Fundamenta Informaticae 76 (2007) 113–128
IOS Press

Intercode Regular Languages[*]

Yo-Sub Han[†‡]

System Technology Division, Korea Institute of Science and Technology

P.O.BOX 131, Cheongryang, Seoul, Korea, emmous@kist.re.kr

Kai Salomaa[§]

School of Computing, Queen's University, Kingston, Ontario K7L 3N6, Canada

ksalomaa@cs.queensu.ca

Derick Wood[¶]

Department of Computer Science, The Hong Kong University of Science and Technology

Clear Water Bay, Kowloon, Hong Kong SAR, dwood@cs.ust.hk

Abstract. Intercodes are a generalization of comma-free codes. Using the structural properties of finite-state automata recognizing an intercode we develop a polynomial-time algorithm for determining whether or not a given regular language L is an intercode. If the answer is yes, our algorithm yields also the smallest index k such that L is a k-intercode.

Furthermore, we examine the prime intercode decomposition of intercode regular languages and design an algorithm for the intercode primality test of an intercode recognized by a finite-state automaton. We also propose an algorithm that computes the prime intercode decomposition of an intercode regular language in polynomial time. Finally, we demonstrate that the prime intercode decomposition need not be unique.

Keywords: regular languages, finite-state automata, intercodes, state-pair graphs, prime decompositions

[*] Part of this research was carried out while Han and Salomaa were in HKUST.

[†] Han was supported by the Research Grants Council of Hong Kong Competitive Earmarked Research Grant HKUST6197/01E and the KIST Tangible Space Initiative Grant 2E19020.

[‡] Address for correspondence: System Technology Division, Korea Institute of Science and Technology, P.O.BOX 131, Cheongryang, Seoul, Korea

[§] Salomaa was supported by the Natural Sciences and Engineering Research Council of Canada Grant OGP0147224.

[¶] Wood was supported by the Research Grants Council of Hong Kong Competitive Earmarked Research Grant HKUST6197/01E.

1. Introduction

Finite-state automata (FAs) are the basic model used to represent regular languages in many applications. FAs are essentially labeled directed graphs and each path from a start state to a final state spells out an accepted string. There are two well-known families of FAs in the literature: the Thompson automata [22] and the position automata [8, 20]. One advantage of using such families of FAs is that these automata preserve the structural properties of corresponding regular expressions. Caron and Ziadi [3] studied the structural properties of the position automata and Giammarresi et al. [7] examined the structural properties of the Thompson automata.

On the other hand, if we manipulate FAs, then these FAs easily lose certain structural properties; for example, if we catenate a position automaton and a Thompson automaton, then the resulting automaton does not preserve either the position automaton properties or the Thompson automaton properties. Nevertheless, one property remains unchanged in FAs: a path from a start state to a final state spells out an accepted string. The use of *state-pair graphs* relies on this fact. Applications of state-pair graphs have been investigated by the authors [12], or earlier by Berstel and Perrin [1], where this notion is called the square of an automaton. Each node of a state-pair graph is a pair of states of a given FA and the directed edges are labeled by alphabet symbols. We recall the formal definition in Section 3.

Codes play a crucial role in many areas such as information processing, data compression, cryptography, information transmission and so on [15]. They are categorized with respect to different conditions (for example, *prefix-free*, *suffix-free*, *infix-free* or *outfix-free* codes) according to the applications. The theory of codes is closely related to formal languages: a code is a *language*. The conditions that classify code types define proper subfamilies of families of formal languages. For regular languages, for example, prefix-freeness defines the family of prefix-free regular languages, which is a proper subfamily of regular languages. Most of the decision problems related to code properties are decidable for regular languages whereas they often become undecidable for context-free languages [15]. Decidability of general code properties is also investigated in the literature [6, 17].

While comma-free languages have not been studied to the extent of prefix-free languages in the literature, the comma-free property was already introduced in 1958 [9]. Furthermore, Shyr and Yu [21] introduced *intercodes*, as a generalization of comma-free codes, see also Yu [23]. Comma-free codes are the intercodes of index one. Jürgensen et al. [16] have studied the decidability of the intercode property. Fernau et al. [6] mentioned in the conclusion of their paper that “it would be nice to know more about the (time or space) complexities of the decidable code properties”. Complexity questions have been raised also by Berstel and Perrin [2].

Note that if an index k is given, then we can fairly easily check whether or not L is an intercode of index k. However, if no index is given, then the problem is not as straightforward. Jürgensen et al. [16] established that it is decidable whether or not a given regular language is an intercode (of any index). There the complexity of the decision algorithm is not discussed explicitly, but it is easy to verify that an algorithm derived from the construction of the decidability proof is not a polynomial-time algorithm in the general case where the input language is specified by a nondeterministic finite-state automaton (NFA).

It is already shown that state-pair graphs are useful to solve decision problems for subfamilies of regular languages [11, 12, 13]. Based on state-pair graphs, here we design an algorithm that determines whether or not a given regular language L is an intercode (of any index). The algorithm works in polynomial time in the general case where L is given as an NFA. Besides having better time complexity, the algorithm is conceptually easier to understand and implement compared with the algorithm derived from Jürgensen et al. [16].

In Section 2, we define some basic notions. In Section 3, we investigate the decision problem of intercodes and design a polynomial-time algorithm that, given an NFA A, determines whether or not the language $L(A)$ is an intercode of any index. The algorithm relies on the structural properties of A via state-pair graphs. In Section 4, we develop an $O(m^6)$ time algorithm to compute a prime decomposition of an intercode regular language L where m is the number of states of the minimal deterministic finite-state automaton (DFA) for L. Note that it remains an open question whether prime decompositions of general regular languages can be found efficiently [10, 19]. We also demonstrate that the prime decomposition of an intercode is not, in general, unique. In comparison, it is known that prefix codes always have a unique decomposition where the components are prime prefix codes [5].

2. Preliminaries

Let Σ denote a finite alphabet and Σ^* denote the set of all strings over Σ. A language over Σ is any subset of Σ^*. The symbol $\emptyset$ denotes the empty language and the character λ denotes the null string. The cardinality of a finite set S is denoted by $|S|$.

