Enumerating Cryptarithms Using Deterministic Finite Automata

Yuki Nozaki\textsuperscript{1}, Diptarama Hendrian\textsuperscript{1}, Ryo Yoshinaka\textsuperscript{1}, Takashi Horiyama\textsuperscript{2}, and Ayumi Shinohara\textsuperscript{1}

\textsuperscript{1} Graduate School of Information Sciences, Tohoku University
yuki_nozaki@shino.ecei.tohoku.ac.jp
\{diptarama@, ryoshinaka@, ayumis\}@tohoku.ac.jp

\textsuperscript{2} Graduate School of Science and Engineering, Saitama University
horiyama@al.ics.saitama-u.ac.jp

August 1, 2018

Abstract

A cryptarithm is a mathematical puzzle where given an arithmetic equation written with letters rather than numerals, a player must discover an assignment of numerals on letters that makes the equation hold true. In this paper, we propose a method to construct a DFA that accepts cryptarithms that admit (unique) solutions for each base. We implemented the method and constructed a DFA for bases $k \leq 7$. Those DFAs can be used as complete catalogues of cryptarithms, whose applications include enumeration of and counting the exact numbers $G_k(n)$ of cryptarithm instances with $n$ digits that admit base-$k$ solutions. Moreover, explicit formulas for $G_2(n)$ and $G_3(n)$ are given.

1 Introduction

A cryptarithm is a mathematical puzzle where a given arithmetic formula consisting of letters rather than numerals, players try to find an injective substitution of numerals for letters that makes the formula hold true. Figure 1 shows a well-known example of a cryptarithm and its solution. To solve a cryptarithm is, in principle, not quite hard. One can find a solution (if any) by trying at most $10!$ assignments of numerals on letters, i.e., cryptarithms are solvable by brute force in linear time. Nevertheless, cryptarithms have been an interesting topic of computer science \cite{8} and different methods for solving cryptarithms have been proposed \cite{1, 9} including a number of online solvers on the web (e.g., \cite{2, 3, 10}). In fact, although cryptarithms can be solved in linear time under the decimal system, Eppstein \cite{7} showed that to decide whether a given cryptarithm has a

\textsuperscript{*}This manuscript comprehends and improves our paper \cite{10} in the proceedings of CIAA 2018.

\textsuperscript{†}He is currently working in VOYAGE GROUP, Inc.
solution under the base-$k$ system is strongly NP-complete when $k$ is not fixed. His discussions involve only arithmetic formulas with just one addition, like the one in Figure 1. Following Eppstein, this paper focuses on such formulas only.

A cryptarithm example that has a binary solution but no decimal solution is shown in Figure 2.

\[
\begin{array}{c}
s& e & n & d \\
+ & m & o & r & e \\
\hline 
 m & o & n & e & y \\
\end{array}
\quad
\begin{array}{c}
P \\
+ & P \\
\hline 
P & A \\
\end{array}
\]

\[
\begin{array}{c}
9 & 5 & 6 & 7 \\
+ & 1 & 0 & 8 & 5 \\
\hline 
1 & 0 & 6 & 5 & 2 \\
\end{array}
\quad
\begin{array}{c}
1 \\
+ & 1 \\
\hline 
1 & 0 \\
\end{array}
\]

Figure 1: Example of a cryptarithm and its solution

Figure 2: Cryptarithm solvable under the binary system

Our goal is not only to provide a cryptarithm solver but to propose a method to enumerate cryptarithms for different base systems. Towards the same goal, Endoh et al. [6] presented a method for constructing a deterministic finite automaton (DFA) that accepts cryptarithms solvable under the base-$k$ system for $k = 2, 3, 4$. Their method constructs the goal DFA as the product of several auxiliary DFAs corresponding to different conditions that solvable cryptarithms must satisfy. On the other hand, our proposed method constructs the objective DFA directly. This approach enabled us to construct the goal DFAs for $k \leq 7$.

Those DFAs can be seen as complete catalogues of cryptarithms for different bases. Once the cryptarithm DFA for base-$k$ arithmetics is constructed, this can be used as a cryptarithm solver using the information added to its states that runs in linear time in the size of the input with no huge coefficient. Moreover, different types of analyses on cryptarithms are possible with standard techniques on edge-labeled graphs. For example, one can enumerate all the solvable cryptarithms one by one in the length-lexicographic order. It is also possible to compute the $m$th solvable cryptarithm quickly without enumerating the first $m - 1$ cryptarithms. Counting the number of solvable cryptarithms of $n$ digits is also easy. In particular, we derived explicit formulas for the number $G_k(n)$ of cryptarithms of $n$ digits solvable under the base-$k$ system for $k = 2, 3$ as $G_2(n) = 6 \times 4^{n-2} - 3 \times 2^{n-2}$ and $G_3(n) = 4 \times 9^{n-1} - 2 \times 5^{n-1} - 3^n - 1$, respectively.

