cohen, 1991; Sipser, 1997).

Taking a deterministic finite automaton A and making all its final states not final and not final states final, we come to the following result the deterministic Duality Theorem because the new automation is also finite and deterministic.

Theorem 2.2. For any deterministic finite automaton A, there is a deterministic finite automaton B such that $L_+(B) = L_-(A)$ and $L_-(B) = L_+(A)$.

The automaton B is called the *dual automaton* to the automaton A. It is denoted by DA and by Theorem 2.2, it exists for any deterministic finite automaton.

In addition, because the positive (upper) language of a deterministic finite automaton is regular, we have the same property of negative (lower) languages.

Theorem 2.3. For any deterministic finite automaton A, its lower language $L_-(A)$ is regular.

Indeed, the language $L_+(B)$ of the automaton B that is dual to A is regular and $L_-(A) = L_+(B)$.

Theorems 2.2 and 2.3 imply the following result.

Corollary 2.3. For finite automata, the class of all positive (upper) languages coincides with the class of all negative (lower) languages and with the class of all regular languages.

Remark 2.1. There are different ways of defining languages. Automaton languages are determined by (abstract) automata. Algorithmic languages are determined by algorithms as system of computational rules. Denotational languages are determined by formulas.

Examples of automaton languages are languages of finite automata or Turing machines. Examples of algorithmic languages are instructional programming languages or languages of formal grammars. Examples of automaton languages are languages determined by regular expressions, which are called regular languages.

Some classes of these languages coincide. For instance, it is proved that regular languages coincide with class of languages of finite automata and with class of languages of regular grammars.

In what follows, we do not make a distinction between automaton languages and algorithmic languages calling all of them algorithmic languages.

For nondeterministic finite automata, the variety of algorithmic languages is essentially richer. Indeed, when A is a nondeterministic finite automaton, then given a word w as the input to A, four results are possible:

(1) A consumes w always ending in a final (accepting) state, which means that A accepts w

(2) A consumes w always ending not in the final state, that is, in a rejecting state, and this means that A rejects w

(3) A does not consume w, which means that the computation of A is undefined for w

(4) A consumes w with some paths of computation bringing it to a final state while there are other paths of computation that bring A to the state that is not final, which means that A both accepts and rejects w

In the first case, we assume that w is accepted by A. In the second case, we assume that w is rejected by A. In the third case, we assume that it is not defined whether w is accepted by A or not. Finally, in the fourth case, we assume that w is both accepted and rejected by A.

As a result, we have four algorithmic languages of a nondeterministic finite automaton A in a general case:

- The *positive* (*upper*) *language* $L_+(A)$ of the finite automaton A consists of all words w totally accepted (recognized) by A, that is, all paths from the start state produced by the consumption of w end in a final state.

- The *negative* (*lower*) *language* $L_-(A)$ of an automaton A consists of all words w totally rejected (recognized) by A, that is, all paths from the start state produced by the consumption of w end in a state that is not final.
- The *impartial language* $L_0(A)$ of an automaton A consists of all words computationally undefined by A, that is, of those words that are not consumed by A.
- The *double language* $L_\pm(A)$ of an automaton A consists of all words that are consumed by A and have both the transition paths reaching accepting states and transition paths reaching rejecting states of A

Example 2.1. Let us consider a finite automaton A with the alphabet $\Sigma = \{0, 1\}$, the set of states $Q = \{ q_0 , q_1 , q_2 , q_3\}$, the set of final states $F = \{ q_1\}$ and transition rules:

$$q_0 , 1 \to q_1$$

$$q_1 , 1 \to q_1$$

$$q_1 , 0 \to q_1$$

$$q_0 , 0 \to q_2$$

$$q_2 , 0 \to q_3$$

$$q_3 , 1 \to q_3$$

$$q_3 , 0 \to q_3$$

Then we have:

- The *positive* (*upper*) *language* $L_+(A)$ of the automaton A consists of all words that start with 1
- The *negative* (*lower*) *language* $L_-(A)$ of an automaton A consists of one word 0
- The *impartial language* $L_0(A)$ of an automaton A consists of all words that start with 01
- The *double language* $L_\pm(A)$ of an automaton A consists of all words that start with 00

Lemma 2.1. For any nondeterministic finite automaton A, all four languages are disjoint.

Properties of nondeterministic finite automata allow proving the nondeterministic Duality Theorem for finite automata.

Theorem 2.4. For any nondeterministic finite automaton A, there is a nondeterministic finite automaton B such that $L_+(B) = L_-(A)$, $L_-(B) = L_+(A)$, $L_0(A) = L_0(B)$ and $L_\pm(A) = L_\pm(B)$.

The automaton B is also called dual to the automaton A and it is denoted by DA.

This gives us the following result.

Corollary 2.2. For any nondeterministic finite automaton A, there is the dual automaton DA.

As for deterministic finite automata, we show regularity of the languages of nondeterministic finite automata.

Theorem 2.5. For any nondeterministic finite automaton A, all its four algorithmic languages are regular.

Proof. Let us consider a nondeterministic finite automaton A. A nondeterministic finite automaton A accepts a word w if there is a path from the start state to a final state produced by the consumption of w. For languages, it means that the standard language $L(A)$ of a nondeterministic finite automaton A is $L_{+}(A) \cup L_{\pm}(A)$ and it is a regular language (Cohen, 1991; Sipser, 1997).