An FA A is specified by a tuple $(Q, \Sigma, \delta, s, F)$, where Q is a finite set of states, Σ is an input alphabet, $\delta \subseteq Q \times \Sigma \times Q$ is a set of transitions, $s \in Q$ is the start state and $F \subseteq Q$ is a set of final states. When F has only a single state f, we write this FA as $(Q, \Sigma, \delta, s, f)$ instead of $(Q, \Sigma, \delta, s, \{f\})$ for simplicity. An FA as defined above is, in general, nondeterministic (an NFA). An FA A is deterministic (a DFA) if for all $(q, a) \in \Sigma \times Q$, $|\{(q, a, q') \in \delta \mid q' \in Q\}| \leq 1$.

Then, the size $|A|$ of A is $|Q| + |\delta|$. A transition (p, a, q) in δ, where $p, q \in Q$ and $a \in \Sigma$, is an *out-transition* of p and an *in-transition* of q. Furthermore, in this case we say that p is a *source state* of q and q is a *target state* of p. A string x over Σ is accepted by A if there is a labeled path from s to a state in F such that the labeled path spells out the string x. Thus, the language $L(A)$ of an FA A is the set of all strings that are spelled out by paths from s to a final state in F. We say that A is *non-returning* if the start state of A does not have any in-transitions and A is *non-exiting* if the final state of A does not have any out-transitions. Note that if all final states of A do not have out-transitions, without loss of generality, we can assume that A has only one final state by merging them. In the following, we always assume that A has only *useful* states; that is, each state of A appears on some path from the start state to some final state.

3. State-pair graphs and intercode regular languages

We first recall the definition of state-pair graphs and the definition of intercodes. Given a fixed index k, it is easy to determine whether a given regular language L is an intercode of index k, basically using closure properties of regular languages. On the other hand, if an index is not specified, the decision problem becomes more involved. In this section we design a polynomial-time algorithm for this problem.

Given an FA $A = (Q, \Sigma, \delta, s, F)$, we assign a unique number for each state in A from 1 to m, where m is the number of states in A.

Definition 3.1. (Han et al. [12])
Given an FA $A = (Q, \Sigma, \delta, s, F)$, we define the state-pair graph $G_A = (V_G, E_G)$, where V_G is a set of nodes and E_G is a set of labeled edges, as follows:

$$V_G = \{(i, j) \mid i, j \in Q\} \text{ and}$$

$$E_G = \{((i, j), a, (x, y)) \mid (i, a, x), (j, a, y) \in \delta \text{ and } a \in \Sigma\}.$$

The crucial property of state-pair graphs is that if there is a string w spelled out by two distinct paths in A, for example, one path is from i to x and the other path is from j to y, then, there is a path from (i, j) to (x, y) in G_A that also spells out the same string w. Note that state-pair graphs do not require the given FAs to be deterministic.

Definition 3.2. (Jürgensen et al. [15])
A language L is an intercode of index k (or a k-intercode) if $L^{k+1} \cap \Sigma^+ L^k \Sigma^+ = \emptyset$. Generally, L is an intercode if L is an intercode of index k, for some k.

First we consider the problem to determine whether or not a given regular language L is a k-intercode, for given $k \geq 1$. We assume that L is bifix-free[1]. Otherwise, we know immediately that L is not an intercode. We can check bifix-freeness of regular languages efficiently [12]. Note that an FA A must be non-exiting and non-returning for $L(A)$ to be bifix-free and, thus, there is one start state and only one final state. Furthermore, λ is not in $L(A)$. If we want to construct an FA A^2 for the language $L(A)L(A)$, we can merge the final state of the first copy of A and the start state of the second copy of A. The FA A^2 has $2|Q| - 1$ states and $2|\delta|$ transitions; namely, $|A^2| < 2|A|$. We can repeat this procedure to construct an FA for the catenation of several As. We use A^k to denote the FA for the catenation of k copies of A and A_i to denote the ith component A in A^k, for $1 \leq i \leq k$.

We now design an algorithm based on state-pair graphs that determines whether or not the language of a given FA $A = (Q, \Sigma, \delta, s, f)$ is a k-intercode, for a given k. We, first, catenate $k{+}1$ As as shown in Fig. 1 and, thus, we have $k{+}1$ copies of states in A. We use (i, j) to denote the state i in A_j. For example, $(m, 1)$ in Fig. 1 is the final state of A_1 in A^{k+1}, where $m = |Q|$; in fact, $(m, 1)$ and $(1, 2)$ are the same state.

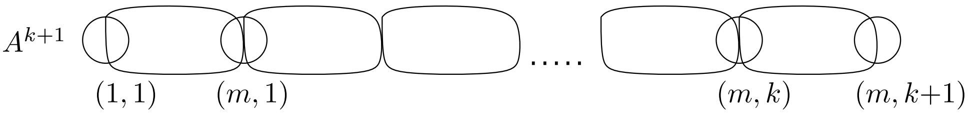

Figure 1. An example of an FA for the catenations of $k{+}1$ As.

Lemma 3.1. Given an FA $A = (Q, \Sigma, \delta, s, f)$, $L(A)$ is a k-intercode if and only if there is no path from $((i, 1), (1, 1))$ to $((j, g), (m, k))$ such that $1 < i < m$ and $(j, g) \neq (m, k{+}1)$ in the state-pair graph $G_{A^{k+1}}$ for A^{k+1}.

[1] A language is *bifix-free* if it is prefix-free and suffix-free.

Proof:
Given strings u and v, we say that u is a *strict infix* of v if u is an infix of v but not a prefix or a suffix of v. By the definition, $L(A)$ is a k-intercode if and only if there is no string $u \in L(A^k)$ such that u is a strict infix of a string $v \in L(A^{k+1})$.
$\Longrightarrow$ Assume that there is a path from $((i,1),(1,1))$ to $((j,g),(m,k))$ in $G_{A^{k+1}}$ that spells out a string w. Thus, there exist two distinct paths, one of which is from $(i,1)$ to (j,g) and the other is from $(1,1)$ to (m,k) and both spell out w in A^{k+1}. Note that $w \in L(A^k)$. Since A^{k+1} has only useful states, there should be a transition sequence from $(1,1)$ to $(i,1)$ that spells out a string x that is not λ since A is non-returning, and a transition sequence from (j,g) to $(m,k{+}1)$ that spells out a string y, which is not λ since $(j,g) \neq (m,k{+}1)$. This implies that A^{k+1} accepts xwy, and x and y are not λ. Then, $L(A^{k+1}) \cap \Sigma^+ L(A^k)\Sigma^+ \neq \emptyset$ — a contradiction.
$\Longleftarrow$ Assume that $L(A)$ is not a k-intercode. Then, there are two strings $u \in L(A^k)$ and $v \in L(A^{k+1})$ such that u is a strict infix of v; namely $v = xuy$, where x and y are not λ. Note that $u = u_1u_2\cdots u_k$ and each u_h, for $1 \leq h \leq k$, is spelled out by A_h in A^{k+1} and, thus, there is a path from $(1,1)$ to (m,k) that spells out u in A^{k+1}. Since A^{k+1} accepts $v = xuy$, we reach some state q after reading the prefix x of v. Note that q cannot be $(1,1)$ since A is non-returning.