2 Preliminaries

For an alphabet $\Sigma$, $\Sigma^*$ and $\Sigma^+$ denote the sets of strings and non-empty strings, respectively. For a map $\theta$ from an alphabet $\Sigma$ to another $\Delta$, its homomorphic extension from $\Sigma^*$ to $\Delta^*$ is denoted by $\hat{\theta}$. For a string or a tuple of strings $w$ over $\Delta$, $\Sigma\upharpoonright w$ denotes the subset of $\Sigma$ consisting of letters occurring in $w$. An extension of a function $f : A \to B$ is a function $g : A' \to B'$ such that $A \subseteq A'$ and $g(x) = f(x)$ for all $x \in A$. The cardinality of a set $A$ is denoted by $|A|$. The length of a string $w$ is also denoted by $|w|$. We let $N_k$ denote the alphabet of numerals $0, \ldots, k - 1$. 

2
2.1 Cryptarithms

A cryptarithm is a triple \( \vec{w} = (w_1, w_2, w_3) \) of non-empty strings over an alphabet \( \Sigma \). Each \( w_i \) is called the \( i^{th} \) term. The size of \( \vec{w} \) is defined to be \( \max\{|w_1|, |w_2|, |w_3|\} \). Any injection from \( \Sigma^* \vec{w} \) to \( N_k \) is called a base-\( k \) assignment for \( \vec{w} \). Moreover it is a base-\( k \) solution if it makes the equation \( \theta(w_1) + \theta(w_2) = \theta(w_3) \) true when interpreting strings over \( N_k \) as numerals in the base-\( k \) system: that is, for \( w_i = w_{i,1}w_{i,2}\ldots w_{i,n} \) with \( w_{i,j} \in \Sigma \), it holds \( \sum_{j=1}^{w_{i,1}} \theta(w_{i,j})k^{j-1} + \sum_{j=1}^{w_{i,2}} \theta(w_{i,j})k^{j-1} = \sum_{j=1}^{w_{i,3}} \theta(w_{i,j})k^{j-1} \) and \( \theta(w_{i,\ell}) \neq 0 \) for each \( i = 1, 2, 3 \). A cryptarithm that admits a solution is said to be base-\( k \) solveable.

Following Endoh et al. [6], in order for DFAs to treat cryptarithms, we convert cryptarithms into single strings over \( \Sigma \cup \{\$\} \) with \$ \not\in \Sigma \) by

\[
\psi(\langle w_1, w_2, w_3 \rangle) = w_{1,1}w_{2,1}w_{3,1}w_{1,2}w_{2,2}w_{3,2}\ldots w_{1,n}w_{2,n}w_{3,n} \\
\]

where \( w_i = w_{i,1}\ldots w_{i,n}, n = \max\{|w_1|, |w_2|, |w_3|\} \), and \( w_{i,j} = $ \) for \( |w_i| < j \leq n \). Such a string \( \psi(\vec{w}) \) is called a cryptarithm sequence.

**Example 1.** Let \( \vec{w} = (\text{send}, \text{more}, \text{money}). \) This admits a unique base-10 solution \( \theta = \{d \mapsto 7, e \mapsto 5, y \mapsto 2, n \mapsto 6, r \mapsto 8, o \mapsto 0, s \mapsto 9, m \mapsto 1\} \). The sequential form of \( \vec{w} \) is \( \psi(\vec{w}) = \text{deynre eon smo} $$m $$$ \). We say that two instances \( \vec{w} \) and \( \vec{v} \) are equivalent if there is a bijection \( \gamma \) from \( \Sigma^* \vec{w} \) to \( \Sigma^* \vec{v} \) such that \( \gamma(\vec{w}) = \vec{v} \). In such a case, an injection \( \theta: \Sigma^* \vec{v} \rightarrow N_k \) is a base-\( k \) solution for \( \vec{v} \) if and only if so is \( \theta \circ \gamma \) for \( \vec{w} \). Fixing the alphabet to be \( \Sigma_k = \{a_1, \ldots, a_k\} \), we define the canonical form among equivalent instances. A base-\( k \) cryptarithm \( \vec{w} \in (\Sigma_k^n)^3 \) is said to be canonical if

- wherever \( a_{i+1} \) occurs in the sequential form \( \psi(\vec{w}) \) of \( \vec{w} \), it is after the first occurrence of \( a_i \) for any \( i \geq 1 \).

Identifying a cryptarithm and its sequential form, we adapt terminology on cryptarithms for cryptarithm sequences as well. For example, the sequential form of a canonical cryptarithm is also called canonical. A solution of a cryptarithm instance is also said to be a solution of its sequential form.

For the ease of presentation, we use Latin letters \( a, b, c, \ldots \) instead of \( a_1, a_2, a_3, \ldots \) when \( k \) is relatively small: \( k \leq 26 \).

**Example 2.** The cryptarithm \( \vec{w} = (\text{send}, \text{more}, \text{money}) \) in Example [1] is not canonical. Its canonical form is \( \vec{v} = (\text{gbda}, \text{hfeb}, \text{hfdbc}) \), whose sequential form is \( \psi(\vec{v}) = \text{abc deb bfd ghf $$h $$$} \).