For the dual automaton DA, we have

$$L(\mathrm{D}A) = L_{+}(\mathrm{D}A) \cup L_{\pm}(\mathrm{D}A) = L_{-}(A) \cup L_{\pm}(A)$$

and $L(\mathrm{D}A)$ is also a regular language.

Thus,

$$L(A) \setminus L(\mathrm{D}A) = L_{+}(A) \setminus L_{-}(A) = L_{+}(A)$$

because by Lemma 2.1,

$$L_{+}(A) \cap L_{-}(A) = \varnothing$$

Regular languages are closed with respect to difference and both languages $L(A)$ and $L(\mathrm{D}A)$ are regular. Consequently, $L_{+}(A)$ is a regular language.

Considering the difference

$$L(\mathrm{D}A) \setminus L(A) = L_{+}(\mathrm{D}A) = L_{-}(A),$$

we come to the conclusion that $L_{-}(A)$ is also a regular language.

In addition, we have

$$L_{\pm}(A) = L(A) \setminus L_{+}(A)$$

It means that $L_{\pm}(A)$ is also a regular language.

Regular languages are closed with respect to union. Thus, $L_{+}(\mathrm{D}A) \cup L_{-}(A) \cup L_{\pm}(A)$ is a regular language. The $L_0(A) = \Sigma^* \setminus (L_{+}(\mathrm{D}A) \cup L_{-}(A) \cup L_{\pm}(A))$ and Σ^* is a regular language. Consequently, $L_0(A)$ is a regular language.

Theorem is proved.

We naturally defined two types of languages of deterministic finite automata proving their regularity. However, a deterministic finite automaton can have (determine) more than two algorithmic languages. Indeed, in the conventional definition of finite automata, only accepting states are defined while all other states are treated as rejecting states. At the same time, it is possible to separate the set of all states Q of a finite automaton A into three groups: P consists of positive states, N consists of negative states, and D consists of undefined states. This partition determines three *partition languages* for the deterministic finite automaton A.

- The *positive* (*upper*) *language* $L_{+}(A) = L_P(A)$ of the finite automaton A consists of all words w totally accepted (recognized) by A, that is, the path from the start state produced by the consumption of w ends in a state from P.
- The *negative* (*lower*) *language* $L_{-}(A) = L_N(A)$ of an automaton A consists of all words w totally rejected (recognized) by A, that is, the path from the start state produced by the consumption of w ends in a state from N.
- The *indeterminate language* $L_{\div}(A) = L_D(A)$ of an automaton A consists of all words w the path from the start state produced by the consumption of w ends in a state from D.

As deterministic finite automata consume all words in their alphabet, they do not have impartial languages.

Lemma 2.2. For any nondeterministic finite automaton A, all its three partition languages are disjoint.

Theorem 2.6. For any deterministic finite automaton A, all its three partition languages are regular.

Indeed, defining X a the set of accepting states of the automaton B where $X \in \{P, N, D\}$ and B has the same transition rules as A, we have that B is a deterministic automaton with the conventional language $L(B) = L_X(A)$. As $X \in \{P, N, D\}$ and $L(B)$ is a regular language, all three partition languages of the automaton A are regular.

3. Languages of Turing machines

As we proved for finite automata, all their languages are regular. For other classes of automata, positive and negative languages can be essentially different.

Example 3.1. Let us take the class $\boldsymbol{T}$ of all Turing machines that work with words in the alphabet Σ and define that a word w is accepted by a Turing machine T from $\boldsymbol{T}$ if and only if given w as an input to T, the machine T stops after making a finite number of steps. If given w as an input to T, the machine T does not stop, then w is computationally undefined by T. In this case, for any Turing machine T, $L_{-}(A) = L_{\pm}(A) = \varnothing$ and $L_{+}(A) \cup L_{0}(A) = \Sigma^*$. As for this class, all negative languages are empty while there are many non-empty positive languages, the class of all positive languages of the automata from $\boldsymbol{T}$ is the class of all recursively enumerable languages and it does not coincide with the class of all negative languages of the machines from $\boldsymbol{T}$.

Example 3.2. If T is a Turing machine from $\boldsymbol{T}$, then it is possible to define that the word w is rejected by a Turing machine T from $\boldsymbol{T}$ if the machine T does not stop given w as the input. In this case, the class of all positive languages of the automata from $\boldsymbol{T}$ is also the class of all recursively enumerable languages and it does not coincide with the class of all negative languages because the latter contains languages that are not recursively enumerable.

We see that relations between languages of a given Turing machine depend on how we define these languages. Thus, let us define these languages in a natural way and study algorithmic languages of accepting Turing machines that work with words in a finite alphabet Σ.

- The *upper* or *positive language* $L_{+}(T)$ of a deterministic Turing machine T consists of all words w accepted (recognized) by T, i.e., when with the input w, T stops in a final (accepting) state after making a finite number of steps
- The *lower* or *negative language* $L_{-}(T)$ of a deterministic Turing machine T consists of all words w rejected (negatively recognized) by T, i.e., when with the input w, T stops not in a final, that is, in a rejecting state after making a finite number of steps
- The *impartial language* $L_{0}(T)$ of a deterministic Turing machine T consists of all words w such that T does not stop given w as its input

This gives us three languages for deterministic Turing machines. At the same time, it is also possible to split all states of a deterministic Turing machine into three groups – positive states, negative states and indefinite states – obtaining more languages.