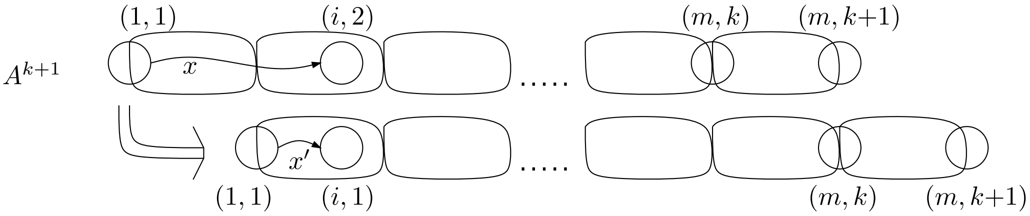

Figure 2. If q is not in A_1, then we can choose another x' such that $v' = x'uy'$ has u as a strict infix. Therefore, we can always guarantee that there exists a state q such that $q = (i,1)$ for $1 < i < m$.

It might be possible that q is not in A_1 but, say, in A_h; namely, $q = (i,h)$. However, if q is in $A_h \neq A_1$, then we can choose another x' that is spelled out by a path from $(1,h)$ to (i,h) in A_h such that $v' = x'uy'$ as illustrated in Fig. 2.

Now we know that $q = (i,1)$ and we show that $i \neq m$. If $i = m$, then this implies that $L(A)$ is not prefix-free and, therefore, not an intercode since we need to spell out u from A_2 and, eventually, there are two strings accepted by A_h and one of them is a prefix of the other as illustrated in Fig. 3.

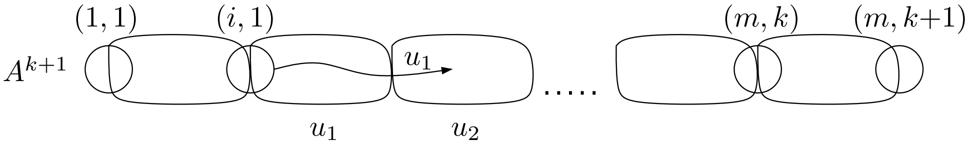

Figure 3. A case when $i = m$. Then, later we must reach a state, which is not a final state when reading each u_h of u from i and it follows that $L(A)$ is not prefix-free. This contradicts our assumption that $L(A)$ bifix-free. Therefore, $i < m$.

Since $u \in L(A^k)$ and $v \in L(A^{k+1})$, there should be two distinct sequences of transitions, one of which is from $(1,1)$ to (m,k) and the other is from $(i,1)$ from (j,g), and both spell out the same string u. Now we prove that we are not at $(m, k+1)$ after reading u from $(i, 1)$. Since $y \neq \lambda$ and A^{k+1} is non-exiting, we must arrive at some state q' such that $q' \neq (m, k+1)$. It follows that there is a path from $((i, 1), (1, 1))$ to $((j, g), (m, k))$ in $G_{A^{k+1}}$ such that $1 < i < m$ and $(j, g) \neq (m, k+1)$ — a contradiction. □

Based on Lemma 3.1, we design an algorithm for checking the k-intercode property as follows:

```
$k$-intercode ($A$, $k$)
/* $A$ is an input FA and $k$ is a fixed index. */

Construct $A^{k+1}$ by catenating $k+1$ $A$s

Construct $G_{A^{k+1}} = (V_G, E_G)$ from $A^{k+1}$

for each node $((i,1),(1,1))$ in $V_G$, where $1 < i < m$
    DFS($((i,1),(1,1))$) in $G_{A^{k+1}}$
    if we meet a node $((j,g),(m,k))$ for any $(j,g) \neq (m,k+1)$
        then output $L(A)$ is not a $k$-intercode

output $L(A)$ is a $k$-intercode
```

Figure 4. A k-intercode checking algorithm for a given FA.

A sub-function DFS($((i, j), (i', j'))$) in Fig. 4 is a depth-first search (DFS) that starts at a node $((i, j), (i', j'))$ in $G_{A^{k+1}}$. Although DFS($((i, j), (i', j'))$) is executed several times inside the **for** loop in the algorithm, each node in $G_{A^{k+1}}$ is visited at most twice and thus, the total time complexity of exploring G_A is linear in the size of G_A. For details on DFS, we refer the reader to Cormen et al. [4]. Since $|A^{k+1}| = (k+1) \cdot O(|Q| + |\delta|)$, the construction of $G = (V_G, E_G)$ from A^{k+1} takes $k^2 \cdot O(|Q|^2 + |\delta|^2)$ time in the worst-case. Therefore, the total running time of the algorithm in Fig. 4 is $k^2 \cdot O(|Q|^2 + |\delta|^2)$ and we obtain the following result.

Lemma 3.2. Given an FA $A = (Q, \Sigma, \delta, s, f)$ and an index k, we can determine whether or not $L(A)$ is a k-intercode in $k^2 \cdot O(|Q|^2 + |\delta|^2)$ worst-case time.

If a regular language is given by a regular expression E, then we can use the Thompson construction [22] that gives a $k^2 \cdot O(|E|^2)$ runtime algorithm since the number of states and the number of transitions of the Thompson automata are of the order $O(|E|)$. Note that if a given language L is context-free, then it is undecidable whether or not L is an intercode [16].

Next we continue with the question of deciding whether a given regular language is an intercode when the index is not specified.

Lemma 3.3. Given an FA $A = (Q, \Sigma, \delta, s, f)$, $L(A)$ is not an intercode for any index if there is a string w that is spelled out by a path from $(i, 1)$ to (i, p) in A^{k+1} and a path from $(1, 1)$ to (m, k) in A^k for some k, where $i \neq 1$, $i \neq m$ and $1 \leq p \leq k + 1$.