3 Cryptarithm DFAs

3.1 Naive Cryptarithm DFA

We will define a DFA \( M_k \) that accepts all and only canonical cryptarithms that admit solutions. Our DFA is slightly different from the standard ones. First, each edge is labeled by a trigram so that letters belonging to the same place

---

1 For readability a small space is inserted in every three letters.
will be read at once. Second, it has two distinguishable accepting states \( f_1 \) and \( f_2 \) where cryptarithm sequences with unique and multiple solutions shall be accepted at \( f_1 \) and at \( f_2 \), respectively. Accordingly, our DFA is a sextuple \( M_k = (Q, \Sigma_k, \delta, q_0, f_1, f_2) \) where \( Q_k \) is the state set, \( \delta : Q \times (\Sigma_k \cup \{\$\})^3 \rightarrow Q \) is the transition partial function, and \( f_1 \) and \( f_2 \) are accepting states, which define two languages

\[
L_{k,\text{uniq}} = \{ w \in (\Sigma_k \cup \{\$\})^+ \mid \delta(q_0, w) = f_1 \} = \{ \psi(\vec{w}) \in (\Sigma_k \cup \{\$\})^+ \mid \vec{w} \text{ admits exactly one solution} \},
\]

\[
L_{k,\text{multi}} = \{ w \in (\Sigma_k \cup \{\$\})^+ \mid \delta(q_0, w) = f_2 \} = \{ \psi(\vec{w}) \in (\Sigma_k \cup \{\$\})^+ \mid \vec{w} \text{ admits at least two solutions} \},
\]

where \( \delta \) is the usual extension of \( \delta \) for domain \((\Sigma_k \cup \{\$\})^3)^*\). We call a string \( w \in ((\Sigma_k \cup \{\$\})^3)^* \) valid if it is a prefix of some canonical cryptarithm sequence with at least one solution. We say that an assignment \( \theta : \Sigma_k \rightarrow N_k \) is consistent with \( w \) if there is an extension of \( \theta \) which is a solution of a cryptarithm sequence of which \( w \) is a prefix. When \( \Sigma_k \upharpoonright w = \Sigma_{k-1} \), each consistent assignment on \( \Sigma_{k-1} \) has just one trivial proper extension injection with domain \( \Sigma_k \). Therefore, we “promote” consistent assignments on \( \Sigma_{k-1} \) to their extensions on \( \Sigma_k \). We let \( \Theta(w) \) denote the set of consistent assignments, possibly with promotion:

\[
\Theta(w) = \begin{cases} \{ \theta : \Sigma_k \rightarrow N_k \mid \theta \text{ is consistent with } w \} & \text{if } \Sigma_k \upharpoonright w = \Sigma_{k-1}, \\ \{ \theta : \Sigma_k \upharpoonright w \rightarrow N_k \mid \theta \text{ is consistent with } w \} & \text{otherwise}. \end{cases}
\]

For a valid sequence \( w \), one can characterize succeeding sequences \( v \) that will make \( wv \) a solvable canonical cryptarithm sequence with \( \Theta(w) \) and other parameters. The parameters the DFA \( M_k \) maintains in its states have the form \((d_1, d_2, \ell, P)\), which we will call a configuration. Every state except accepting ones has a unique configuration. Among those parameters, \( d_1, d_2 \in \{0, 1\} \) are used to ensure that a sequence may be extended to a cryptarithm sequence and \( \ell \in \{1, \ldots, k\} \) is used to ensure that a sequence may be extended to a canonical one. The last parameter \( P \) is a non-empty set that remembers possible assignments on letters together with auxiliary information. Suppose that the configuration of the state \( q \) reached from \( q_0 \) by reading a valid sequence \( w \) in \( M_k \) is \((d_1, d_2, \ell, P)\) and let \( w = w_1x_1x_2x_3 \) with \( x_1, x_2, x_3 \in \Sigma_k \cup \{\$\} \) and \( \psi^{-1}(w) = (w_1, w_2, w_3) \). Then,

- \( d_i = 1 \) if \( x_i = \$ \) and \( d_i = 0 \) otherwise for \( i = 1, 2 \),
- \( \ell = \min\{k, |\Sigma|w| + 1\} \),
- \( P \) consists of \([\theta, c, b_1, b_2] \in \Theta(w) \times \{0, 1\}^3 \) where
  - \( \theta \in \Theta(w) \),
  - \( \hat{\theta}(w_1) + \hat{\theta}(w_2) = \hat{\theta}(w_3) + ck^{\left|w_3\right|} \),
  - \( b_i = 0 \) if \( x_i \neq \$ \) and \( \theta(x_i) = 0 \), and \( b_i = 1 \) otherwise, for \( i = 1, 2 \).

One can see \( P \) as a function from \( \Theta(w) \) to \( \{0, 1\}^3 \). For \([\theta, c, b_1, b_2] \in P \), when \( c = 1 \), we have a carry under the assignment \( \theta \). When \( b_i = 0 \), the \( i^{th} \) term must
be extended to have a more significant digit since the current most significant
digit is 0 under \( \theta \).

Now let us define \( M_k = (Q, \Sigma_k, \delta, q_0, f_1, f_2) \) so that \( M_k \) satisfies the above.
We identify a state and its configuration, since no distinct states have the same
configuration: in case two states happen to have the same configuration, they
must be merged. The initial state is the configuration \( (0, 0, 1, \{[0, 0, 0, 0]\}) \),
where \( \emptyset \) is the empty assignment.