Let us study properties of algorithmic languages of deterministic Turing machines.

Lemma 3.1. For any deterministic Turing machine, all three languages – positive, negative and impartial - are disjoint.

Remark 3.1. Assuming that the Turing machine T accepts words some of which are logical statements, it is possible to suggest the following interpretation of the considered languages. The positive language $L_{+}(T)$ consists of true statements. The negative language $L_{-}(T)$ consists of false statements. The impartial language $L_{0}(T)$ consists of statements the truth values of which are undefined.

Similar to what is done above for finite automata, it is possible to define dual Turing machines.

Definition 3.1. A deterministic Turing machine Q is *dual* to a deterministic Turing machine T if we have $L_{+}(Q) = L_{-}(T)$ and $L_{-}(Q) = L_{+}(T)$.

Note that the dual deterministic Turing machine is not defined in a unique way because different machines can determine the same group of languages.

Properties of Turing machines imply the following result.

Theorem 3.1. For any deterministic Turing machine, there is the dual deterministic Turing machine.

Definition 3.2. a) A formal language L in Σ is *positively* (*negatively*) *recursively decidable* if there is a Turing machine T such that $L = L_{+}(A)$ ($L = L_{-}(A)$) and $L_{+}(A) \cup L_{-}(A) = \Sigma^{*}$.

b) A formal language L in Σ is *positively* (*negatively*) *recursively recognizable* if there is an automaton (algorithm) A such that $L = L_{+}(A)$ ($L = L_{-}(A)$).

Theorem 3.1 implies the following result.

Corollary 3.1. A language L is positively recursively recognizable (decidable) if and only if it is negatively recursively recognizable (decidable).

It is known that for any Turing machine, there is another Turing machines that determines the same conventional language and accepts a word only if it stops (Hopcroft, et al, 2001; Burgin, 2005). This gives us the following result.

Proposition 3.1. For any deterministic Turing machine T, there is a deterministic Turing machine Q such that $L_{+}(Q) = L_{+}(T)$, $L_{0}(Q) = L_{0}(T) \cup L_{-}(T)$ and $L_{-}(Q) = \varnothing$.

Corollary 3.2. For any deterministic Turing machine T, there is a deterministic Turing machine Q such that $L_{-}(Q) = L_{-}(T)$, $L_{0}(Q) = L_{0}(T) \cup L_{+}(T)$ and $L_{+}(Q) = \varnothing$.

As conventional languages of Turing machines are recursively enumerable and for any deterministic Turing machine T its conventional language coincides with $L_{+}(T)$, Theorem 3.1 implies the following result.

Theorem 3.2. a) For any deterministic Turing machine, its languages $L_{-}(T)$ and $L_{+}(T)$ are recursively enumerable.

b) Any recursively enumerable language L is equal to the positive language $L_{+}(T)$ of some deterministic Turing machine T and to the negative language $L_{-}(Q)$ of some deterministic Turing machine Q.

Example 3.2 shows that impartial languages of Turing machines are not always recursively enumerable.

Corollary 3.3. A formal language L in Σ is *positively* (*negatively*) *recursively recognizable* if and only if it is recursively enumerable.

The dual to recursive enumerability concept is recursive coenumerability.

Definition 3.3. A formal language L is *recursively coenumerable* if it is the complement CL of a recursively enumerable language L.

Theorem 3.3. A language is recursively coenumerable if and only if it is the impartial language $L_0(T)$ for some deterministic Turing machine T.

Proof. *Necessity*. Let us consider a deterministic Turing machine T. Then its impartial language $L_0(T)$ is the complement of the union $L_{-}(T) \cup L_{+}(T)$ and this union is a recursively enumerable language because both $L_{+}(T)$ and $L_{-}(T)$ are recursively enumerable languages and the union of two recursively enumerable languages is a recursively enumerable language (Hopcroft, 2001).

Sufficiency. Let us take a recursively coenumerable language L. It is a complement of a recursively enumerable language M. By definition, M is the conventional language of a Turing machine T such that when given the input w, the machine T stops after making a finite number of steps when w belongs to M and does not stop otherwise (Burgin, 2005). It means that $L_{+}(T) = M$ and $L_0(T) = L$.

Theorem is proved.

Corollary 3.4. The complement of an impartial language L is a recursively enumerable language.

Corollary 3.5. The complement CL of a recursively enumerable language L is an impartial language.

Post theorem (cf. (Shen and Vereshchagin, 2003)) implies the following result.

Lemma 3.2. A language L is recursive decidable if and only if it is both recursively enumerable and recursively coenumerable.

Recursive enumerability is naturally connected to recursive decidability.

Theorem 3.4. For any deterministic Turing machine T, the languages $L_+(T)$ and $L_-(T)$ are recursively decidable if and only if all three languages $L_+(T)$, $L_-(T)$ and $L_0(T)$ are recursively enumerable.