Proof:

If a language L is an intercode of index k, then L is an intercode of index $k+1$ [21]. Because of w, $L(A)$ is not a k-intercode and, thus, $L(A)$ is not an intercode for any index less than k. We now show that $L(A)$ is not a $2k$-intercode.

Since w is spelled out by a path from $(i, 1)$ to (i, p) in A^{k+1}, there is a path from $(i, 1)$ to $(i, 2p-1)$ for ww in A^{2k+1}. Since ww is also accepted by A^{2k}, it follows that $L(A)$ is not a $2k$-intercode. Using this argument inductively, it follows that $L(A)$ is not an intercode of any, arbitrarily large, index. □

Lemma 3.3 suggests that if we can find a string w as in the lemma, then we can show that $L(A)$ is not an intercode.

Lemma 3.4. Given an FA $A = (Q, \Sigma, \delta, s, f)$, $L(A)$ is not an intercode for any index k if $L(A)$ is not a $(|Q|+1)$-intercode.

Proof:
First, we show that if $L(A)$ is not an intercode of index $|Q|+1$, then there is $c > 0$ such that $L(A)$ is not an intercode of index $|Q|+1+c$.

Let $t = |Q|+1$. Since $L(A)$ is not a t-intercode, there are two strings $u \in L(A^t)$ and $v \in L(A^{t+1})$ such that u is a strict infix of v; namely, $v = xuy$, and x and y are not λ.

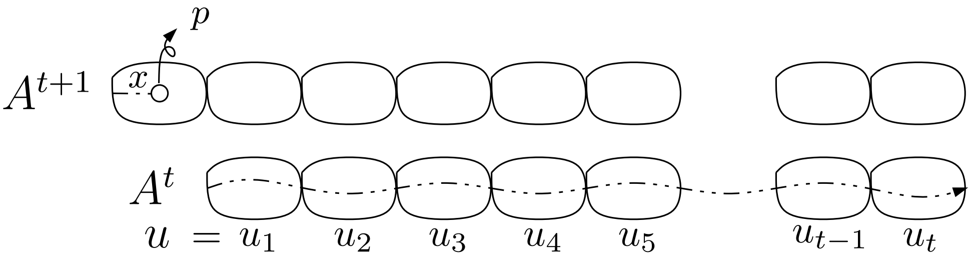

Figure 5. An illustration of two strings u and v, where u is a strict infix of v.

Once we read x in A^{t+1}, we reach a state p, which is not the start state since A is non-returning. We now start reading u from p in A_r, $r \geq 1$, of A^{t+1} as shown in Fig. 5. Note that $u = u_1u_2u_3 \cdots u_t$, where u_i is spelled out by A_i in A^t for $1 \leq i \leq t$. Further, note that $u_i \neq \lambda$, $1 \leq i \leq t$, since $L(A)$ is bifix-free. When we have completed reading each u_i, $1 \leq i \leq t$, we keep a record of the states of A^{t+1} that we reach at that point. Since we have $t = |Q|+1$ such states, two of them must be the same state j of A as shown in Fig. 6. Let a be the "distance" between the two js in A^{t+1} in terms of the different components A, that is, if the first occurrence of j is a state of A_r, the second is a state of A_{r+a}. Let b be the number of infixes u_i that the path between the js spells out, for $1 \leq i \leq t$. Note that $a \geq 0$ and $b > 0$. We use $u'' = u_iu_{i+1} \cdots u_{i+b-1}$ to denote the string spelled out by the path between the two js.

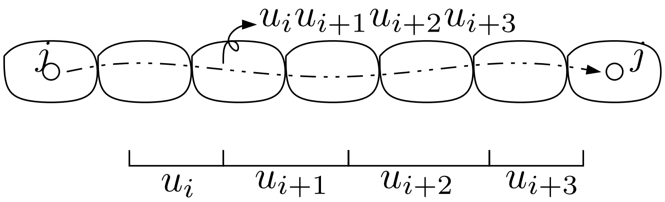

Figure 6. An example of a case when state j appears twice while reading u in A^{t+1} from x shown in Fig. 5. In this case, $a = 6$ and $b = 4$.

Since the state j appears twice, we can define new strings $u' = u_1 u_2 \cdots u'' u'' \cdots u_t$ and $v' = xu'y$, where $u' \in L(A^{t+b})$ and $v' \in L(A^{t+1+a})$. Note that u' is a strict infix of v'. This implies that

$$L(A^{t+1+a}) \cap \Sigma^+ L(A^{t+b}) \Sigma^+ \neq \emptyset. \tag{1}$$

Based on (1), we show that $L(A)$ is not an intercode of index $t{+}c$, for some $c > 0$.

1. $a = 0$: If $a = 0$, then u'' is spelled out by revisiting the same state j in A_i. From (1), we have

$$\begin{aligned} & L(A^{t+1}) \cap \Sigma^+ L(A^{t+b}) \Sigma^+ \neq \emptyset \\ \Rightarrow\ & L(A^{t+1+b}) \cap \Sigma^+ L(A^{t+b}) \Sigma^+ \neq \emptyset, (b > 0). \end{aligned}$$

   Therefore, $L(A)$ is not an intercode of index $t{+}b$ and recall that $b > 0$. Thus, $L(A)$ is not an intercode of index $t{+}c$ when we choose $c = b$.

2. $a, b > 0$: There are two cases to consider separately.

   (a) $a \leq b$: If $L(A^{t+1+a}) \cap \Sigma^+ L(A^{t+b}) \Sigma^+ \neq \emptyset$, then, $L(A^{t+1+b}) \cap \Sigma^+ L(A^{t+b}) \Sigma^+$ must be not empty since $a \leq b$. We can choose c to be b, and now $L(A)$ is not an intercode of index $t{+}c$.