The transition function \( \delta \) is defined as follows. For \( x_1, x_2, x_3 \in \Sigma_k \cup \{\$\} \), let
us write \( \langle d_1, d_2, \ell, P \rangle \xrightarrow{z_1z_2z_3} \langle d'_1, d'_2, \ell', P' \rangle \) if

- \( x_3 = \$ \) implies \( x_1 = x_2 = \$ \),
- \( d_i = 1 \) implies \( x_i = \$ \) for \( i = 1, 2 \),
- \( d'_i = 1 \) if \( x_i = \$ \), and \( d'_i = 0 \) otherwise, for \( i = 1, 2 \),
- \( x_1 \in \Sigma_k \cup \{\$\} \), \( x_2 \in \Sigma_{k_2} \cup \{\$\} \), \( x_3 \in \Sigma_{k_3} \cup \{\$\} \), where \( \ell_1 = \ell \) if \( x_1 \in \Sigma_k \)
and \( \ell_1 = \min \{k, \ell + 1\} \) otherwise, and \( \ell_2 \) is defined from \( \ell_1 \) and \( x_2 \) in the
same manner,
- \( \ell' \) is defined from \( \ell_2 \) and \( x_3 \) in the same manner,
- \( P' = \{ p' \mid p \xrightarrow{z_1z_2z_3} p' \text{ for some } p \in P \} \) is not empty,

where we write \( [\theta, c, b_1, b_2] \xrightarrow{z_1z_2z_3} [\theta', c', b'_1, b'_2] \) if

- \( b_i = 0 \) implies \( x_i \neq \$ \) for \( i = 1, 2 \),
- \( \theta' : \Sigma' \rightarrow N_k \) is an extension of \( \theta \) where \( \Sigma' = \Sigma_k \) if \( \ell' = k \), and \( \Sigma' = \Sigma_{k+1} \)
otherwise,
- \( c + \tilde{\theta}'(x_1) + \tilde{\theta}'(x_2) = c'k + \tilde{\theta}'(x_3) \) where \( \tilde{\theta}' \) extends \( \theta' \) by \( \tilde{\theta}'(\$) = 0 \),
- \( b'_i = 0 \) if \( x_i \neq \$ \) and \( \theta'(x_i) = 0 \), and \( b'_i = 1 \) otherwise, for \( i = 1, 2 \).

If \( x_1x_2x_3 \neq $$$ \), then we define \( \delta(q, x_1x_2x_3) = q' \) for \( q \xrightarrow{z_1z_2z_3} q' \). When
\( x_1x_2x_3 = $$$ \), this means the end of the input sequence, if it is a cryptarithm
sequence. For \( q' = (d'_1, d'_2, \ell', P') \) with \( q \xrightarrow{$$$} q' \), we define \( \delta(q, $$$) = f_1 \text{ if } |P'| = 1, \) and \( \delta(q, $$$) = f_2 \text{ if } |P'| \geq 2 \).

The state set \( Q \) is defined to consist of the states reachable from the initial
state according to \( \delta \).

**Example 3.** Let \( k = 3 \). Suppose that a state \( q \) in \( M_3 \) has a configuration
\( (d_1, d_2, \ell, P) = (0, 0, 2, P) \) with

\[
P = \{ \{a \mapsto 0\}, c, b_1, b_2 \} = \{ \{a \mapsto 0\}, 0, 0, 0 \}.
\]

In fact, this state is reached by reading \( aaa \) from the initial state, where we did
not yet find \( \$ \) (so \( d_1 = d_2 = 0 \)), the second letter \( b \) may appear in the nearest
future (so \( \ell = 2 \)), and the only consistent assignment \( \theta \) maps \( a \) to 0 (otherwise
(\( \theta(a) + \theta(a) \neq \theta(a) \)), under which we have no carry (\( c = 0 \)), but each term must
not finish (\( b_1 = b_2 = 0 \)). Therefore, this state \( q \) has no outgoing transition edge
labeled with a trigram including \( \$ \). When reading \( aaa \) again from this state,
the situation does not change. So we have \( \delta(q, aaa) = q \). If we read \( abb \), where
b is a new letter, we reach a new state $q'$. Although the last letter $c$ in $\Sigma_3$ has not appeared yet, it is ready to come. The domain of the assignments in the configuration of $q'$ is now $\Sigma_3$. We have two consistent assignments extending the one $\{a \mapsto 0\}$ in $q$. One maps $b$ to 1 and the other maps $b$ to 2. In both cases, we have no carry and the second term may finish. Thus, the configuration of $q'$ is $\langle 0, 0, 3, P' \rangle$ with

$$P' = \{ \{a \mapsto 0, b \mapsto 1, c \mapsto 2\}, 0, 0, 1\}, \{\{a \mapsto 0, b \mapsto 2, c \mapsto 1\}, 0, 0, 1\} \}.$$

On the other hand, it is not hard to see that there is no $p''$ such that $\{a \mapsto 0\}, 0, 0, 0 \xrightarrow{abc} p''$. Hence $q$ has no edge labeled with $abc$. In this way, we decide whether a state has an outgoing edge labeled with a trigram over $\Sigma_k \cup \{\$\}$ and the configuration of the reached state.