Proof. *Sufficiency*. Let us take a deterministic Turing machine T such that all three languages $L_+(T)$, $L_-(T)$ and $L_0(T)$ are recursively enumerable. The union $L_-(T) \cup L_0(T)$ is a recursively enumerable language because both $L_-(T)$ and $L_0(T)$ are recursively enumerable languages and the union of two recursively enumerable languages is a recursively enumerable language (Hopcroft, 2001). Thus, $L_+(T)$ is a recursively coenumerable language because its complement $L_-(T) \cup L_0(T)$ is a recursively enumerable language. By Lemma 3.2, the language $L_+(T)$ is recursively decidable.

The proof that the language $L_-(T)$ is recursively decidable is similar.

Necessity. Let us take a deterministic Turing machine T such that both languages $L_+(T)$ and $L_-(T)$ are recursively decidable. The union $L_-(T) \cup L_+(T)$ is a recursively decidable language because both $L_-(T)$ and $L_+(T)$ are recursively decidable languages and the union of two recursively decidable languages is a recursively decidable language (Hopcroft, 2001). Then by Lemma 3.2, the language $L_-(T) \cup L_+(T)$ is recursively coenumerable. It means that the language $L_0(T)$ is recursively enumerable.

Theorem is proved.

Corollary 3.5. For any deterministic Turing machine T, the languages $L_+(T)$, $L_-(T)$ and $L_0(T)$ are recursively decidable if and only if they are recursively enumerable.

Remark 3.2. The condition that both languages $L_+(T)$ and $L_-(T)$ are recursively decidable is essential in Theorem 3.4 as the following result demonstrates.

Example 3.3. Let us consider a universal deterministic Turing machine U assuming that all its states are positive. Then the language $L_+(U)$ is recursively enumerable, the language $L_-(T)$ is empty and thus, decidable while the language $L_0(T)$ is not recursively enumerable.

Corollary 3.6. For any deterministic Turing machine T, if at least one of the languages $L_+(T)$, $L_-(T)$ and $L_0(T)$ is not recursively decidable, then $L_0(T)$ is not recursively enumerable.

Corollary 3.7. For any deterministic Turing machine T, if at least one of the languages $L_+(T)$, $L_-(T)$ and $L_0(T)$ is not recursively enumerable, then $L_0(T)$ is not recursively enumerable.

It is known that the sum and intersection of recursively enumerable languages are recursively enumerable languages (Cohen, 1991; Sipser, 1997). Let us study this property for recursively coenumerable languages.

Theorem 3.4. The sum of two recursively coenumerable languages is a recursively coenumerable language.

Proof. Let us consider two recursively coenumerable languages L and M. By De Morgan's laws (Halmos, 1974), we have

$$L \cup M = \mathrm{C}(\mathrm{C}L \cap \mathrm{C}M)$$

As the languages L and M are recursively coenumerable, their complements CL and CM are recursively enumerable. The intersection of two recursively enumerable languages is recursively enumerable, that is, C$L \cup$ CM is a recursively enumerable language and its complement $L \cup M$ is a recursively coenumerable language.

Theorem is proved.

Theorems 3.3 and 3.4 imply the following result.

Corollary 3.8.The sum of two impartial languages $L_0(T)$ and $L_0(Q)$ of Turing machines T and Q, correspondingly, is the impartial language $L_0(R)$ of some Turing machine R.

The intersection of languages has similar properties.

Theorem 3.5. The intersection of two recursively coenumerable languages is a recursively coenumerable language.

Proof. Let us consider two recursively coenumerable languages L and M. By De Morgan's laws (Halmos, 1974), we have

$$L \cap M = \mathrm{C}(\mathrm{C}L \cup \mathrm{C}M)$$

As the languages L and M are recursively coenumerable, their complements CL and CM are recursively enumerable. The union of two recursively enumerable languages is recursively enumerable, that is, C$L \cup$ CM is a recursively enumerable language and its complement $L \cup M$ is a recursively coenumerable language.

Theorem is proved.

Theorems 3.3 and 3.5 imply the following result.

Corollary 3.9.The intersection of two impartial languages $L_0(T)$ and $L_0(Q)$ of Turing machines T and Q, correspondingly, is the impartial language $L_0(R)$ of some Turing machine R.

The difference of languages can preserve linguistic properties only partially.

Theorem 3.6. The difference of a recursively enumerable language and recursively coenumerable language is a recursively enumerable language.

Proof. Let us consider a recursively enumerable language L and recursively coenumerable language M. Taking their difference, we have

$$L \setminus M = L \cap \mathrm{C}M$$

By definition, CM is a recursively enumerable language and the intersection of two recursively enumerable languages is a recursively enumerable language. Thus, the difference $L \setminus M$ is a recursively enumerable language.

Theorem is proved.

Theorems 3.2, 3.3 and 3.6 imply the following result.

Corollary 3.9.The difference of the positive language $L_+(T)$ (negative language $L_-(T)$) of a Turing machines T and the impartial language $L_0(Q)$ of a Turing machines Q is the positive language $L_+(R)$ (negative language $L_-(T)$) of some Turing machine R.

When the classes of languages are changed, we obtain a similar result.

Theorem 3.7. The difference of a recursively coenumerable language and a recursively enumerable language is a recursively coenumerable language.

Proof. Let us consider a recursively coenumerable language L and recursively enumerable language M. Taking their difference, we have

$$L \setminus M = L \cap \mathrm{C}M$$

By definition, CM is a recursively coenumerable language and by Theorem 3.5, the intersection of two recursively coenumerable languages is a recursively coenumerable language. Thus, the difference $L \setminus M$ is a recursively coenumerable language.