   (b) $a > b$: Because of (1), we have two strings u' and v' as shown in Fig. 7.
   Now we remove v'_1 and v'_{t+1+a} from $v' = v'_1 v'_2 \cdots v'_{t+1+a}$, where v'_i is an infix of v' that is spelled out by A_i of A^{t+1+a} for $1 \leq i \leq t+1+a$. Let v'' be the resulting string. Then, v'' becomes a strict infix of u' as illustrated in Fig. 8.
   We now have

$$\begin{aligned} & L(A^{t+b}) \cap \Sigma^+ L(A^{t+a-1}) \Sigma^+ \neq \emptyset \\ \Rightarrow\ & L(A^{t+1+(b-1)}) \cap \Sigma^+ L(A^{t+(a-1)}) \Sigma^+ \neq \emptyset. \end{aligned}$$

   Since $a{-}1 > b{-}1$, this case is analogous to the previous case when $a \leq b$. Therefore, $L(A)$ is not an intercode of index $t{+}(a{-}1)$. Note that $(a{-}1) > 1$ and, thus, we can take $(a{-}1)$ as the value c. Then, $L(A)$ is not an intercode of index $t{+}c$.

To summarize, we have shown that if $L(A)$ is not an intercode of index t, $t \geq |Q| + 1$, then $L(A)$ is not an intercode of index $t{+}c$, for some $c > 0$. Consequently, if $L(A)$ is not an intercode of index $|Q|{+}1$, then $L(A)$ is not an intercode of any index. □

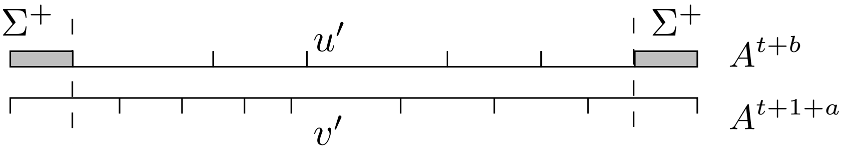

Figure 7. An example for the proof: Two strings u' and v' such $u' \in L(A^{t+b})$, $v' \in L(A^{t+1+a})$ and u' is a strict infix of v', and $a > b$. A gray part is spelled out by Σ^+.

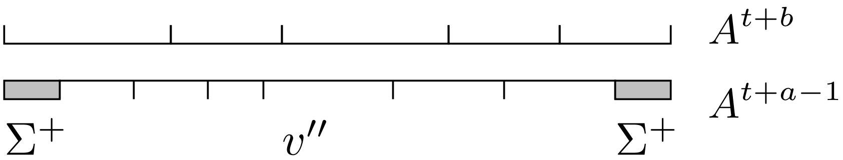

Figure 8. An example for the proof: After we remove v'_1 and v'_{k+1+a} from v', v'' becomes a strict infix of u'.

Based on Lemmas 3.2 and 3.4, we obtain the following result.

Theorem 3.1. Given an FA $A = (Q, \Sigma, \delta, s, f)$, we can determine whether or not $L(A)$ is an intercode of index k, for some k, in $O(|Q|^4 + |Q|^2|\delta|^2)$ worst-case time.

Proof:
Using the algorithm in Fig. 4, we can check whether or not $L(A)$ is a $(|Q|{+}1)$-intercode. The runtime is $O(|Q|^4 + |Q|^2|\delta|^2)$ from Lemma 3.2. If $L(A)$ is not a $(|Q|{+}1)$-intercode, then $L(A)$ is not an intercode at all by Lemma 3.4. □

Note that Theorem 3.1 gives a polynomial-time algorithm to decide the general intercode property, and the input automaton A can be nondeterministic. The previously known decidability result [16] does not yield a polynomial-time algorithm when the input automaton is nondeterministic. Moreover, as an extension of Theorem 3.1, we can compute the smallest index k such that $L(A)$ is a k-intercode. Assume that $L(A)$ is a $(|Q|{+}1)$-intercode for an FA $A = (Q, \Sigma, \delta, s, f)$ from Theorem 3.1. Since intercodes form a proper hierarchy with respect to their index [15], we can repeat the checking procedure for indices $|Q|, |Q|{-}1, \ldots$, until we find the smallest index. However, instead of going down linearly, we can search the smallest index using a binary technique as follows. We jump to the index $|Q|/2$. If $L(A)$ is a $(|Q|/2)$-intercode, then we jump to index $|Q|/4$. Otherwise, we jump back to index $3|Q|/4$. Based on this technique, we establish the following result.

Theorem 3.2. Given an FA $A = (Q, \Sigma, \delta, s, f)$, in $O(\log|Q| \cdot (|Q|^4 + |Q|^2|\delta|^2))$ worst-case time, we can determine whether or not $L(A)$ is an intercode for some index $k > 0$, and if the answer is positive we can find the smallest index l such that $L(A)$ is an l-intercode but not an $(l{-}1)$-intercode.

4. Prime intercode regular languages and decomposition

Decomposition can be viewed as the reverse operation for catenation. Let L, L_1 and L_2 be languages. If L has a decomposition $L = L_1 \cdot L_2$, we call L_1 and L_2 *factors* of L. Note that every language L has trivial decompositions, $L = \{\lambda\} \cdot L = L \cdot \{\lambda\}$. We call $\{\lambda\}$ a *trivial* language. We define a language L to be *prime* if $L \neq L_1 \cdot L_2$, for any non-trivial languages L_1 and L_2. A *prime decomposition* of L is a decomposition $L = L_1 L_2 \cdots L_k$, where $L_1, L_2, \cdots, L_k$ are prime languages and $k \geq 1$.

Mateescu et al. [18, 19] showed that the primality of regular languages is decidable and the prime decomposition of a regular language is not unique even for finite languages. Czyzowicz et al. [5] considered prefix-free regular languages and showed that the prime prefix-free decomposition for a prefix-free regular language L is always unique and the unique prime decomposition for L can be computed in $O(m)$ worst-case time, where m is the size of the minimal DFA for L. Recently, Han et al. [12] investigated the prime infix-free decomposition of infix-free regular languages and demonstrated that the prime infix-free decomposition is not unique. On the other hand, it turns out that the prime outfix-free decomposition of outfix-free regular languages is unique [14].

4.1. Prime intercode regular languages

In this section we examine prime intercode regular languages and decompositions of intercode regular languages.

Definition 4.1. We define a regular language L to be a *prime intercode* language if $L \neq L_1 \cdot L_2$, for any non-trivial intercode regular languages L_1 and L_2.

We define structural properties of DFAs that are useful in finding prime decompositions of intercodes. Recall that since an intercode is necessarily bifix-free, we can, without loss of generality, assume that a DFA accepting an intercode is non-exiting and has only one final state.