We have now established Equations (1) and (2). An assignment $\theta$ is a solution of a cryptarithm sequence $\$\$ if and only if $[\theta, 0, 1, 1] \in P$ of the configuration $\langle \delta(q_0, w) \rangle$. In other words, one can regard our DFA as a Mealy machine that outputs solutions when reading $\$\$.

We remark that the constructed DFA is minimum as a Mealy machine but is not necessarily minimum if we ignore output solutions. For example, let us consider the states reached by $abc$aba and $abc$aba from the initial state in $M_3$. They have different configurations $(1, 0, 3, P_1)$ and $(1, 0, 3, P_2)$ where $P_1 = \{ \{a \mapsto i, b \mapsto (3-i), c \mapsto 0\}, 0, 1, 1\}$ for $i = 1, 2$. Those states are not merged but the strings that will lead us to the accepting state $f_1$ from those states coincide; namely, they have the form $\$n_1x_1\ldots x_n\$\$ where $x_i \in \{a, b, c\}$ for $i < n$, $x_n \in \{a, b\}$ and $n \geq 0$.

The number of states of $M_k$ is bounded by the number of possible configurations. A trivial and loose upper bound on it is $2^{O(k^3)}$. If one is interested only in cryptarithms with a unique solution, one can remove the state $f_2$. If uniqueness of a solution does not matter, two accepting states $f_1$ and $f_2$ can be merged.

Figure 3 shows the finally obtained automaton for $k = 2$. This automaton $M_2$ misses the accepting state $f_2$, because no cryptarithm has two distinct binary solutions.

### 3.2 Compressed Cryptarithm DFA

By observing Fig. 3 one may realize that the DFA has isomorphic substructures. Namely, the sub-automaton $M_2^2$ whose initial state is set to be 2 is isomorphic to $M_2^{15}$ with initial state 15 by swapping $a$ and $b$ on the edge labels. There exist just two binary assignments, $\{a \mapsto 0, b \mapsto 1\}$ and $\{a \mapsto 1, b \mapsto 0\}$. The first trigram of any cryptarithm sequence uniquely determines one of the two as a consistent assignment. The former assignment corresponds to $M_2^{12}$ and the latter to $M_2^{15}$. We say that two configurations $\langle d_1, d_2, \ell, P \rangle$ and $\langle d'_1, d'_2, \ell', P' \rangle$ are permutative variants if $d_1 = d'_1$, $d_2 = d'_2$, $\ell = \ell'$, and there is a bijection $\pi$ on $\Sigma_m$ with $m = k$ if $\ell = k$ and $m = \ell - 1$ otherwise such that

$$P' = \pi(P) = \{[\theta \circ \pi, c, b_1, b_2] | [\theta, c, b_1, b_2] \in P\}.$$

Clearly if the configurations of two states are permutative variants, the subautomata consisting of reachable states from those states are isomorphic under the
Figure 3: DFA $M_2$ that accepts binary solvable canonical cryptarithm sequences. The initial state is $q_0 = 1$ and the accepting state is $f_1 = 28$. The other accepting state $f_2$ is missing in $M_2$. 
Figure 4: DFA with permutation edges for binary solvable canonical cryptarithm sequences, where black trigram labels are with the identity permutation $\iota$ and red boxed trigram labels are with $\{a \rightarrow b, b \rightarrow a\}$. 
permutation. This allows us to reduce the size of the automaton by merging those states. In our new DFA $\overline{M}_k$, each transition edge has two labels: one is a trigram as before and the other is a permutation on $\Sigma_k$. After passing a transition edge labeled with a permutation $\pi$, we will follow transition edges by replacing each letter in accordance with $\pi$. Figure 4 shows $\overline{M}_2$.

We formally define this new kind of DFAs with edges labeled with a letter and a permutation. A DFA with permutation edges is a sextuple $M = \langle Q, \Sigma, \delta, \gamma, q_0, F \rangle$, where $\delta$ and $\gamma$ are partial functions $Q \times \Sigma \rightarrow Q$ and $Q \times \Sigma \rightarrow \Pi$, respectively, where $\Pi$ is the set of all permutations over $\Sigma$, such that the domains of $\delta$ and $\gamma$ coincide. For $x \in \Sigma$, $w \in \Sigma^*$ and $q \in Q$, define $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$ and $\hat{\gamma} : Q \times \Sigma^* \rightarrow \Pi$ by

$$
\begin{align*}
\hat{\delta}(q, \varepsilon) &= q, \\
\hat{\gamma}(q, \varepsilon) &= \iota, \\
\hat{\delta}(q, wx) &= \delta(\hat{\delta}(q, w), \hat{\gamma}(q, w)(x)), \\
\hat{\gamma}(q, wx) &= \gamma(\hat{\delta}(q, w), \hat{\gamma}(q, w)(x)) \circ \hat{\gamma}(q, w),
\end{align*}
$$

where $\iota$ is the identity. The strings that the automaton $M$ accepts are those $w \in \Sigma^*$ such that $\hat{\delta}(q_0, w) \in F$. Mealy machines with permutation edges can also be defined, where outputs may depend on the current state, permutation and next input letter.