Theorem is proved.

Theorems 3.2, 3.3 and 3.7 imply the following result.

Corollary 3.10.The difference of the impartial language $L_0(T)$ of a Turing machines T and the positive language $L_+(Q)$ (negative language $L_-(Q)$) of a Turing machines Q is the impartial language $L_0(R)$ of some Turing machine R.

Remark 3.3. The difference of recursively enumerable languages can be recursively coenumerable while the difference of recursively coenumerable languages can be recursively enumerable.

For nondeterministic Turing machines, their algorithmic languages are defined in a different way. Let us consider a nondeterministic Turing machine T, in which all states are divided into two groups: final or accepting states and rejecting states.

- The *upper* or *positive language* $L_{+}(T)$ of a nondeterministic Turing machine T consists of all words w accepted (recognized) by T, i.e., when given the input w, all threads of the computation of T stop in a final (accepting) state after making a finite number of steps
- The *lower* or *negative language* $L_{-}(T)$ of a nondeterministic Turing machine T consists of all words w rejected (negatively recognized) by T, i.e., when given the input w, all threads of the computation of T stop not in a final, that is, in a rejecting state after making a finite number of steps
- The *impartial language* $L_{0}(T)$ of a nondeterministic Turing machine T consists of all words w such that T does not stop given w as its input
- The *double language* $L_{\pm}(T)$ of T consists of all words w such that given the input w, each thread of the computation of T stops either in a final (accepting) state or in a rejecting state after making a finite number of steps
- The *positively mixed language* $L_{+0}(T)$ of a nondeterministic Turing machine T consists of all words w such that given the input w, each thread of the computation of T either stops in a final (accepting) state after making a finite number of steps or does not stop at all
- The *negatively mixed language* $L_{-0}(T)$ of a nondeterministic Turing machine T consists of all words w such that given the input w, each thread of the computation of T either stops in a rejecting state after making a finite number of steps or does not stop at all
- The *lenient double language* $L_{\pm 0}(T)$ of T consists of all words w such that given the input w, each thread of the computation of T either stops in a final (accepting) state or stops in a rejecting state after making a finite number of steps or does not stop at all

We see that a nondeterministic Turing machine determines seven algorithmic languages. Let us study properties of these languages.

As conditions determining seven algorithmic languages are incompatible, we have the following result.

Lemma 3.4. For any nondeterministic Turing machine T, all seven algorithmic languages are disjoint.

Remark 3.4. Assuming that the Turing machine T accepts words some of which are logical statements, it is possible to suggest the following interpretation of the considered languages. The positive language $L_{+}(T)$ consists of the statements the truth of which is proved. The negative language $L_{-}(T)$ consists of the statements the falsehood of which is proved. The impartial language $L_{0}(T)$ consists of statements the truth values of which are undefined. The double language $L_{\pm}(T)$ consists of the statements for which it is proved that they are true and false at the same time. The positively mixed language $L_{+0}(T)$ consists of the true statements the truth of which is not proved. The negatively mixed language $L_{+0}(T)$ consists of the false statements the falsehood of which is not proved. The lenient double language $L_{\pm 0}(T)$ consists of the statements that are true and false at the same time but this is not proved.

Remark 3.5. It is also possible to split all states of a nondeterministic Turing machine T into three groups – positive states, negative states and indefinite states – obtaining much more languages determined by T.

Definition 3.4. A nondeterministic Turing machine T^{o} is *dual* to a nondeterministic Turing machine T if we have $L_{+}(T^{o}) = L_{-}(T)$, $L_{+0}(T^{o}) = L_{-o}(T)$, $L_{-0}(T^{o}) = L_{+0}(T)$, and $L_{-}(T^{o}) = L_{+}(T)$.

Properties of nondeterministic Turing machines imply the following result.

Theorem 3.8. For any nondeterministic Turing machine T, there is the dual nondeterministic Turing machine T^{o}.

Note that the dual nondeterministic Turing machine is not defined in a unique way because different machines can determine the same group of languages.

Theorem 3.9. For any nondeterministic Turing machine T, languages $L_{+}(T)$, $L_{-}(T)$, $L_{\pm}(T)$, and $L_{\pm}(T) \cup L_{\pm 0}(T)$ are recursively enumerable and languages $L_{0}(T)$ and $L_{\pm 0}(T) \cup L_{+0}(T) \cup L_{-0}(T) \cup L_{0}(T)$ are recursively coenumerable.

Proof. Let us consider a nondeterministic Turing machine T with the set D of states and assume that q is an arbitrary element from D. We also assume that all states D of the machine T are divided into two parts: the set B of accepting (positive) states and the set C of rejecting (negative) states.

For proving this theorem, we use the technique “making an infinite cycle at a state q,“ which is described below.

Taking the Turing machine T, we build the Turing machine T_q by adding the new state r to D and the system of new transition rules $aq \to ar$ and $ar \to aq$ for all symbols a from the alphabet Σ. As a result, when T_q reaches the sate q, it goes into an infinite cycle never stopping.