Definition 4.2. Let A be a DFA such that $L(A)$ is an intercode. We define a state b of A to be a *bridge state* if the following conditions hold:

1. The state b is neither a start nor the final state.
2. For any string $w \in L(A)$, its path in A must pass through b at least once.
3. The state b does not belong to any cycle of A.
4. If we construct DFAs A_1 and A_2 as described in Definition 4.3, the languages $L(A_1)$ and $L(A_2)$ are intercodes.

We say that a state b of a DFA A is a *candidate bridge state* if it satisfies conditions 1., 2. and 3. of Definition 4.2.

Definition 4.3. Given an intercode DFA $A = (Q, \Sigma, \delta, s, f)$ with a candidate bridge state $b \in Q$, we can partition A into two subautomata A_1 and A_2, that share only the state b, as follows:

- $A_1 = (Q_1, \Sigma, \delta_1, s, b)$,

  Q_1 is the set of states that appear on some path from s to b in A including both s and b.

δ_1 is the set of transitions that appear on some path from s to b in A.

- $A_2 = (Q_2, \Sigma, \delta_2, b, f)$,

  Q_2 is the set of states that appear on some path from b to f in A including both b and f.

  δ_2 is the set of transitions that appear on some path from b to f in A.

Note that if A does not satisfy the third condition in Definition 4.2, then for A_1 and A_2 as constructed in Definition 4.3, $L(A_1)$ and $L(A_2)$ may not be intercodes since FAs for intercode regular languages must be non-returning and non-exiting. Thus, condition 3. of Definition 4.2 follows from condition 4. We include condition 3. in the definition for clarity.

The following result is crucial for finding efficiently prime decompositions of intercode regular languages.

Theorem 4.1. An intercode regular language L is prime if and only if the minimal DFA A for L does not have any bridge states.

Proof:
Let s denote the start state and f denote the final state in A. Note that since an intercode is always bifix-free, the minimal DFA for L has only one final state.
$\Longrightarrow$ Assume that A has a bridge state q. Then, we can construct from A two automata A_1 and A_2 as in Definition 4.3 such that s is the start state and q is the final state of A_1 and q is the start state and f is the final state of A_2. Then, $L = L(A_1) \cdot L(A_2)$, where $L(A_1)$ and $L(A_2)$ are intercodes — a contradiction.
$\Longleftarrow$ Assume that L is not prime. Then, L can be represented as $L_1 \cdot L_2$, where L_1 and L_2 are intercodes; namely, $L = L_1 \cdot L_2$. Czyzowicz et al. [5] showed that given prefix-free languages A, B and C such that $A = B \cdot C$, A is regular if and only if B and C are regular. Thus, if L is regular, then L_1 and L_2 must be regular since all intercodes are prefix-free. Let A_1 and A_2 be minimal DFAs for L_1 and L_2, respectively. Since A_1 and A_2 are non-returning and non-exiting, there is only one start state and one final state for A_1 and A_2. We catenate A_1 and A_2 by merging the final state of A_1 and the start state of A_2 as a single state q. Then, it is easy to verify that the catenated automaton is the minimal DFA for $L(A_1) \cdot L(A_2) = L$ and it has a bridge state q — a contradiction. □

4.2. Prime decomposition of intercode regular languages

Here we develop an algorithm to find the prime decomposition of an intercode regular language. The prime decomposition of an intercode regular language L represents L as a catenation of prime intercode regular languages, and the rough idea is as follows. If L is prime, then L itself is a prime decomposition. Thus, given L, we first check whether or not L is prime and decompose L if it is not prime. If L is not prime, by Theorem 4.1, we can decompose L into $L(A_1)$ and $L(A_2)$ at some bridge state. If both $L(A_1)$ and $L(A_2)$ are prime, a prime decomposition of L is $L(A_1) \cdot L(A_2)$. Otherwise, we repeat the preceding procedure for a non-prime language.

Let B denote the set of bridge states for a given minimal DFA A. The number of states in B is at most m, where m is the number of states in A. Note that once we partition A at $b \in B$ into A_1 and A_2, then only states in $B \setminus \{b\}$ can be bridge states of A_1 and A_2. Therefore, we can determine the primality of $L(A)$ by checking whether A has bridge states and can compute a prime decomposition of

$L(A)$ using these bridge states. Since there are at most m bridge states in an intercode FA A, we can compute a prime decomposition of $L(A)$ after a finite number of decompositions at bridge states.

Recall that if a state q in A satisfies the first three conditions of Definition 4.2, we call q a candidate bridge state. We can compute the set of candidate bridge states from a given minimal DFA $A = (Q, \Sigma, \delta, s, f)$ for an intercode regular language $L(A)$ in linear time using the DFS [12].

Once we compute a set $\mathcal{C}$ of candidate bridge states from A, we check for each state $b_i \in \mathcal{C}$ whether or not two subautomata A_1 and A_2 that are partitioned at b_i are intercodes using the algorithm in Fig. 4. If both A_1 and A_2 are intercodes, then L is not prime and we decompose L into $L(A_1) \cdot L(A_2)$ and continue to check and decompose each of the "subautomata" A_1 and A_2, respectively, using the remaining states in $\mathcal{C} \setminus \{b_i\}$.

The correctness of the recursive procedure relies on the "if and only if" condition given by Theorem 4.1 that in turn relies on the minimality of the DFA's in question. Hence we still need to verify the following technical property.

Lemma 4.1. Let $A = (Q, \Sigma, \delta, s, f)$ be a DFA with a candidate bridge state $b \in Q$. Let A_1 and A_2 be the subautomata of A that share the state b and are constructed as in Definition 4.3. If A is minimal, then both A_1 and A_2 are minimal DFAs.

Proof:
Assume that A is minimal. We use for A_1 and A_2 the notations as in Definition 4.3. Since all states of A_i are clearly reachable from the start state, it is sufficient to show that no two states in A_i can be equivalent, for $i = 1, 2$.

First consider distinct states q_1 and q_2 of A_2. Since A is minimal there exists $w \in \Sigma^*$ that distinguishes between the states q_1 and q_2. Without loss of generality, we assume that $\delta(q_1, w) = f$ and $\delta(q_2, w) \neq f$ since the other possibility is symmetric. Above it is possible that $\delta(q_2, w)$ is undefined. Since b is a candidate bridge state of A, A_1 cannot have any out-transitions from b. This means that the computations along w starting from q_1 and q_2, respectively, are the same in A_2 as in A. Hence, q_1 and q_2 are not equivalent in A_2.