We modify $M_k$ to $\overline{M}_k$ by merging states that are permutative variants into a representative and adding appropriate permutation labels to edges. We choose as representative the state that is lexicographically earliest among permutative variants with respect to their representations in the implementation. In our cryptarithm DFAs with permutation edges, permutation labels are defined on letters in $\Sigma_k$ and homomorphically extended to trigrams on $\Sigma_k \cup \{\$\}$, where $\$\$ is always mapped to $\$\$ itself. Algorithm 1 shows the pseudo code for constructing $\overline{M}_k$.

We modify $M_k$ to $\overline{M}_k$ by merging states that are permutative variants into a representative and adding appropriate permutation labels to edges. We choose as representative the state that is lexicographically earliest among permutative variants with respect to their representations in the implementation. In our cryptarithm DFAs with permutation edges, permutation labels are defined on letters in $\Sigma_k$ and homomorphically extended to trigrams on $\Sigma_k \cup \{\$\}$, where $\$\$ is always mapped to $\$\$ itself. Algorithm 1 shows the pseudo code for constructing $\overline{M}_k$. An assignment $\theta \circ \hat{\gamma}(q_0, w)$ is a solution of a cryptarithm sequence $w\$\$\$\$ if and only if $[\theta, 0, 1, 1] \in P$ for the configuration $(d_1, d_2, \ell, P)$ of the state $\hat{\delta}(q_0, w)$.

### 3.3 Comparison of Naive and Compressed Cryptarithm DFAs

Table 1 compares the numbers of states and edges of $M_k$ and $\overline{M}_k$. We succeeded in calculating the automata for $k \leq 7$ but gave up for $k \geq 8$ due to the large time calculation and big memory consumption. For the purpose of reference, we also show the number of states of $\min(M_k)$, the minimized version of $M_k$. Note that minimization loses the information of possible solutions for cryptarithm sequences and therefore $\min(M_k)$ cannot be used as a solver. Our compression technique achieves a more compact representation than the classical state minimization technique for solvable cryptarithm sequences, while keeping the solver function. Table 2 compares the time and space used to construct $M_k$ and $\overline{M}_k$. Our implementation was compiled with Go 1.10 on Ubuntu 14.04 LTS with CPU Xeon E5-2609 2.4GHz and 256 GB memory. To construct $\overline{M}_k$ requires much shorter time and smaller memory than $M_k$ for all $k$. 

9
Algorithm 1 Constructing $M_k$

1: let $q_0 := \langle 0, 0, 1, \{\emptyset, 0, 0, 0\} \rangle$ and $Q := \{q_0, f_1, f_2\}$;
2: push $q_0$ to the stack;
3: while the stack is not empty do
4: pop the top element $q$ from the stack;
5: for each trigram $u$ on $\Sigma_k \cup \{\$$\} do
6: if there is a configuration $q'$ such that $q \Rightarrow q'$ then
7: if $u = $$$ then
8: if $q' = \langle d_1, d_2, \ell, P \rangle$ with $|P| = 1$ then
9: add an edge from $q$ to $f_1$ with label $(u, \iota)$;
10: else
11: add an edge from $q$ to $f_2$ with label $(u, \iota)$;
12: end if
13: else
14: let $\pi \in \Pi_{\Sigma_k}$ be such that $\pi(q') = q''$ is the canonical form of $q'$;
15: if $q'' \notin Q$ then
16: add $q''$ to $Q$ and push $q''$ to the stack;
17: end if
18: add an edge from $q$ to $q''$ with label $(u, \pi)$;
19: end if
20: end if
21: end for
22: end while
23: return $\langle Q, \Sigma_k, \delta, \gamma, q_0, f_1, f_2 \rangle$;

| Table 1: Numbers of states and edges of cryptarithm automata |
|-------------------------------------------------------------|
| Base $k$ | $M_k$ | $\tilde{M}_k$ | $\min(M_k)$ | $M_k$ | $\tilde{M}_k$ | $\min(M_k)$ |
|-----------|-------|---------------|--------------|-------|---------------|--------------|
| 2         | 28    | 15            | 27           | 112   | 111           | 111          |
| 3         | 110   | 27            | 93           | 1032  | 985           | 985          |
| 4         | 859   | 163           | 607          | 17662 | 16602         | 16602        |
| 5         | 10267 | 1061          | 6589         | 350019| 330297        | 330297       |
| 6         | 370719| 17805         | 248192       | 23508141| 22673144     | 22673144     |
| 7         | 30909627| 472518       | 48635469     | 3017993409| –             | –            |

| Table 2: Used computational resources for constructing cryptarithm automata |
|--------------------------------------------------------------------------|
| Base $k$ | Time (sec.) | $M_k$ | $\tilde{M}_k$ | Space (MB) | $M_k$ |
|-----------|-------------|-------|---------------|------------|-------|
| 2         | < 0.01     | < 0.01| < 0.01       | < 0.01     | < 2   |
| 3         | < 0.01     | < 0.01| < 0.01       | < 0.01     | < 2   |
| 4         | < 0.01     | < 0.01| < 0.01       | < 0.01     | < 2   |
| 5         | < 0.01     | < 0.01| < 0.01       | < 0.01     | < 2   |
| 6         | < 0.01     | < 0.01| < 0.01       | < 0.01     | < 2   |
| 7         | < 0.01     | < 0.01| < 0.01       | < 0.01     | < 2   |