Now we build the Turing machine T_C that has all states and transition rules of T plus one more state r that does not belong to D by making an infinite cycle at all states from C. As the machine T_C never stops in the states from C, we have the following equalities

$$L_{+}(T_C) = L_{+}(T)$$

$$L_{-}(T_C) = L_{\pm}(T_C) = L_{\pm 0}(T_C) = L_{-0}(T_C) = \varnothing$$

$$L_{0}(T_C) = L_{0}(T) \cup L_{-0}(T)$$

$$L_{+0}(T_C) = L_{+0}(T) \cup L_{\pm 0}(T)$$

To continue, we consider two cases: one when the language $L_{+0}(T_C)$ is empty and another when the language $L_{+0}(T_C)$ is not empty.

In the first case, we have

$$L_{+}(T_C) \cup L_{0}(T_C) = \Sigma^*$$

Besides, in this case, $L_{+}(T_C)$ is the conventional language of the Turing machine T_C and thus, it is recursively enumerable. Consequently, $L_{0}(T_C)$ is recursively coenumerable.

Now we consider the second case when the language $L_{+0}(T_C)$ is not empty. By the standard technique in the theory of Turing machines described, for example, in Theorem 8.11 from (Hopcroft, et al, 2001), it is possible to build a deterministic Turing machine DT_C that simulates all steps of the Turing machine T_C and stops when T_C stops. We modify this machine so that it does not stop when there is at least one possible move of the machine T_C . It means that given a word w as the input to DT_C , this machine never stops if there is at least one infinite thread in the computation of the machine T_C with the same input. This also means that the machine DT_C stops if and only if the lengths of all threads are bounded.

Let us prove that the lengths of all threads for computations of T_C with any input w from $L_{+}(T_C)$ are bounded. Let us assume that this is not true, that is, for some w from $L_{+}(T_C)$, there are threads of arbitrary big length ending in the state q from B. In this case, by Kőnig's infinity lemma (Kőnig, 1927), there is an infinite thread starting with w because the graph of computational threads is locally finite as any Turing machine has the infinite number of transition rules. However, by definition, the word w must belong not to $L_{+}(T_C)$ but to $L_{+}(T_C)$. Thus, our assumption was not true and all threads for computations of T_C with any input w from $L_{+}(T_C)$ are bounded.

As a result, the deterministic Turing machine DT_C stops given the input from $L_+(T_C)$ and does not stop on all other inputs. It means that $L_+(T_C)$ is the conventional language of the machine DT_C and thus, it is recursively enumerable.

As we know, $L_+(T_C) = L_+(T)$. Consequently, the language $L_-(Q)$ is also recursively enumerable.

Taking the dual nondeterministic Turing machine T^o instead of T and using the same technique, we prove that the language $L_-(Q)$ is also recursively enumerable.

Let H be a nondeterministic Turing machine obtained from T by taking positive states as final (accepting) states. The conventional language of the machine H is $L_+(T) \cup L_\pm(T) \cup L_{\pm 0}(T) \cup L_{+0}(T)$. Consequently, this language is recursively enumerable.

In a similar way, we build the nondeterministic Turing machine Z obtained from T by taking negative states as its final (accepting) states. The conventional language of the machine Z is $L_-(T) \cup L_\pm(T) \cup L_{\pm 0}(T) \cup L_{-0}(T)$. Consequently, this language is recursively enumerable.

The intersection of two recursively enumerable languages is a recursively enumerable language (Hopcroft, et al, 2001). Thus, $L_\pm(T) \cup L_{\pm 0}(T) = [L_+(T) \cup L_\pm(T) \cup L_{\pm 0}(T) \cup L_{+0}(T)] \cap [L_-(T) \cup L_\pm(T) \cup L_{\pm 0}(T) \cup L_{-0}(T)]$ is a recursively enumerable language.

As it was explained, taking the machine H, we can build a deterministic Turing machine DH simulating each step of H and working without stopping when there is at least one rule for making the next step. Analyzing the work of DH, we see that its conventional language is equal to $L_+(T) \cup L_\pm(T)$. Consequently, this language is recursively enumerable.

Taking the machine Z, we can build a deterministic Turing machine R simulating each step of H and working without stopping when there is at least one rule for making the next step. This will show that $L_-(T) \cup L_\pm(T)$ is a recursively enumerable language.

The intersection of two recursively enumerable languages is a recursively enumerable language (Hopcroft, et al, 2001). Thus, $L_\pm(T) = [L_+(T) \cup L_\pm(T)] \cap [L_-(T) \cup L_\pm(T)]$ is a recursively enumerable language.

Let R be a nondeterministic Turing machine obtained from T by making all states final (accepting) states. The conventional language of the machine R is $L(R) = L_+(T) \cup L_\pm(T) \cup L_{\pm 0}(T) \cup L_{+0}(T) \cup L_-(T) \cup L_{-0}(T)$. Consequently, $L(R) \cup L_0(T) = \Sigma^*$ and the language $L_0(T)$ is recursively coenumerable.

By definition, $L_{+}(T) \cup L_{-}(T) \cup L_{\pm}(T) \cup L_{\pm 0}(T) \cup L_{+0}(T) \cup L_{-0}(T) \cup L_{0}(T) = \Sigma^*$. Languages $L_{+}(T)$, $L_{-}(T)$ and $L_{\pm}(T)$ are recursively enumerable. As the union of three recursively enumerable languages is a recursively enumerable language, the language $L_{\pm 0}(T) \cup L_{+0}(T) \cup L_{-0}(T) \cup L_{0}(T)$ is recursively coenumerable because the union of three recursively enumerable languages is a recursively enumerable language.