Second consider distinct states p_1 and p_2 of A_1. Again, since A is minimal there exists $u \in \Sigma^*$ such that

$$\delta(p_1, u) = f \text{ and } \delta(p_2, u) \neq f \tag{2}$$

(or vice versa). By condition 2. of Definition 4.2, some prefix u_1 of u takes p_1 to the state b. Now (2) implies that $\delta(p_2, u_1) \neq b$. We note that $\delta_1(p_1, u_1) = \delta(p_1, u_1) = b$. If the computation of A starting from p_2 on input u_1 does not pass through the state b, we have $\delta_1(p_2, u_1) = \delta(p_2, u_1)$, and otherwise $\delta_1(p_2, u_1)$ is undefined. In both cases p_1 and p_2 are inequivalent in A_1. □

Theorem 4.2. Given a minimal DFA $A = (Q, \Sigma, \delta, s, f)$ for an intercode regular language $L(A)$, we can determine primality of $L(A)$ in $O(m^5)$ worst-case time and compute a prime decomposition of $L(A)$ in $O(m^6)$ worst-case time, where m is the number of states in A.

Proof:
First, we compute the set $\mathcal{C}$ of candidate bridge states in linear time in the size of A [12]. Note that the number of states in $\mathcal{C}$ is at most m by definition, where $m = |Q|$. For each state in $\mathcal{C}$, we check whether or not $L(A_1)$ and $L(A_2)$ are intercodes in $O(m^4)$ time. Note that a state q of A_i, $1 \leq i \leq 2$, can be

a candidate bridge state only if q was a candidate bridge state of the original DFA A. Thus, the total running time for determining primality of $L(A)$ is $O(m) \times O(m^4) = O(m^5)$ in the worst-case.

Once we find a bridge state b_j, we partition A into A_1 and A_2 at b_j and repeat the procedure for $L(A_1)$ and $L(A_2)$, respectively, using the remaining candidate states in $\mathcal{C} \setminus \{b_j\}$. By Lemma 4.1, A_r is a minimal DFA and $L(A_r)$ is an intercode since b_j was a bridge state, $r = 1, 2$. Thus, by Theorem 4.1, $L(A_r)$ is prime if and only if A_r does not have any bridge states, $r = 1, 2$.

We continue this partitioning until the component languages are prime intercodes. Therefore, the total time complexity for computing a prime decomposition of $L(A)$ is $O(m^6)$ in the worst-case. □

The algorithm for computing a prime decomposition for $L(A)$ in Theorem 4.2 looks similar to the algorithm [12] for the infix-free regular language case. However, there is one crucial difference between these two algorithms because of the different closure properties of the two families. Many classes of codes are closed under catenation; examples include the prefix-free, bifix-free, infix-free and outfix-free codes. Based on this observation, Han et al. [12] speeded up the algorithm for the infix-free case by a linear factor. In contrast, intercodes are not closed under catenation.

Theorem 4.3. The family of intercodes is closed under intersection but not closed under catenation, union, complement or star.

Proof:
We consider here only the case of catenation. The other cases can be proved straightforwardly.

Assume that L is an intercode and let L_0 be the the square of L; namely, $L_0 = LL$. If L_0 is an intercode, then $L_0^{k+1} \cap \Sigma^+ L_0^k \Sigma^+$ must be $\emptyset$ for some integer $k \geq 1$. However, we observe that for any $k \geq 1$,

$$L_0^{k+1} \cap \Sigma^+ L_0^k \Sigma^+ = LL^{2k}L \cap \Sigma^+ L^{2k} \Sigma^+ \neq \emptyset.$$

Therefore, L_0 is not an intercode (of any index) and the class of intercodes is not closed under catenation. □

In the proof of Theorem 4.2 we observed that all bridge states of the component automata A_i, $1 \leq i \leq 2$, must be bridge states also in the original DFA A. However, the implication does not hold in the converse direction and sometimes a bridge state $b_i \in \mathcal{C}$ of a minimal DFA A is no longer a bridge state after a decomposition at some other bridge state b_j of A. Fig. 9 illustrates this situation.

The example of Fig. 9 hints at the possibility that the prime intercode decomposition might not be unique. Czyzowicz et al. [5] demonstrated that the prime prefix-free decomposition for a prefix-free regular language is unique; this can be extended for the suffix-free and bifix-free cases. Since intercodes are a subfamily of bifix-free languages, it is natural to investigate the uniqueness of prime intercode decompositions.

Example 4.1. The following example shows that the prime intercode decomposition need not be unique.

$$L(a(bcb+c)a) = \begin{cases} L_1(a(bcb+c)) \cdot L_2(a). \\ L_2(a) \cdot L_3((bcb+c)a). \end{cases}$$

The language L is an intercode but not prime and it has two different prime decompositions, where L_1, L_2 and L_3 are prime intercodes.

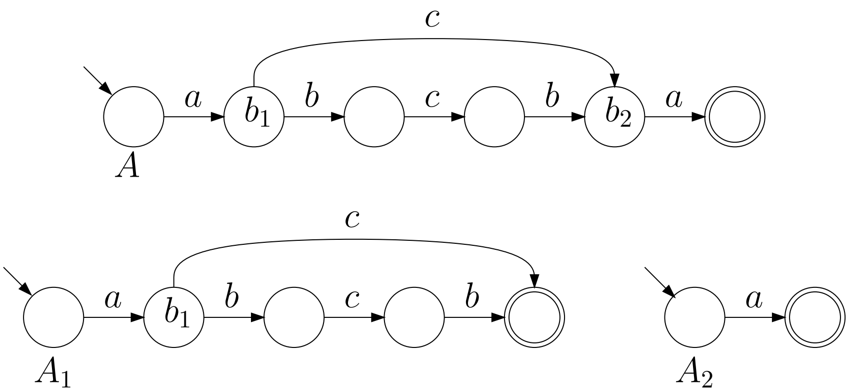

Figure 9. States b_1 and b_2 are bridge states for A. However, once we decompose A at b_2, then b_1 is no longer a bridge state in A_1 since b_1 now violates the fourth condition in Definition 4.2. Similarly, if we decompose A at b_1, then b_2 is not a bridge state.