10
Table 3: Numbers of states of $\widetilde{M}_{k,s}$

| $k \setminus s$ | 2  | 3  | 4  | 5  | 6  |
|-----------------|----|----|----|----|----|
| 7               | 19 | 271| 4098| 57356| 390370|
| 8               | 23 | 302| 5623| 133385| 2180416|
| 9               | 20 | 313| 6688| 220255| 6279611|
| 10              | 19 | 320| 7507| 328959| 13920691|

Table 4: Construction times of $\widetilde{M}_{k,s}$ (sec.)

| $k \setminus s$ | 2  | 3  | 4  | 5  | 6  |
|-----------------|----|----|----|----|----|
| 7               | < 0.01 | 0.01 | 0.6 | 10 | 90 sec. |
| 8               | < 0.01 | 0.03 | 1.4 | 34 | 11 min. |
| 9               | < 0.01 | 0.04 | 3.0 | 107 | 52 min. |
| 10              | < 0.01 | 0.07 | 6.1 | 307 | 210 min. |

3.4 Cryptarithms with Limited Number of Letters

As we have observed in the previous subsection, we were unable to compute $M_k$ and $\widetilde{M}_k$ for $k \geq 8$. On the other hand, there are many interesting decimal cryptarithms in the real world that do not involve all the ten numerals. It is still interesting to construct a DFA $\tilde{M}_{k,s}$ that accepts all and only base-$k$ solvable cryptarithm sequences over $\Sigma_s$ for $s \leq k$. This can be achieved by a slight modification on Algorithm 1, where we refrain from making transition edges whose label includes forbidden letters not in $\Sigma_s$. In addition, when $s = k - 1$, we need to give up “promotion” of an assignment with domain $\Sigma_{k-1}$ to its extension with domain $\Sigma_k$. This results actually in a simpler construction algorithm.

Tables 3 and 4 show the numbers of states and the computation times of the construction of $\widetilde{M}_{k,s}$ for $7 \leq k \leq 10$ and $2 \leq s \leq 6$.

4 Analysis of Cryptarithms

Cryptarithm automata $M_k$, $\tilde{M}_k$ and $\widetilde{M}_{k,s}$ can be used as cryptarithm puzzle solvers as we have described in the previous section. Moreover, they can be used as complete catalogues of solvable cryptarithms. For example, one can count the number of base-$k$ solvable cryptarithms of size $n$ and one can enumerate the base-$k$ solvable cryptarithm sequences by the length-lexicographic order.

4.1 Counting Solvable Cryptarithms

The number $F_k(n)$ of base-$k$ uniquely solvable cryptarithms of size $n$ is the number of the paths of length $n + 1$ from the initial states to the accepting state $f_1$ in $\tilde{M}_k$. The number $G_k(n)$ of (not necessarily uniquely) solvable cryptarithms is obtained by adding the number of paths to the accepting state $f_2$ to this number. Those numbers can be calculated by the standard technique using the adjacency matrix $A_k$ of the automaton in $O(m_k^3 \log n)$ time, where $m_k$ is the number of states of the automaton (i.e., $m_k$ is the number of rows (columns) of $A_k$). Table 5 summarizes the numbers of uniquely and not necessarily uniquely solvable cryptarithms for $n \leq 8$. Note that there are no difficulties to compute $F_n(k)$ and $G_n(k)$ for bigger $n$. Although we have computed $\bar{F}_7$ and $\bar{G}_7$ even for small numbers $n$ by multiplying the adjacency matrices due to the size of the matrices. Moreover, for $k = 2, 3$, we obtain $F_k(n)$ and $G_k(n)$ as explicit formulas of $n$ using Mathematica as
follows.

\[ F_2(n) = G_2(n) = 6 \times 4^{n-2} - 3 \times 2^{n-2} \]
\[ F_3(n) = 4 \times 9^{n-1} - 4 \times 5^{n-1} - 3^{n-1} \]
\[ G_3(n) = 4 \times 9^{n-1} - 2 \times 5^{n-1} - 3^{n-1} \]

Unfortunately, Mathematica \textsuperscript{TM} returned no answers for bigger \( k \geq 4 \) within 3 days on our environment.

Table 5: The numbers \( F_k(n) \) and \( G_k(n) \) of uniquely and not necessarily uniquely solvable cryptarithms, respectively. Among those, numbers show \( n \) with bold figures were not known in [5].

\begin{table}[h]
\begin{tabular}{cccccccc}
\hline
\( k \setminus n \) & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
2 & 0 & 3 & 18 & 84 & 360 & 1488 & 6048 & 24384 \\
3 & 1 & 19 & 233 & 2443 & 23825 & 223939 & 2063993 & 18821563 \\
4 & 1 & 46 & 1200 & 431424 & 7326008 & 121032266 & 1970599868 & \\
5 & 0 & 42 & 3190 & 106663574 & 2797440502 & 71604333066 & \\
6 & 0 & 10 & 3470 & 336367 & 18978996 & 84749530 & 33983003374 & \\
\hline
\end{tabular}
\end{table}

\begin{table}[h]
\begin{tabular}{cccccccc}
\hline
\( k \setminus n \) & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
2 & 0 & 3 & 18 & 84 & 360 & 1488 & 6048 & 24384 \\
3 & 1 & 23 & 265 & 24913 & 229703 & 2093785 & 18973439 & \\
4 & 2 & 69 & 1463 & 456639 & 7561377 & 123194460 & 1990281467 & \\
5 & 2 & 115 & 4622 & 148483 & 4184478 & 110899540 & 2852251360 & 72299094358 & \\
6 & 2 & 123 & 8650 & 498307 & 22931188 & 933488391 & 35745728867 & 1327783229135 & \\
\hline
\end{tabular}
\end{table}