Theorem is proved.

In contrast to the languages $L_{+}(T)$, $L_{\pm}(T)$, $L_{-}(T)$ and $L_{0}(T)$, some of the languages $L_{\pm 0}(T)$, $L_{+0}(T)$ and $L_{-0}(T)$ can be recursively enumerable and some can be recursively coenumerable. Even more, we have the following result.

Theorem 3.10. a) Any recursively enumerable language is equal to the positively mixed language $L_{+0}(T)$ of some nondeterministic Turing machine T.

b) Any recursively enumerable language is equal to the negatively mixed language $L_{-0}(T)$ of some nondeterministic Turing machine T.

c) Any recursively enumerable language is equal to the lenient double language $L_{\pm 0}(T)$ of some nondeterministic Turing machine T.

d) Any recursively coenumerable language is equal to the positively mixed language $L_{+0}(T)$ of some nondeterministic Turing machine T.

e) Any recursively coenumerable language is equal to the negatively mixed language $L_{-0}(T)$ of some nondeterministic Turing machine T.

f) Any recursively coenumerable language is equal to the lenient double language $L_{\pm 0}(T)$ of some nondeterministic Turing machine T.

Proof. a) Let us consider a recursively enumerable language L. By Theorem 3.2, L is equal to the positive language $L_{+}(T)$ of some deterministic Turing machine T. We transform the machine T into the nondeterministic Turing machine P using the following operations. At first, two new states r and p are added to the states D of the machine T. Then for each symbol a from the alphabet Σ and for each state q from D, new transition rules $aq \to ap$, $ap \to ar$ and $ar \to ap$ are added to the transition rules of the machine T.

As a result, the computation of the machine P contains an infinite thread for any input w because from any state and any observed symbol on the tape, the machine P can go into an infinite cycle. Consequently, we have $L_{+}(P) = \varnothing$ and $L_{+0}(P) = L_{+}(T)$. It means that $L = L_{+0}(P)$ and thus, part a) of the theorem is proved.

b) The proof of part b) is similar to the previous proof only instead of positive languages, we consider negative languages of Turing machines.

c) Let us consider a recursively enumerable language L. By Theorem 3.2, L is equal to the positive language $L_{+}(T)$ of some deterministic Turing machine T. We transform the machine T into the nondeterministic Turing machine Q using the following operations. At first, a new state r is added to the states D of the machine T. Then for each symbol a from the alphabet Σ and for each state q from D, new transition rules $aq \to ap$ are added to the transition rules of the machine T. In addition, we extend the set of negative states of T by adding the new state r to this set.

As a result, the positive language $L_{+}(T)$ is transformed into the double language $L_{\pm}(Q)$. As the double language $L_{\pm}(T)$ is empty, it means that $L = L_{\pm}(Q)$ and part c) is proved.

d) Let us consider a recursively coenumerable language L. By Theorem 3.3, L is equal to the impartial language $L_0(T)$ of some deterministic Turing machine T. We transform the machine T into the nondeterministic Turing machine R using the following operations. At first, the new states r is added to the states D of the machine T. Then for each symbol a from the alphabet Σ and for each state q from D, new transition rules $aq \to ar$ are added to the transition rules of the machine T. In addition, we extend the set of positive states of T by adding the new state r to this set.

As a result, the computation of the machine R contains a finite thread for any input w, in which the machine R stops in a positive state r. Consequently, we have $L_0(R) = \varnothing$ and $L_{+0}(R) = L_0(T)$. It means that $L = L_{+0}(R)$ and thus, part f) of the theorem is proved.

e) The proof of part e) is similar to the proof of part e) only instead of positive languages, we consider negative languages of Turing machines.

f) Let us consider a recursively coenumerable language L. By Theorem 3.3, L is equal to the impartial language $L_0(T)$ of some deterministic Turing machine T. We transform the machine T into the nondeterministic Turing machine P using the following operations. At first, the new states r and p are added to the states D of the machine T. Then for each symbol a from the alphabet Σ and for each state q from D, new transition rules $aq \to ar$ and $aq \to ap$ are added to the transition rules of the machine T. In addition, we extend the set of positive states of T by adding the new state r to this set and the set of negative states of T by adding the new state p to this set.

As a result, the computation of the machine P contains a finite thread for any input w, in which the machine P stops in a positive state r, and a finite thread for any input w, in which the machine

P stops in a negative state p. Consequently, we have $L_0(P) = \varnothing$ and $L_{\pm 0}(P) = L_0(T)$. It means that $L = L_{\pm 0}(P)$ and thus, part f) of the theorem is proved.

Theorem is proved.

We remind that a formal language L is recursively decidable if it belongs to the class of all recursively enumerable languages and recursively coenumerable languages.

Corollary 3.9. Each of the set of languages $\{L_{+0}(T)$; T is a nondeterministic Turing machine$\}$, $\{L_{-0}(T)$; T is a nondeterministic Turing machine$\}$, and $\{L_{\pm 0}(T)$; T is a nondeterministic Turing machine$\}$ contains all recursively decidable languages.

The results of Theorem 3.10 bring us to a new class of formal languages.