In Example 4.1, L, L_1, L_2 and L_3 are all 1-intercodes. However, $L' = (bcb + c)$ is not an intercode for any index by Lemma 3.3 since $cbcb \in L(A^2)$ is spelled out by a path from $(2, 1)$ to $(2, 3)$ in A^3; see Fig. 10 for an example. Therefore, the prime intercode decomposition is not unique.

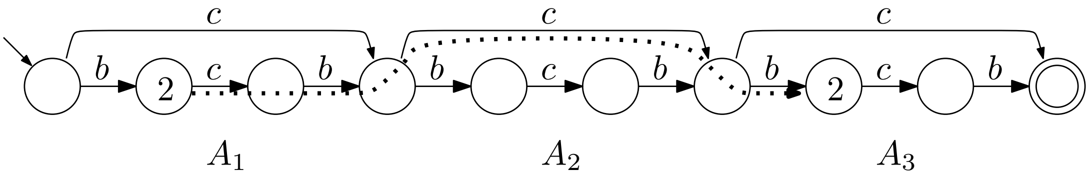

Figure 10. Given a minimal DFA A for $L' = (bcb + c)$, we construct A^3 as a catenation of three As. The dotted line represents a path from $(2, 1)$ to $(2, 3)$ that spells out $cbcb \in L(A^2)$.

5. Conclusion

There has been much research on formal languages aspects of codes. With this viewpoint, we have investigated regular intercodes, their decision properties and prime decompositions.

Given a regular language L and a fixed index k, it is not difficult to determine whether or not L is an intercode of index k. On the other hand, if no index is given, then the decision problem is not as straightforward. We have given an algorithm that determines in polynomial time whether or not the language $L(A)$ of a given NFA A is an intercode (of any index). The algorithm relies, via state-pair graphs, on the structural properties of a given NFA. Furthermore, we have shown that in the positive case we can compute, in polynomial time as well, the smallest index for which L is an intercode. If L is defined by a regular expression E, then we can use the Thompson construction [22] that guarantees that the size of the corresponding automaton is linear in the size of E.

We have provided an algorithm for determining the primality of an intercode regular language and also provided an efficient algorithm for computing a prime intercode decomposition. Finally, we have presented an example that shows the non-uniqueness of prime intercode decompositions.

Acknowledgements

We thank the anonymous referees for useful suggestions.

References

[1] Berstel, J., Perrin, D.: *Theory of Codes*, Academic Press, Inc., 1985.

[2] Berstel, J., Perrin, D.: Trends in the theory of codes, *EATCS* Bulletin, **29**, 1986, 84–95.

[3] Caron, P., Ziadi, D.: Characterization of Glushkov automata, *Theoretical Computer Science,* **233**(1–2), 2000, 75–90.

[4] Cormen, T.H., Leiserson, C.E., Rivest, R.L.,Stein, C.: *Introduction to Algorithms,* McGraw-Hill Higher Education, 2001.

[5] Czyzowicz, J., Fraczak, W., Pelc, A., Rytter, W.: Linear-time prime decomposition of regular prefix codes, *International Journal of Foundations of Computer Science,* **14**, 2003, 1019–1032.

[6] Fernau, H., Reinhardt, K., Staiger, L.: Decidability of code properties, *Proc. 4th International Conference Developments in Language Theory,* (G. Rozenberg, W. Thomas, Eds.) World Scientific, Singapore, 2000, 153–160.

[7] Giammarresi, D., Ponty, J.-L., Wood, D., Ziadi, D.: A characterization of Thompson digraphs, *Discrete Applied Mathematics,* **134**, 2004, 317–337.

[8] Glushkov, V.: The abstract theory of automata, *Russian Mathematical Surveys,* **16**, 1961, 1–53.

[9] Golomb, S., Gordon, B.,Welch, L.: Comma-free codes, *The Canadian Journal of Mathematics,* **10**, 1958, 202–209.

[10] Han, Y.-S., Salomaa, K., Wood, D.: Prime decompositions of regular languages, *Proceedings of DLT'06,* LNCS 4036, Springer-Verlag, 2006, 145–155.

[11] Han, Y.-S., Wang, Y., Wood, D.: Prefix-free regular-expression matching, *Proceedings of CPM'05*, LNCS 3537, Springer-Verlag, 2005, 298–309.

[12] Han, Y.-S., Wang, Y., Wood, D.: Infix-free regular expressions and languages, *International Journal of Foundations of Computer Science,* **17**(2), 2006, 379–393.

[13] Han, Y.-S., Wood, D.: Overlap-free regular languages, *Proceedings of COCOON'06,* LNCS 4112, Springer-Verlag, 2006, 469–478.

[14] Han, Y.-S., Wood, D.: Outfix-free regular languages and prime outfix-free decomposition, *Proceedings of ICTAC'05,* LNCS 3722, Springer-Verlag, 2005, 96–109.

[15] Jürgensen, H., Konstantinidis, S.: Codes, in: *Word, Language, Grammar,* volume 1 of *Handbook of Formal Languages* (G. Rozenberg, A. Salomaa, Eds.), Springer-Verlag, 1997, 511–607.

[16] Jürgensen, H., Salomaa, K., Yu, S.: Decidability of the intercode property, *Elektronische Informationsverarbeitung und Kybernetik,* **29**(6), 1993, 375–380.

[17] Jürgensen, H., Salomaa, K., Yu, S.: Transducers and the decidability of independence in free monoids, *Theoretical Computer Science,* **134**, 1994, 107–117.

[18] Mateescu, A., Salomaa, A., Yu, S.: On the decomposition of finite languages, Technical Report 222, TUCS, 1998.

[19] Mateescu, A., Salomaa, A., Yu, S.: Factorizations of languages and commutativity conditions, *Acta Cybernetica*, **15**(3), 2002, 339–351.

[20] McNaughton, R., Yamada, H.: Regular expressions and state graphs for automata, *IEEE Transactions on Electronic Computers*, **9**, 1960, 39–47.

[21] Shyr, H., Yu, S.S.: Intercodes and some related properties, *Soochow J. Math.*, **16**(1), 1990, 95–107.

[22] Thompson, K.: Regular expression search algorithm, *Communications of the ACM*, **11**, 1968, 419–422.

[23] Yu, S.S.: A characterization of intercodes, *International Journal of Computer Mathematics*, **36**, 1990, 39–45.