4.2 Enumerating and Indexing Cryptarithms

By depth-first search on a cryptarithm automaton, one can enumerate all the base-\( k \) (uniquely) solvable cryptarithm sequences by length-lexicographic order. Moreover, from an index number \( i \), one can efficiently give the \( i \)th (uniquely) solvable cryptarithm sequence. This can be computed in \( O(m_3^4 n \log n) \) time, where \( n \) is the length of the \( i \)th cryptarithm, using powers of the adjacency matrix \( A_k \). Conversely, from a solvable cryptarithm of length \( n \), the indexing number of it can be computed in \( O(m_3^4 n \log n) \) time as well. As examples, the first 30 ternary solvable cryptarithm sequences are given below.

\begin{itemize}
  \item aab$$, aabbc$$, aab$b$$, aab$a$$, aab$ba$$,
  \item aab$bb$$, aaba$a$$, aaba$b$$, aabb$a$$, aabb$b$$,
  \item aba$aa$$, aba$c$$, abaaac$$, abacca$$, abbb$a$$,
  \item abbb$cc$$, abc$c$$, abcc$b$$, abc$ca$$, abc$ba$$,
  \item abc$ab$$, abc$bc$$, abcb$b$$, abc$b$$, aaaaab$bc$$,
  \item aabbb$b$$, aaaaab$bc$,$ aaaaabc$bc$$, aaaaabc$c$$, aabaab$bb$$
\end{itemize}

5 Conclusions and Discussions

This paper proposed an algorithm to construct a DFA that accepts solvable cryptarithms under the base-\( k \) numeral system. Our construction method in-
volves a technique to reduce the number of states more significantly than the classical minimization of DFAs by enriching transition edge labels. We implemented the algorithm and constructed cryptarithm DFAs for $2 \leq k \leq 7$. Moreover, by limiting the number of letters used in cryptarithms to $s \leq k$, we managed to construct DFAs for even bigger bases. Using those automata, we demonstrated that the numbers of base-$k$ solvable cryptarithms of $n$ digits are computable for $2 \leq k \leq 6$.

Our compression technique is based on the symmetry among assignments. Another type of symmetry is found between the first and second summand terms. It is future work to take advantage of this type of symmetry to reduce the size of cryptarithmetic DFAs. We are also interested in applying our DFAs for generating alphametics, which are cryptarithms with meaningful words.

Acknowledgments

We thank to Kaizaburo Chubachi for assisting us in some of the experiments. We also appreciate helpful comments by anonymous reviewers of CIAA 2018. The work is supported in part by KAKENHI 15H05706 and 18K11153.

References

[1] Reza Abbasian and Masoud Mazloom. Solving cryptarithmetic problems using parallel genetic algorithm. In Second International Conference on Computer and Electrical Engineering, pages 308–312, 2009.

[2] Truman Collins. Alphametic puzzle solver. http://www.tkcs-collins.com/truman/alphamet/alpha_solve.shtml accessed: July 20, 2018.

[3] dCode. Cryptarithmetic solver - alphametic puzzle - online software tool. https://www.dcode.fr/cryptarithm-solver accessed: July 20, 2018.

[4] Henry E. Dudeney. Strand Magazine, volume 68, pages 97–214. George Newnes, July 1924.

[5] Hiroshi Endoh. Automata-theoretic approaches to puzzle analysis. Master’s thesis, Graduate School of Information Sciences, Tohoku University, 2013. (in Japanese).

[6] Hiroshi Endoh, Kazuyuki Narisawa, and Ayumi Shinohara. An automaton theory approach for analyzing alphametic. In Proceedings of the 16th Game Programming Workshop, pages 54–61, 2011. (in Japanese).

[7] D Eppstein. On the NP-completeness of cryptarithms. ACM SIGACT News, 18(3):38–40, 1987.

[8] Donald E. Knuth. The Art of Computer Programming, volume 4A. Addison-Wesley, 2017.

[9] Keith Luoma. Cryptarithms: A non-programming approach using Excel. Spreadsheets in Education (eJSiE), 9(2):6, 2016.
[10] Yuki Nozaki, Diptarama Hendrian, Ryo Yoshinaka, and Ayumi Shinohara. Enumeration of cryptarithms using deterministic finite automata. In Proceedings of the 23rd International Conference on Implementation and Application of Automata, pages 286–298, 2018.

[11] Naoyuki Tamura. Cryptarithmetic puzzle solver. http://bach.istc.kobe-u.ac.jp/llp/crypt.html accessed: July 20, 2018.