Definition 3.6. A formal language L is *recursively bienumerable* if it belongs either to the class of all recursively enumerable languages or to the class of all recursively coenumerable languages.

Corollary 3.10. Each of the set of languages $\{L_{+0}(T)$; T is a nondeterministic Turing machine$\}$, $\{L_{-0}(T)$; T is a nondeterministic Turing machine$\}$, and $\{L_{\pm 0}(T)$; T is a nondeterministic Turing machine$\}$ contains all recursively bienumerable languages.

An interesting problem is to find which of the set of languages $\{L_{+0}(T)$; T is a nondeterministic Turing machine$\}$, $\{L_{-0}(T)$; T is a nondeterministic Turing machine$\}$, or $\{L_{\pm 0}(T)$; T is a nondeterministic Turing machine$\}$ coincides with all recursively bienumerable languages.

4. Conclusion

While in the conventional theory of automata and algorithms, only one language is associated with each automaton or algorithm, here it is demonstrated that each automaton or algorithm determines several algorithmic languages. Properties of these languages were studied and in addition to recursively enumerable and recursively coenumerable languages, we discovered a new class of languages. The new languages are called recursively bienumerable comprising both recursively enumerable and recursively coenumerable languages.

It is necessary to mention that recursively bienumerable languages are conventional languages of certain classes of grammars with prohibition (Burgin, 2005a; 2013), learning correction grammars (Carlucci, et al, 2009; Case and Jain, 2011; Case and Royer, 2016), and selective machines (Burgin, 2021).

The obtained results stimulate new directions of research in the area of formal languages, automata and computation. Thus, it would be interesting to study algorithmic languages

determined by inductive Turing machines (Burgin, 2005), by probabilistic Turing machines (Salomaa, 1969; Sipser, 2006), by Turing machines with oracles, and by selective machines (Burgin, 2021). Another related direction for future resarch is investigation of the impact of operations with automata on operations with their languages.

References

1. Burgin, M. *Super-recursive Algorithms*, Springer, New York/Heidelberg/Berlin, 2005
2. Burgin, M. Grammars with Prohibition and Human-Computer Interaction, in "Proceedings of the Business and Industry Simulation Symposium," Society for Modeling and Simulation International, San Diego, California, 2005a, pp. 143-147
3. Burgin, M. *Basic Classes of Grammars with Prohibition*, Preprint in Computer Science, cs.FL/CL. 1302.5181, 2013, 15 p. (electronic edition: http://arXiv.org)
4. Burgin, M. *Grammars with Exclusion*, Journal of Computer Technology & Applications (JoCTA), v. 6, No. 2, 2015, pp. 56 – 66
5. Burgin, M. *Modelling distributive computation by selective machines*, International Journal of Parallel, Emergent and Distributed Systems, 2021, v. 36, No.5, pp. 395 - 411
6. Carlucci, L., Case, J., Jain, S. Learning correction grammars. J. Symb. Logic 74, 489-516 (2009)
7. Case, J., Jain, S. Rice and Rice-Shapiro theorems for transfinite correction grammars, Math. Logic Quarterly 57(5), 504-516 (2011)
8. Case, J. and Royer, J. Program Size Complexity of Correction Grammars in the Ershov Hierarchy, in 12th Conference of Computability in Europe (CiE 2016), Proceedings, lecture notes in Computer Science, vol. 7921. Heidelberg: Springer; 2016. p. 240–250.
9. Chi, R. S. Y. (1969) *Buddhist formal logic: a study of Dignāga's Hetucakra and K'uei-chi's Great Commentary on the Nyāyapraveśa*, The Royal Asiatic Society of Great Britain, London
10. Church, A. (1956) *Introduction to Mathematical Logic*, Princeton University Press, Princeton
11. Cohen, D.I.A. *Introduction to computer theory*, John Wiley & Sons, Hoboken, NJ, 1991
12. Dummett, M. (1973) The Philosophical Basis of Intuitionistic Logic, in *Logic Colloquium 1973*, North-Holland, pp. 5-40
13. Ganeri, J. (2004) *Indian Logic*. in: Gabbay, Dov& Woods, John (eds.),*Greek, Indian and Arabic Logic*, Volume I of the *Handbook of the History of Logic*, Amsterdam: Elsevier, pp. 309–396
14. Ganeri, J. (Ed.) (2001) *Indian Logic. A Reader*. Routledge Curzon, New York
15. Garfield, J.L. *Empty Words: Buddhist Philosophy and Cross-Cultural Interpretation*, New York: Oxford University Press, 2002
16. Halmos, P.R. *Naive Set Theory*, Springer, New York, 1974
17. Hopcroft, J.E., Motwani, R., and Ullman, J.D. *Introduction to Automata Theory, Languages, and Computation*, Addison Wesley, Boston/San Francisco/New York, 2001

18. Kőnig, D. (1927) Über eine Schlussweise aus dem Endlichen ins Unendliche, *Acta Sci. Math.*, v. 3, No. 2-3, pp. 121–130
19. Salomaa, A. *Theory of Automata*, v. 100 in International Series of Monographs on Pure and Applied Mathematics, Pergamon Press,Oxford,1969
20. Shen, A. and Vereshchagin, N.K. *Computable functions*, AMS, Providence, RI, 2003
21. Sipser, M. *Introduction to the Theory of Computation*, PWS Publishing Co., Boston, 2006